Mining Reference Tables for Automatic Text Segmentation

Eugene Agichtein*
Columbia University
eugene@cs.columbia.edu

Venkatesh Ganti
Microsoft Research
vganti@microsoft.com

ABSTRACT

Automatically segmenting unstructured text strings into structured records is necessary for importing the information contained in legacy sources and text collections into a data warehouse for subsequent querying, analysis, mining and integration. In this paper, we mine tables present in data warehouses and relational databases to develop an automatic segmentation system. Thus, we overcome limitations of existing supervised text segmentation approaches, which require comprehensive manually labeled training data. Our segmentation system is robust, accurate, and efficient, and requires no additional manual effort. Thorough evaluation on real datasets demonstrates the robustness and accuracy of our system, with segmentation accuracy exceeding state of the art supervised approaches.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, design, performance, experimentation

Keywords

Text segmentation, machine learning, text management, information extraction, data cleaning

1. INTRODUCTION

Information in unstructured text needs to be converted to a structured representation to enable effective querying and analysis. For example, addresses, bibliographic information, personalized web server logs, and personal media filenames are often created as unstructured strings that could be more effectively queried and analyzed when imported into a structured relational table. Building and maintaining large data warehouses by integrating data from such sources requires automatic conversion, or *segmentation* of text into structured records of the target schema before loading them into relations.

Informally, the problem of segmenting input strings into a structured record with a given n-attribute schema is to partition the string into n contiguous sub-strings and to assign each sub-string to a unique attribute of the schema. For instance, segmenting the input string "Segmenting text into structured records V. Borkar, Deshmukh and Sarawagi SIGMOD" into a bibliographic record with schema [Authors, Title, Conference, Year] requires the assignment of the sub-string "V. Borkar, Deshmukh and Sarawagi" to the Authors attribute, the sub-string "Segmenting text into structured records" to the Title attribute, "SIGMOD" to the Conference attribute, and the NULL value to the Year attribute.

Current techniques for automatically segmenting input strings into structured records can be classified into *rule-based* and *supervised model-based* approaches. Rule-based approaches require a domain expert to design a number of rules and maintain them over time. This approach does not scale as deployment for each new domain requires designing, crafting, deploying, and maintaining a new set of rules. Supervised approaches alleviate this problem by automatically learning segmentation models from training data consisting of input strings and the associated correctly segmented tuples [5]. However, it is often hard to obtain training data, especially data that is comprehensive enough to illustrate all features of test data. This problem is further exacerbated when input test data as in our target data warehouse scenario is error prone; it is much harder to obtain comprehensive training data that effectively illustrates all kinds of errors. These factors limit the applicability and the accuracy of supervised approaches.

In this paper, we exploit *reference relations*—relations consisting of clean tuples—in typical data warehouse environments. For example, most data warehouses include large customer and product tables, which contain examples that are specific to the domain of interest. Reference relations can be a source of rich vocabularies and structure within attribute values. Our insight is to exploit such widely available reference tables to automatically build a robust segmentation system. Note that we do not attempt to *match* the newly segmented records with the tuples in the reference table. Record matching and deduplication are different problems and are not the focus of this paper.

Our approach relies only on reference tables: we do not need the association between unsegmented input strings and the corresponding segmented strings (labeled data) that supervised systems require for training. Current operational data warehouses and databases do not maintain such associations between input data and the actual data in the database, because input data goes through a series of potentially complex transformations before it is stored in the database.

Building segmentation models from clean standardized tuples in a large reference table requires three critical challenges to be addressed. The first challenge is that information in reference relations is typically clean whereas input strings may contain a variety of errors: missing values, spelling errors, use of inconsistent abbreviations, extraneous tokens, etc. [18]. Therefore, the challenge is to learn from clean reference relations, segmentation models that are robust to input errors. The second challenge is that we do not know the order in which attribute values in an input string are specified. In the data warehouse maintenance scenario, the order in which attribute values are concatenated by an address data source may be [State, Zipcode, City, Name, Address] while another source may concatenate it in the order [Name, Address, City, Zip, State]. Another common example is bibliographic data: some sources may order attributes for

*Work done at Microsoft Research.

KDD'04, August 22–25, 2004, Seattle, Washington, USA.

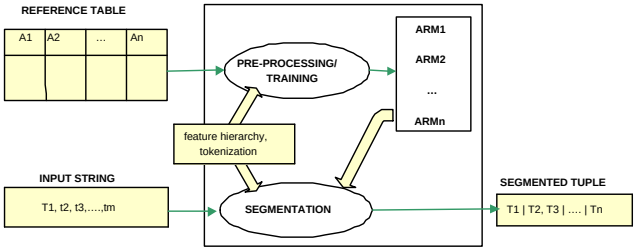

Figure 1: Architecture of the CRAM system

each article as [authors, title, conference, year, pages] while other sources may order them as [title, authors, conference, pages, year]. Therefore, for an unsupervised segmentation system to be deployed over a variety of data sources it has to deal with differences in input orders by automatically detecting the attribute order. The third challenge is that reference tables can usually be very large and consist of millions of tuples. Therefore, to effectively exploit large vocabularies and rich structural information in large reference tables the algorithm for building a segmentation model from large reference tables has to be efficient and scalable. Note that training efficiency is not usually an issue with supervised approaches: manually labeling data is time-consuming and expensive and hence training datasets are much smaller than sizes of available reference tables.

Due to the above challenges, it is not possible to directly adapt existing supervised approaches to the "learn from reference table only" scenario. First, we cannot use data in a reference table to prepare a training dataset for current supervised approaches because we do not know the order in which attribute values would be observed in test input strings. In fact, current supervised models heavily rely on the knowledge of input attribute order to achieve high accuracy [5]. Second, data in reference tables is usually clean whereas input data observed in practice is dirty. And, making a design goal of robustness to input errors during training itself improves the overall segmentation accuracy. Third, current supervised text segmentation systems are designed to work with small labeled datasets (e.g., by using cross validation) and hence tend not to scale to training over large reference tables.

The architecture of our automatic segmentation system *CRAM*[1] is shown in Figure 1. We adopt a two-phased approach. In the first *pre-processing* phase, we build an *attribute recognition model* over each column in the reference table to determine the probability with which a (sub-)string belongs to that column. This process can be customized with a domain-specific tokenizer and a feature hierarchy (i.e., token classification scheme such as "number", "delimiter", etc.). For example, the recognition model on the "Zip Code" column of an address relation indicates that the probability of a five-digit number (e.g., *12084*) being a valid zip code is *0.95* whereas that of a word like *Timbuktoo* is only *0.005*. Models on all columns can be used together to determine the best segmentation of a given input string into sub-strings. In the second run-time *segmentation* phase, we segment an input string s into its constituent attribute values $s_1, \ldots, s_n$ and assign each s_i to a distinct column such that the quality of the segmentation is the best among all possible segmentations.

Contributions: In this paper, we develop a robust segmentation system which can be deployed across a variety of domains because it only relies on learning the internal attribute structure and vocabularies from widely available reference tables. We introduce robust and accurate attribute recognition models to recognize attribute values from an attribute, and develop efficient algorithms for learning these models from large reference relations (Section 4). We then develop an algorithm that effectively uses robust attribute recognition models to accurately and efficiently segment input strings (Section 5). Finally, we present a thorough experimental evaluation of CRAM on real datasets from a variety of domains and show that our domain-independent segmentation system outperforms current state of the art supervised segmentation system (Section 6).

[1]CRAM: **C**ombination of **R**obust **A**ttribute **M**odels

2. RELATED WORK

Extracting structured information from unstructured text has been an active area of research, culminating in a series of Message Understanding Conferences (MUCs) [2, 17] and more recently ACE evaluations [21]. Named entity recognition systems extract names of entities from the natural language text (e.g., PERSON names, LOCATIONs, ORGANIZATIONs). Detecting entities in natural language text typically involves disambiguating phrases based on the actual words in the phrase, and the text context surrounding the candidate entity. Explored approaches include hand-crafted pattern matchers (e.g., [1]), rule learners (e.g., [3, 23]), and other machine learning approaches (e.g., [11]). In contrast, strings in our segmentation scenario are not necessarily natural language phrases, and do not contain surrounding contextual text.

Work on *wrapper induction* (e.g., [12, 19, 24]) exploits layout and formatting to extract structured information from automatically generated HTML pages, as well as heuristics and specialized feature hierarchies to extract boundaries between records [14]. In contrast, the input strings in our problem are short and have no obvious markers or tags separating elements. Furthermore, all of the string except for the delimiters must be assigned to some of the target attributes. As previously shown by Borkar et al. [5], the required techniques therefore differ from traditional named entity tagging and from wrapper induction.

Hidden Markov Models (HMMs) are popular sequential models [4], and have been used extensively in information extraction and speech recognition. Since the structure of HMMs is crucial for effective learning, optimizing HMM structure has been studied in the context of IE and speech recognition (e.g., [15, 27].

The problem of robustness to input errors has long been studied in speech recognition. Some approaches include filtering out noise during pre-processing and training the system in artificially noisy conditions (error injection) [13]. Noise filtering from speech recognition cannot be adapted to text segmentation directly, since in our scenario the input errors are not separable from actual content. To the best of our knowledge, we are the first to explicitly address input errors in the context of text segmentation.

Past work on automatic text segmentation most closely related to ours is the DATAMOLD system [5] and related text segmentation approaches (e.g., [22] and [28]). These are supervised approaches and hence share the limitations discussed earlier.

In this paper, we present a novel, scalable, and robust text segmentation system CRAM. Our system requires only the target reference table and no explicitly labelled data to build accurate and robust models for segmenting the input text strings into structured records.

3. SEGMENTATION MODEL

In this section, we define the segmentation problem and introduce preliminary concepts required in the rest of the paper. We define the "quality" of segmentation of an input string into attribute value sub-strings as a function of the "probabilities" of these sub-strings belonging to corresponding attribute domains. Our goal now is to select the segmentation with the best overall probability. The probability of a sub-string assignment to an attribute domain is estimated by a model called *Attribute Recognition Model* (described below) associated with each attribute. Consider an input string "Walmart 20205 S. Randall Ave Madison 53715 WI" which has to be segmented into

Organization Name, Street Address, City, State, and Zipcode attribute values. For example, the attribute recognition model for *Organization Name* may assign respective probabilities of 0.9 and 0.15 to sub-strings "walmart" and "walmart 20205." If the combination (say, product) of individual probabilities of the segmentation "walmart" as *Organization Name*, "20205 s. randall ave" as *Street Address*, "madison" as *City*, "53715" as *Zipcode*, and "WI" as *State* has the highest numeric value, we output this segmentation of the given input string. The combination of probabilities has to be invariant to changes in the order in which attribute values were concatenated. This is in contrast to traditional (tagging) approaches, which rely heavily on the order in which attribute values are concatenated (e.g, [5, 10]). In the rest of the paper, we use R to denote a reference relation with string-valued attributes (e.g., varchar types) $A_1, \ldots, A_n$.

Attribute Recognition Model (ARM): An attribute recognition model ARM_i for the attribute A_i is a model for the domain of A_i such that $ARM_i(r)$ for any given string r is the probability of r belonging to the domain of A_i.

Optimal segmentation of an input string: Let R be a reference relation with attributes $A_1, \ldots, A_n$ and $ARM_1, \ldots, ARM_n$ be their respective attribute recognition models. Given an input string s, the segmentation problem is to partition s into $s_1, \ldots, s_n$ and to map them to distinct attributes $A_{s1}, \ldots, A_{sn}$ such that $\prod_i^n \{ARM_{si}(s_i)\}$. is maximized over all valid segmentations of s into n substrings. A similar model that assumes partial independence of attribute segmentations was independently developed in [9].

Note that the order of attributes $A_{s1}, \ldots, A_{sn}$ may be different from the order of the attributes $A_1, \ldots, A_n$ specified in the reference table. Attribute constraints for R (e.g., maximum attribute length) can also be incorporated into this model. And, information about the order in which attribute values are usually observed can also be incorporated. For example, if we know that the street address value usually follows the name attribute value, we can potentially bias the assignment of consecutive sub-strings, say "walmart" and "20205 S. Randall Ave," to name and to street address attributes, respectively.

3.1 Preliminaries

Let *tok* be a tokenization function which splits any string into a sequence *tok(s)* of tokens based on a set of user-specified delimiters (say, whitespace characters). The token set of a string s is the set of all tokens in s. For example, *tok(v[1])* of the tuple [Boeing company, Seattle, WA, 98004] is [boeing, company], and {boeing, company} is the token set. Observe that we ignore case while generating tokens. The *dictionary* D_i of the attribute A_i of R is the union of token sets of all attribute values in the projection $R[i]$ of R on A_i. In the rest of the paper, we assume that strings can be segmented only at token boundaries.

Hidden Markov Models: A Hidden Markov Model (HMM) is a probabilistic finite state automaton encoding the probability distribution of sequences of symbols each drawn from a discrete dictionary [25]. Figure 3(a) shows an example HMM. For a sequence s of symbols each drawn from the probability distribution encoded by a HMM, we can compute the probability of observing s. A HMM comprises a set of states and a dictionary of output symbols. Each state can emit symbols from the dictionary according to an *emission* probability distribution for that state and pairs of states are connected by directed edges denoting transitions between states. Further, edges are associated with *transition* probabilities. HMMs have two special states: a *start* state and an *end* state. The probability of observing a string $s = o_1, \ldots, o_k$ of symbols drawn from the dictionary, is the sum of probabilities of all paths from the start state to the end state with k transitions. The probability of any path p is the product of all transition probabilities on each transition in p and the emission probabilities of observing the i^{th} symbol o_i at the i^{th} state on p. The path with the highest probability is usually considered the path that generated the string s. The set of states and the set of transitions constitute the *topology* of a HMM. For any given application, the topology is usually fixed a priori.[2] The emission and transition probabilities are then learned during a training phase over the training data.

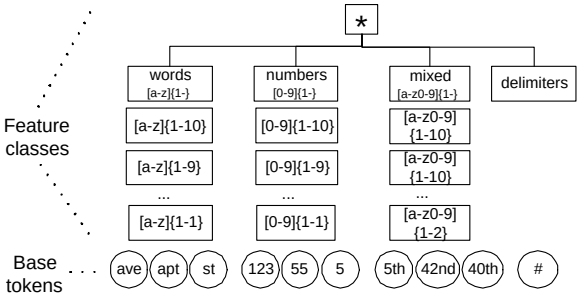

Figure 2: A sample generalized dictionary instantiated for the *Street Address* column

Feature Hierarchy: A HMM built over a dictionary of an attribute cannot be directly used for computing probabilities of sequences with unknown tokens. However, the set of base tokens in a dictionary can be generalized to recognize unknown tokens [5]. For example, it may be sufficient to see a 5-digit number optionally followed by a 4-digit number to recognize zip codes. The successive generalization of features (e.g., from 5-digit numbers to all numbers) is usually encoded as a *feature hierarchy*. In this paper, we use a hierarchy which is similar to the feature hierarchy employed in [5]. In our hierarchy the lower levels are more specific than higher levels. At the top level there is no distinction among symbols; at the next level they are divided into classes "words," "numbers," "mixed," and "delimiters." "Words," "numbers" and "mixed" are then divided into sub-classes based on their lengths. For example, the class of words consisting of 10 or less characters (denoted `[a-z]{1-10}`) is above the class of words consisting of 9 or less characters (denoted `[a-z]{1-9}`). All *base* tokens are at the leaf levels of the feature hierarchy. To distinguish base tokens from the generalized elements in the discussion, we refer to the non-leaf elements in the feature hierarchy as *feature classes*. We say that that a token t *minimally belongs* to a feature class f if t belongs to f but not to any feature class that is a descendant of f. For example, the Zipcode value 21934 is said to minimally belong to the feature class 5-digit numbers.

Observe that it is possible to input domain-specific feature hierarchies to CRAM in the pre-processing phase, but as we show experimentally, the default hierarchy is sufficient for a wide range of domains.

Generalized Dictionary: The *generalized dictionary* consists of all elements in the feature hierarchy in addition to the dictionary of base tokens. Formally, the *generalized dictionary* of an attribute A_i in R is the union of the dictionary D_i of A_i and the set of feature classes in the feature hierarchy. Henceforth, we use the term *dictionary* to denote the generalized dictionary unless otherwise specified. Note that while the feature hierarchy itself is fixed across domains, the generalized dictionary is instantiated for each attribute automatically during training from the provided reference tables. A sample of a

[2]Recently, techniques based on cross validation were developed to identify a good topology from among a target class of topologies [15].

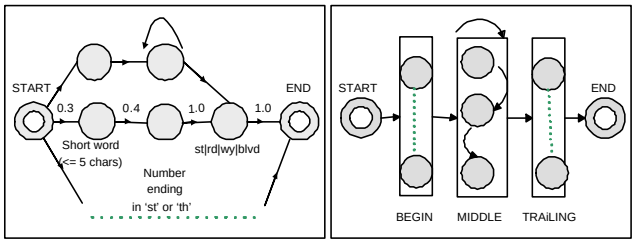

Figure 3: Example HMM Model (a), and our ARM Model (b)

generalized dictionary instantiated for the *Street Address* attribute is shown in Figure 2.

4. ATTRIBUTE RECOGNITION MODELS

In this section, we discuss the efficient construction of robust attribute recognition models from a reference relation. Recall that an attribute recognition model assigns probability with which a string or a sequence of tokens belongs to the attribute domain. Therefore, we adopt the class of hidden markov models (HMMs), a popular class for modelling sequences of elements, for instantiating attribute recognition models. Instantiating an HMM requires us to define (i) the topology consisting of a set of states and the set of transitions among them, and (ii) the emission probabilities at each state and the transition probabilities between states. In this section, we describe the topology we adopt for instantiating ARMs and the computation of emission and transition probabilities. Our primary focus in this design is (i) to improve the robustness of segmentation to input errors, and (ii) to develop an efficient and scalable algorithm for building robust attribute recognition models.

We now discuss the intuition behind the principle, which we call *specificity*, that we exploit to make ARMs more robust to input errors. A more "specific" attribute recognition model assigns higher probabilities only to very few selective token sequences. ARMs can be specific in three aspects: *positional specificity*, *sequential specificity*, and *token specificity*. We illustrate these notions with an example. Consider an HMM example in Figure 3(a). That a token in the street address value ending in "th—st" can only be in the *second* position is an example of positional specificity. The probability of acceptance is much lower if such a token ending in "th—st" appears in the third position instead of the second position. A token ending in "th" or "st" can only *follow* a short word and tokens "st, rd, wy, blvd" can only *follow* a token ending in "th" or "st" are examples of sequential specificity. Note that sequential specificity stresses the sequentiality—of a token following another—and is orthogonal to the positionality of the tokens. That the last state can only accept one of "st, rd, wy, blvd" is an example of token specificity.

Even though highly specific models may be required for some applications, attribute recognition models need only be specific to the extent of being able to identify an attribute value as belonging to the correct attribute and distinguish it from other domains. Moreover, being overly specific in recognizing attribute values may cause the attribute recognition model to reject (i.e., assign very low probability) attribute values with errors, thereby resulting in incorrect segmentations. Often, we can trade specificity off for achieving robustness to input errors. However, the challenge is to make the tradeoff without losing segmentation accuracy and at the same time being able to build the model efficiently.

In this section, we instantiate an accurate and robust attribute recognition model. We first describe the topology of ARMs, and then describe the techniques for relaxing sequential and token specificity. Finally, we describe a procedure for learning such a model from a large reference table.

4.1 ARM Topology

The topology of a hidden Markov model, consisting of the set of states and valid transitions between these states, has a big impact on the accuracy of the model. Recently, techniques based on cross-validation and stochastic optimization have been proposed to automatically decide good topologies [16, 5]. However, these structure optimization techniques require several scans of the training data, and hence are slow when training data is large. Moreover, these techniques—if trained on "clean" data—can result in positionally specific topologies (e.g., Figure 3(a)), conditioned to accept tokens in specific positions, say, first or third or last. Such topologies, even though accurate for segmenting clean input data, can be less robust towards erroneous input. In this section, we discuss a statically fixed topology that (i) enables efficient model building and (ii) relaxes positional specificity in favor of robustness to input errors.

We observe that collapsing positional information into a small number of distinct categories results in a more flexible, compact, and robust ARM topology. More specifically, we categorize tokens in attribute values into three positions: *Beginning*, *Middle*, and *Trailing* positions, resulting in what we will call the *BMT* topology, shown in Figure 3(b).[3] In the example "57th nw 57th st" string corresponding to a street address, the token "57th" is categorized as beginning token, "st" as the trailing token, and the rest as middle tokens.

Collapsing token positions into these three categories, we gain efficiency while building ARMs by avoiding a computationally expensive search for "optimal" topology. We also gain robustness to several common types of input errors—token deletions, token insertions, and token re-orderings. For example, the probability of observing a token $57th$ as the second or third token in "nw 57th 57th st" is the same for both occurrences of the token. Observe that we still are specific about the positionality of the beginning and trailing tokens because these are critical for correctly recognizing boundaries between attribute values.[4] And, by not grouping boundary tokens with the middle tokens, we are able to collect more specific statistics on the emission and transition probabilities for boundary tokens. Our empirical evaluation shows that the BMT topology captures the salient structure required for robust segmentation, and performs as well as, and sometimes better than, other topologies involving multiple middle positions.

The categorization of tokens into positions induces a categorization on the (generalized) dictionary of an attribute. The dictionary D_i corresponding to an attribute A_i is now categorized into the beginning, middle, and trailing dictionaries D_i^B, D_i^M, and D_i^T. For example, a token occurring in the beginning position of an attribute value of any tuple in R belongs to the D_i^B dictionary.

Set of States and Possible Transitions: The set of states in an ARM model is also categorized into *beginning, middle, and trailing* states. Each category consists of a state s for each element e (base token or feature class) in the corresponding categorized (generalized) dictionary, and s emits only e with non-zero probability. The union of all three categorized states along with the special *start* and *end* states constitutes the set of states in an ARM. The broad structure of the set of allowed transitions is shown in Figure 3(b). Each category—

[3]The categorization can be generalized to more sophisticated topologies with multiple middle positions. However, our experiments (not included due to space constraints) showed that the simplest BMT topology is equally good.

[4]This intuition is corroborated by the human ability to recognize words even if all but boundary characters are randomly rearranged: "Aoccdrnig to a rscheearch at an Elingsh uinervtisy, it deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt tihng is taht frist and lsat ltteer is at the rghit pclae. The rset can be a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae we do not raed ervey lteter by it slef but the wrod as a wlohe."

beginning, middle, and trailing—may consist of several states in the HMM but the transitions among these state categories are restricted to non-backward transitions, as indicated by the arrows in Figure 3(b). That is, beginning states can only transition to either middle or to trailing or to the end states, middle states to middle or to trailing or the end states, and trailing states only to the end state.

Observe that by assigning a state to each token or feature class, we encode transition probabilities more accurately than the usually adopted approach grouping all base tokens into one state. Thus, we are better able to exploit large sets of examples in reference tables. For example, grouping base tokens "st, hwy" into one *BaseToken* state also collapses all transitions from previous states (say, "49th, hamilton, SR169") to any of these individual tokens into one transition. It is possible that the states (e.g., "SR169") transitioning into the token "hwy" are very different from the states (e.g., "49th, hamilton") transitioning into the token "st." Grouping several base tokens into one BaseToken state loses the ability to distinguish among transitions. Therefore, we associate one base token per state. The cost of associating a token per state is that the number of transitions increases. However, we show in Section 6.5 that the resulting transition matrices are very sparse, and hence the comprehensive ARM models easily fit in main memory.

Emission and Transition Probabilities: To complete the instantiation of an attribute recognition model ARM_i on attribute A_i, we need to define the emission probabilities at each state and the transition probabilities between states. Since we in include in ARM_i a state s per element (base token or feature class) e in the categorized feature hierarchy, the emission probability distribution at s is: $P(x|e) = 1$ if $x = e$ and the position of x within ARM_i and that of e within the attribute value are identical, and 0 otherwise. In Section 4.4, we describe an algorithm for learning, during the preprocessing phase, the transition probability distribution from the reference table. In the next section, where we describe the relaxation of sequential specificity for robustness, we assume that these probabilities are known.

4.2 Sequential Specificity Relaxation

Consider the example path in Figure 3(a) consisting of a "short word," a "number ending in th or st," and a token in the set rd, wy, st, blvd, which accepts the string "nw 57th st" with a high probability. However, its erroneous versions "57th st," "nw57th st," "nw 57th," "nw 57th 57th st" have a very low acceptance probability due to the sequential specificity of the model in Figure 3(a): that a specific token has to follow another specific token. Therefore, erroneous transitions (from the state accepting "57th" to the end state, from the start state to the state accepting "57th") have a very low probability.

Our approach for trading sequential specificity for robustness is to "adjust" attribute recognition models trained on clean data by creating appropriate states and transitions in an ARM to deal with some of the commonly observed types of errors: *token insertions, token deletions, and missing values* [18]. Our adjustment operations exploit the uniform BMT topology and are illustrated in Figure 4. In order to deal with erroneous token insertions, we copy states along with transitions to and from these states in the *beginning and trailing* positions to the middle position (as shown in Figure 4(a)). We add low probability (according to the Good-Turing estimate [8]) transitions to the states copied from the beginning position from all states in the beginning position. Similar transitions to trailing states are added from the states copied from the trailing position. In order to deal with erroneous token deletions, we copy states and associated transitions from the *middle* position to the *beginning and trailing* positions (as shown in Figure 4(b)). In order to deal with missing attribute values, we create a transition (as shown in Figure 4(b)) directly from the

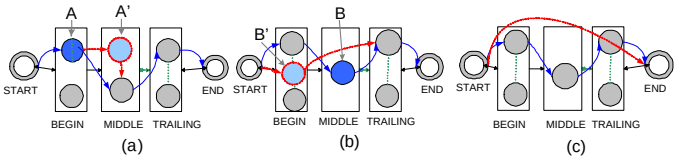

Figure 4: Illustration of robustness operations: Recover Insertions (a), Deletions (b), Missing Values (c)

start state to the *end* state. Observe that the implementation of these operations on a non-uniform topology would be very complex. Other common errors are addressed by other characteristics of ARMs. Our begin-middle-trailing topology is designed to be robust to token reorderings, especially, those in the middle position. *Spelling errors* are handled through token specificity relaxation as we describe next.

4.3 Token Specificity Relaxation

We now describe an approach for relaxing token specificity to account for unknowns or rare tokens. Our approach is a departure from the smoothing approach used by Borkar et al. [5]: during training, we *propagate* base token statistics to all matching feature classes. At runtime, if the observed token t is in the dictionary it is mapped directly to the appropriate state in the ARM; otherwise, the token is generalized to the minimal feature class that accepts t. In order to explicitly address *spelling errors*, we have experimented with matching unseen base tokens to states corresponding to known base tokens according to standard distance measures such as edit distance (e.g., "street" with "streat"). However, our experiments did not show any accuracy improvement over our original method.

4.4 ARM Training

We now describe the training procedure for computing transition probabilities between states in the BMT topology. The importance of each transition in recognizing whether or not a string exhibiting this transition belongs to the attribute depends on two factors: (i) *generative* factor: how often is the transition observed in the current attribute? (ii) *discriminative* factor: how unique is the transition to the current attribute?

Many traditional approaches for learning HMM transition probabilities rely only the *generative* factors. That is, the (generative) transition probability between two states s_1 and s_2 in ARM_i is the probability of observing token pairs t_1 and t_2 that can be emitted from s_1 and s_2. The generative approach results in high probability transitions between higher level feature classes (e.g., transitions between the $w+$ states that accept any token) even though such transitions may not discriminate an attribute from other attributes. For example, consider an erroneous bibliographic input string "editorial wilfred hodges of logic and computation" that has to be segmented into attributes [Title, Authors, Journal][5]. In our experiments, purely generative models segment this string as ["editorial", "wilfred hodges of logic", "and computation"] because the token "and" generalizes to a three-character string which is often observed as the beginning token for a journal name (e.g., "ACM TODS").

We ensure that the transition probabilities depend on both the generative and the discriminative factors. We now formalize this intuition to define the transition probabilities between pairs of states. In the following description, let s_1 and s_2 be two states in the attribute recognition model ARM_i. Let the *position* $pos_i(s)$ of a state s denote the position—beginning, middle, or trailing—of s in ARM_i. The *position* $pos(t, v)$ of a token t in an attribute value v is the position—beginning, middle, trailing—of t in the string v. Given two states s_1 and s_2, we say that the transition $t(s_1, s_2)$ from s_1 to

[5]The original record before corruption was ["Editorial", "Wilfred Hodges", "Journal of Logic and Computation"].

Procedure BuildARM (Table R, Column C, f)

1 First Pass: Scan R, build Dictionary(C)
1a Prune Dictionary(f)
2 Second Pass: Scan R, compute transition frequencies
3 Generalize: Propagate base transitions up the hierarchy
4 Compute transition probabilities
5 Apply Robustness transformations

Figure 5: ARM Training Procedure

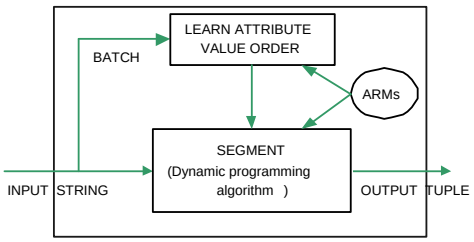

Figure 6: Segmentation Algorithm

s_2 is *valid* only if it is a non-backward transition. That is:

- if $pos_i(s_1) = beginning$ then $pos_i(s_2) \in \{middle, trailing, END\}$
- if $pos_i(s_2) = middle$ then $pos_i(s_2) \in \{middle, trailing, END\}$
- if $pos_i(s_1) = trailing$ then $pos_i(s_2) \in \{END\}$

Given an attribute value v from the attribute A_i and the states of ARM_i, we say that v *supports* a valid transition $t(s_1, s_2)$ if there exists a pair of consecutive tokens t_1 and t_2 in v such that $pos(t_1, v) = pos_i(s_1)$, $pos(t_2, v) = pos_i(s_2)$, and either t_1 and t_2 (i) are emitted with non-zero probability by s_1 and s_2, respectively or (ii) belong to the feature classes emitted by s_1 and s_2, respectively.

Positive Frequency: Given a reference table R, the *positive frequency* $f_i^+(t(s_1, s_2))$ of a transition $t(s_1, s_2)$ with respect to attribute A_i is the number of attribute values in the projection of R on A_i that support $t(s_1, s_2)$, and is 0 for all non-feasible transitions.

Overall Frequency: Given a reference table R, the *overall frequency* $f(t(s_1, s_2))$ of a transition $t(s_1, s_2)$ is the number of attribute values from any attribute that support the transition $t(s_1, s_2)$. That is, $f(t(s_1, s_2)) = \sum_i f_i^+(t(s_1, s_2))$.

Generative Transition Probability: Given a reference table R, the *generative transition probability* $GP(t(s_1, s_2)|A_i))$ of transition $t(s_1, s_2)$ with respect to an attribute A_i is the ratio $\frac{f_i^+(t(s_1,s_2))}{\sum_j f_i^+(t(s_1,s_j))}$.

Transition Probability: Given a reference table R, the *transition probability* $P(t(s_1, s_2)|A_i$ of a transition depends on its generative probability and its ability to distinguish attribute A_i. Assuming independence between the two aspects, we compute the transition probability as the product: A_i: $GP(t(s_1, s_2)|A_i) \cdot \frac{f_i^+(t(s_1,s_2))}{f(t(s_1,s_2))}$. Note that we do not re-normalize the transition probabilities. An EM-style Baum-Welch [26] algorithm or other discriminative training methods (e.g., [10]) can be used to further optimize the transition probabilities. In our implementation, we used the efficient two-pass procedure described next (Figure 5).

The pseudocode for the training procedure is shown in Figure 5. The overall procedure requires two passes: the first pass builds the dictionary and the second pass computes transition probabilities and applies robustness adjustments.

ARMs SUMMARY: The crucial aspects of ARMs are (i) the adoption of a fixed BMT topology that allows us to efficiently learn them from large reference tables and to gain robustness to input errors, (ii) the association of a state per base token to accurately encode transition probabilities and exploit large dictionaries, (iii) the relaxation of sequential specificity by adjusting an ARM learned from clean reference data, and (iv) the computation of transition probabilities to distinguish target attributes on which ARMs are built.

5. SEGMENTATION ALGORITHM

In this section, we describe an efficient algorithm for segmenting input strings. The segmentation problem has two components: first, determining the sequence in which attribute values are concatenated in an input string and second, determining the best segmentation of an input string into the corresponding ordered sequence of attribute values. Previous supervised approaches learned the attribute value order from the training data. For example, Borkar et al. [5] model the probabilistic attribute order using a hidden markov model. For instance, the author attribute immediately precedes the title attribute with probability 0.77 and the year attribute immediately precedes the booktitle attribute with probability 0.35. Once such a probabilistic order is known, a dynamic programming algorithm based on the Viterbi approximation can be employed to determine the best segmentation [25]. Therefore, in the rest of this section, we only discuss the solution for the first sub-problem of determining the attribute value order. For the second sub-problem, a dynamic programming algorithm can be employed. (However, our implementation uses an exhaustive search.) Figure 6 illustrates our approach: first learn the total order of attribute values over a batch of input strings, and then segment each individual string using this (fixed) attribute order.

5.1 Determining Attribute Value Order

We now describe an efficient algorithm for determining the attribute value order in input strings. Our algorithm is based upon the following observation. Attribute value orders of input strings usually remain same within *batches* of several input strings. For example, a data source for bibliographic strings may concatenate authors, title, conference name, year, pages in this order, preserving order across strings within the same page. Therefore, we need to recognize and recover this order only once for the *entire* batch of input strings. We validated this assumption on real data sources on the web for the Media, Address, and Citation domains: content creators do tend to preserve the order of attributes at least within the same page.

We now formalize the above intuition. We first estimate the probability of attribute A_i preceding (not necessarily immediately) attribute A_j, and the use these estimates to determine the most likely total ordering among all attributes. We now describe these two steps of the process.

Pairwise Precedence Estimates

Intuitively, the precedence estimate $prec(A_i, A_j)$ of an attribute A_i preceding attribute A_j is the fraction of input strings where the attribute value for A_i is before the attribute value for A_j. The precedence order among attributes for a single input string is determined as follows. For each attribute, we determine the token in the input string s at which it is most likely to start. For a pair of attributes A_i and A_j, if the token at which A_i is most likely to start precedes the token at which A_j is most likely to start, then we say that A_i precedes A_j with respect to the input string s. We break ties by picking one of the two attributes with equal probability. For example, consider an input string consisting of 8 tokens "walmart 20205 s. randall ave madison 53715 wi." We compute an 8-coordinate vector $[0.05, 0.01, 0.02, 0.1, 0.01, 0.8, 0.01, 0.07]$ for the city attribute. The first component 0.05 in the vector denotes the probability of the city attribute starting at the token "walmart." Because the 6^{th} coordinate is the maximum among all coordinates, the city attribute is

most likely to start at the token "madison." Suppose the vector for the street attribute is $[0.1, 0.7, 0.8, 0.7, 0.9, 0.5, 0.4, 0.1]$. The maximum for the city vector occurs at the 6^{th} coordinate and that for the street occurs at the 5^{th} coordinate. Therefore, street attribute value precedes the city attribute value for this input string. The fraction of input strings in the entire batch where attribute A_i precedes A_j is an estimate for A_i preceding A_j.

Formally, let s be a given input string within a batch S of strings. We tokenize s into a sequence $t_1, \ldots, t_m$ of tokens and associate with each attribute A_i ($1 \leq i \leq n$) a vector $v(s, A_i) = [v_{i1}, \ldots, v_{im}]$. The component v_{ij} is an estimate of the attribute value for A_i starting at the token t_j; v_{ij} is the maximum probability with which ARM_i accepts any prefix of $[t_{ij}, \ldots, t_{im}]$. Let $max(v(s, A_i))$ denote the coordinate corresponding to the maximum among values $v_{i1}, \ldots, v_{im}$. That is, $max(v(s, A_i)) = argmax_j\{v_{ij}\}$. The *precedence estimate* $prec(A_i, A_j)$ is:

$$prec(A_i, A_j) = \frac{|\{s \in S : max(v(s, A_i)) < max(v(s, A_j))\}|}{|S|}$$

At the end of this phase, we possess the pairwise precedence estimates between all pairs of attributes. Computationally, this procedure requires invoking the ARMs for determining acceptance probabilities of sub-sequences of tokens from each input string in a batch. If the average number of tokens in an input string is m, this computation involves $O(m^2)$ calls to ARMs. These acceptance probabilities can be cached and later used during the actual segmentation, thus avoiding repeated invocation of ARMs.

Determining Total Attribute Order

Using the directed attribute precedence probabilities estimated as described above, we can now estimate the best *total* order among attributes. The quality of an attribute order is the product of precedence probabilities of consecutive pairs of attributes in the given order. When the number of target attributes is small (say, less than 10), we can exhaustively search all permutations for the best total order. When the number of attributes is large, we can use more efficient heuristic search techniques (e.g., [20]).

6. EXPERIMENTAL EVALUATION

In this section, we evaluate CRAM using a variety of real datasets from real operational databases to show that CRAM is a robust, accurate, and efficient unsupervised domain-independent segmentation system. We first describe our experimental setup (Section 6.2). We present results on accuracy in Sections 6.3 and 6.4, and those on scalability in Section 6.5.

6.1 Experimental Setup

We now describe the datasets and the evaluation metrics we adopted.

Reference Relations: We consider reference relations from three different domains: addresses, media (music album records), and bibliography domains.

• **Addresses**: The address relation consists of $1,000,000$ clean and standardized individual and organization addresses from United States and Puerto Rico with the schema: [*Name, Number1, Number2, Address, City, State, Zip*].

• **Media**: The media reference relation consists of $280,000$ clean and standardized records describing music tracks with the schema: [*ArtistName, AlbumName, TrackName*].

• **Bibliography**: The bibliography relation consists of $100,000$ bibliography records from the DBLP repository and has the following schema: [*Title, Author, Journal, Volume, Month, Year*].

Error	*Description*
Spelling	A randomly chosen token is corrupted
Deletions	A randomly chosen token is deleted
Insertions	Insert a random token from dictionary
Reorders	Move randomly chosen token to new position
Missing	Replace randomly chosen attribute with null
All 5 Errors	Apply one or more of the above errors to a tuple
Natural	Naturally erroneous data (manually entered by users)

Table 1: Error Model: General attribute error types used for corrupting the test datasets.

Test Datasets: We evaluated CRAM over both naturally concatenated strings obtained from the web and from internal company sources. Further, in order to allow controlled experiments, we generated extensive test sets by concatenating "error-injected" real records into strings. All test datasets are disjoint from training datasets.

• *Naturally Concatenated Test Sets*: Company addresses obtained from the *RISE* repository of information extraction sources (publicly available); Individual addresses and Media filenames from internal company sources (not publicly available); 100 most cited papers from Citeseer (publicly available). We will refer to these collectively as **Natural** datasets.

• *Controlled Test Data Sets*: Both clean and erroneous strings are obtained by concatenating attribute values of records in a test relation held aside for validation. Therefore, the training and test sets are disjoint. To automate evaluation, we fix the order in which attribute values are concatenated to be a randomly chosen permutation of all attributes. Erroneous input strings are generated from test tuples by passing them through an *error injection phase* (as in [7]) before they go into the concatenation phase described above. The error injection phase is a controlled injection of a variety of errors, which are commonly observed in real world dirty data [18], into the test tuples. The attribute value(s) to introduce an error into is chosen randomly from among all attribute values. The types of errors and their effects on attribute values are listed in Table 1. While generating input strings with errors, we introduce at least one error into every tuple. For the mixed error model (**All 5 Errors** in Table 1), we assume that each error type occurs with equal probability.

Systems Compared: We compare our system CRAM with a state of the art supervised text segmentation system DATAMOLD [5], which was designed for automatic text segmentation and was shown to outperform other competing systems such as Rapier [6]. Thus, our comparison is definitive, since by outperforming DATAMOLD, CRAM is virtually guaranteed to outperform other systems as well.

6.2 Evaluation Metrics

As discussed earlier in Section 5, we separate the problem of determining attribute order from that of arriving at the best segmentation given the order. Reflecting this separation, we also split the evaluation of the attribute order determination from that of segmentation accuracy given the order.

Segmentation Accuracy: We measure *segmentation accuracy* as a fraction of attributes segmented correctly. From an input string s, let n' (out of a maximum of n attributes) be the number of attributes correctly identified by a segmentation algorithm Alg. We define the segmentation accuracy $acc(s, Alg)$ on the input string s as $\frac{n'}{n}$, or the fraction of target attributes correctly identified. For a set S of input strings, the overall segmentation accuracy of Alg is defined as the average fraction of correctly identified attributes over all strings in S, or, more formally as:

$$Accuracy(Alg) = \frac{\Sigma_{s \in S} acc(s, Alg)}{|S|} \quad (1)$$

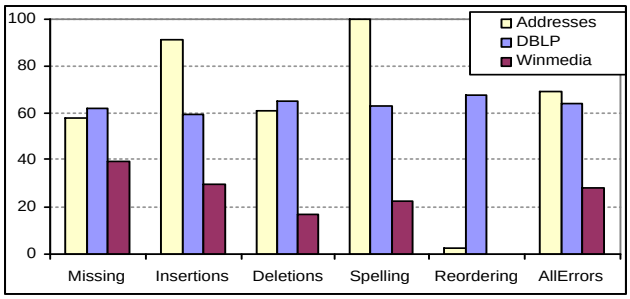

Figure 8: Relative accuracy gain of CRAM over Datamold.

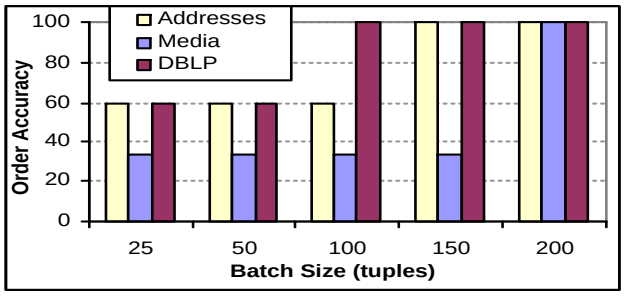

Figure 9: Accuracy of attribute order determination for the web-derived Addresses and Media, and the corrupted DBLP datasets.

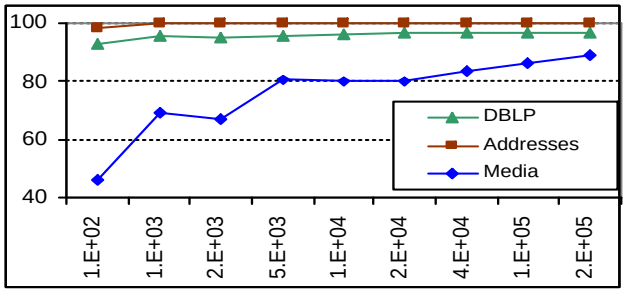

Figure 10: Accuracy of segmentation using CRAM for increasing reference table size (Addresses, DBLP, and Media datasets).

Attribute Order: We compute the accuracy of our order determination as the fraction of attributes in the input batch of strings which were assigned the correct absolute position. For example, if the correct order is *Number, Address, City* and our algorithm proposes *Number, City, Address*, the order accuracy is 0.33 or 33%.

6.3 Robustness of CRAM

We now evaluate the accuracy of CRAM over real datasets. We show that (i) the CRAM system substantially outperforms the state of the art supervised text segmentation system, (ii) the attribute order determination technique is accurate, and (iii) CRAM scales to large reference tables.

Segmentation Accuracy

We compare the segmentation accuracy of CRAM with that of Datamold over both erroneous and clean test datasets discussed in Section 6.1. For this experiment, we fix the order in which attribute values are concatenated to focus on the improvements due to using ARMs.[6] We report the segmentation accuracy in Figure 7 (a, b, and c). Observe that CRAM substantially improves accuracy, often by 8-15%, for all error types and over all datasets illustrating that ARMs are accurate in modelling attribute value domains. In order to put the accuracy improvements in a relative perspective, we report the same results in Figure 8 but in the form of the percentage relative error reduction over Datamold for the Addresses, DBLP, and Media datasets. For many of the noisy datasets, CRAM reduces segmentation errors by over 50%. These improvements directly result in the significant reduction of required manual intervention and increased accuracy while loading the data into relations.

Accuracy of Attribute Order Determination

We now evaluate our attribute order determination technique over batches of *natural* (i.e., user-entered or web-derived) strings. Figure 9 shows the high accuracy of our total order determination algorithm: in many cases we can determine order with 100% for a relatively small number (around 200) of tuples. This is a reasonable requirement, since many personal media libraries, and address and citation collections, and legacy data sources will have that many tuples.

Exploiting Large Reference Tables

We now study the impact of reference table sizes on segmentation accuracy. Figure 10 shows the results. Observe the increase in segmentation accuracy with the size of the reference table, especially for the Media dataset. In fact, note that accuracy of over 80% for the Media dataset is only achieved when the reference table size exceeds 40,000 tuples. Therefore, (i) exploiting rich dictionaries from large reference tables is important to achieve higher segmentation accuracy, and (ii) a segmentation system must *scale* to large reference table sizes. CRAM takes just a few minutes (< 5) to learn ARMs over a reference table of $200,000$ tuples. In contrast, supervised systems relying on cross-validation approaches would be much slower.

[6]Because of privacy restrictions, we were not able to ship "real" individual addresses data to be evaluated by DATAMOLD. All of the remaining results are performed over the *Natural* dataset, unless otherwise indicated. When this was not possible for privacy concerns, our *All 5 Errors* dataset provides a good approximation of the segmentation accuracy of the real data entered by real users.

6.4 Effectiveness of Robustness Techniques

In this section we evaluate the impact on segmentation accuracy of the BMT topology, the association of a state for each base token, and the robustness operations.

Fixed ARM topology and Robustness Operations

We now evaluate the effectiveness of our BMT topology—which alleviates the need for expensive cross-validation approaches for optimizing HMM topology—to show that it results in high segmentation accuracy, and is often better than topologies obtained by using expensive cross-validation approaches. We compare the accuracy of segmentation using two representative ARM topologies: (i) 1-Pos topology only models the sequential information between states by even collapsing all begin, middle, and trailing positions into one category and (ii) our BMT (the 3-Pos) topology. In fact, we found that increasing the number of positions in the fixed topology further does not increase accuracy. Due to space constraints, we omit the results. Figure 11 shows that the fixed BMT topology is usually more accurate than the 1-Pos topology that completely ignores positional specificity.

Figure 11 also reports results of sequential specificity relaxation operations over ARMs. These include the adjustment operations (e.g., recovering from token deletions) and the transition probability computation described in Section 4.4. As we can see, our robustness operations do provide a consistent improvement in accuracy over both erroneous and clean data on all datasets. The improvements on clean test datasets are the result of modified transition probability computation.

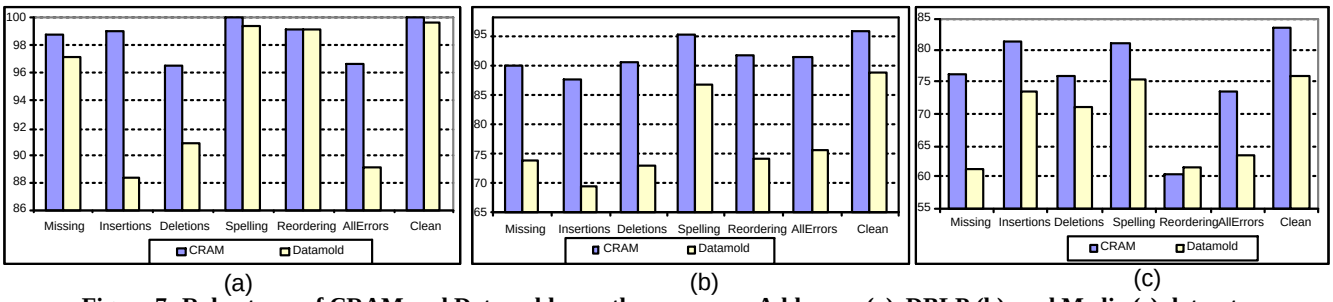

Figure 7: Robustness of CRAM and Datamold over the erroneous Addresses (a), DBLP (b), and Media (c) datasets.

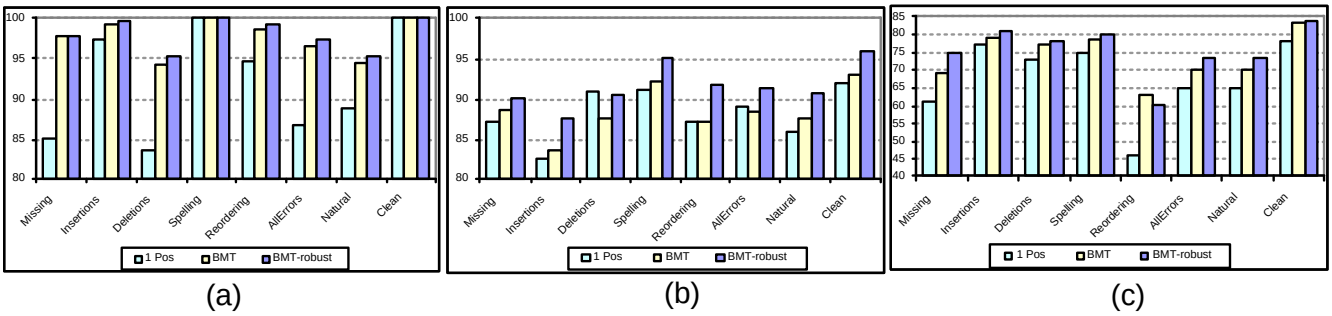

Figure 11: Accuracy of CRAM over Addresses (a), DBLP (b), and Media (c) datasets with different ARM topologies (1-Pos and BMT), and with robustness operations (BMT-robust).

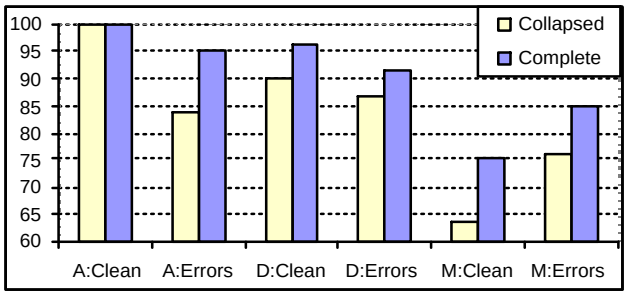

Figure 12: Complete vs. Collapsed token states over Addresses (A), DBLP (D), and Media (M) datasets.

Modelling Individual Base Tokens

As discussed in Section 4.1, we associate a state with each base token that is retained in the ARM. A common alternative is to collapse many base tokens together into one state of the HMM, which results in the loss of transitional information by collapsing transitions together. Figure 12 shows that collapsing base tokens together into one state (as is done in Datamold), results in substantially lower (sometimes by 10%) segmentation accuracy. The price of this accuracy gain is a larger model size. However, we show in Section 6.5 that CRAM achieves high accuracy even if we retain only important base tokens. In related experiments, which we omit due to space constraints, we studied the use of different hierarchies. However, they all resulted in similar segmentation accuracies.

Hypothetical Supervised Approach

We now demonstrate that making robustness to input errors a design criterion is essential. We consider a hypothetical scenario where we know the error model with the percentages of each error type expected in the test input. (Of course, this knowledge is not available in practice.) We use this knowledge to prepare a training dataset that reflects the variety of errors and percentages. As shown in Figure 13, Datamold:Hypothetical (corresponding to Datamold trained over this hypothetical dataset) has significantly higher segmentation accuracy than Datamold trained over clean data, but is still outperformed by CRAM trained on clean reference data. Thus, robustness to input errors is essential for deploying a segmentation system on real data.

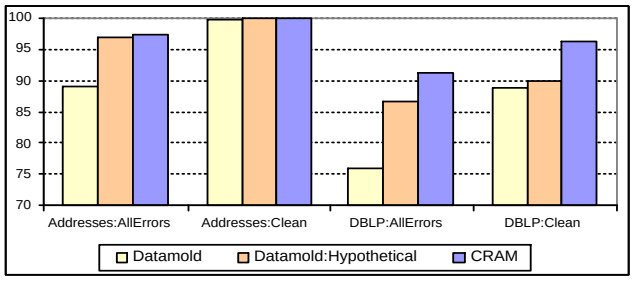

Figure 13: Segmentation accuracy of Datamold trained over clean and hypothetical erroneous training data.

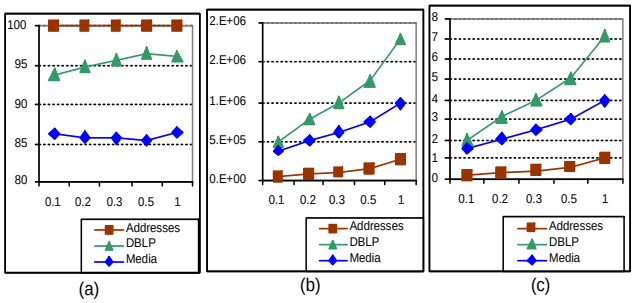

Figure 14: Varying fraction of tokens retained: Segmentation Accuracy (a), #Transitions (b), and Total Model Size (MB) (c).

6.5 Scalability

We now study the sizes of models with respect to reference table sizes and the impact of constraining model sizes on the segmentation accuracy. We validate our hypothesis that (i) the number of transitions grows much slower than a function that is quadratic in the number of states, and (ii) that we can achieve comparable accuracy by considering only a fraction of the most frequent base tokens.

Figure 14 shows results of varying the fraction f of base tokens retained while keeping the reference table size fixed. As shown in Figure 14(a), retaining only a fraction of the base tokens in ARMs gets us the same accuracy as that of retaining all base tokens. At $f = 1$, all tokens are retained and the model size (shown in Fig-

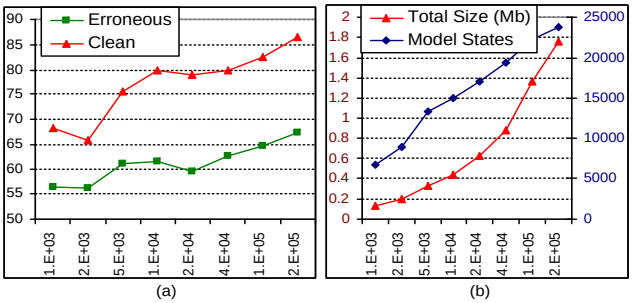

Figure 15: Varying reference table sizes: Segmentation Accuracy (a), Total model size (in MB) on the primary axis, and #Model states on the secondary axis (b).

ure 14(c)) is still less than a few MB for all datasets. And, the number of transitions (shown in Figure 14(b)) is around 10 times that of the number of states—usually $100,000$—in the model; hence, it is orders of magnitude less than $|S|^2$. Thus, we can significantly reduce memory requirements without compromising on segmentation accuracy.[7]

We now study the impact on accuracy of increasing reference table sizes while constraining the size of the CRAM model to be under 2MB (using the Media dataset). Figure 15(a) shows the improvement in accuracy as the reference table size increases from 1000 to $200,000$ tuples. Figure 15(b) shows the corresponding model size (which is constrained to below 2MB) and the number of states. Note that even under constrained model size, the resulting segmentation accuracy of 86% for $200,000$ tuples matches that of the larger unconstrained model trained over the same reference table. Thus, CRAM scales to large reference tables while maintaining a small memory footprint without compromising on segmentation accuracy.

In summary, we have shown CRAM to be robust, scalable, and domain independent. It is accurate on both clean and erroneous test data, and gracefully scales up to large reference table sizes.

7. CONCLUSIONS

In this paper, we exploit widely available reference tables to develop a domain-independent, robust, scalable, and efficient system for segmenting input strings into structured records. We exploit rich dictionaries and latent attribute value structure in reference tables to accurately and robustly model attribute value domains. To easily deploy CRAM over a variety of data sources, we address the problem of automatically determining the order in which attribute values are concatenated in input strings. Using real datasets from several domains, we established the overall accuracy and robustness of our system vis-a-vis state of the art supervised systems.

Acknowledgments: We thank Sunita Sarawagi for helping us compare our system with DATAMOLD. We also thank Theo Vassilakis for several thoughtful comments on the paper.

8. REFERENCES

[1] Microsoft SmartTagger.

[2] *Proceedings of the 7th Message Understanding Conference (MUC-7)*. Morgan Kaufman, 1998.

[3] B. Adelberg. NoDoSE–a tool for semi-automatically extracting structured and semistructured data from text documents. In *Proceedings of the ACM SIGMOD Conference*, 1998.

[4] J. Bilmes. What HMMs can do. Technical report, UWEETR-2002-0003, 2002.

[7] We did not have access to DATAMOLD code and so cannot report DATAMOLD's resulting model size or memory consumption.

[5] V. R. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *Proceedings of the ACM SIGMOD Conference*, 2001.

[6] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Sixteenth National Conference on Artificial Intelligence*, 1999.

[7] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the ACM SIGMOD Conference*, 2003.

[8] S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the annual meeting of ACL*, pages 310–318, 1996.

[9] W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: Combining semi-markov extraction processes and data integration method. In *Proceedings of the ACM SIGKDD Conference*, 2004.

[10] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the EMNLP Conference*, 2002.

[11] M. Collins and Y. Singer. Unsupervised models for named entity classification. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 1999.

[12] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large web sites. In *Proceedings of the VLDB Conference*, 2001.

[13] J. Droppo, L. Deng, and A. Acero. Evaluation of the splice algorithm on the aurora2 database. In *Proceedings of the Eurospeech Conference*, 2001.

[14] D. Embley, S. Jiang, and Y. Ng. Record-boundary discovery in web documents. In *Proceedings of the ACM SIGMOD Conference*, 1999.

[15] D. Freitag and A. McCallum. Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the AAAI/IAAI Conference*, pages 584–589, 2000.

[16] D. Freitag and A. McCallum. Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the AAAI/IAAI Conference*, pages 584–589, 2000.

[17] R. Grishman. Information extraction: Techniques and challenges. In *Information Extraction (International Summer School SCIE-97)*. Springer-Verlag, 1997.

[18] M. A. Hernandez and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[19] C. A. Knoblock, K. Lerman, S. Minton, and I. Muslea. Accurately and reliably extracting data from the web: A machine learning approach. *IEEE Data Engineering Bulletin*, 23(4):33–41, 2000.

[20] M. Lapata. Probabilistic text structuring: Experiments with sentence ordering. In *Proceedings of the annual meeting of ACL*, 2003.

[21] A. Martin and M. Przybocki. NIST 2003 language recognition evaluation. In *Proceedings of the Eurospeech Conference*, 2003.

[22] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the ICML Conference*, 2000.

[23] A. Mikheev, M. Moens, and C. Grover. Named entity recognition without gazetteers. In *Proceedings of EACL*, 1999.

[24] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In O. Etzioni, J. P. Müller, and J. M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 190–197, Seattle, WA, USA, 1999. ACM Press.

[25] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 1989.

[26] L. R. Rabiner and B. H. Juang. *Fundamentals of speech recognition*. Prentice Hall, 1993.

[27] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI 99 Workshop on Machine Learning for Information Extraction*, 1999.

[28] C. Sutton, K. Rohanimanesh, and A. McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of the ICML Conference*, 2004.