AutoDict: Automated Dictionary Discovery

Fei Chiang, Periklis Andritsos, Erkang Zhu, Renée J. Miller

Department of Computer Science, University of Toronto
Toronto, Canada
{fchiang, periklis, zhuerkan, miller}@cs.toronto.edu

***Abstract*— An attribute dictionary is a set of attributes together with a set of common values of each attribute. Such dictionaries are valuable in understanding unstructured or loosely structured textual descriptions of entity collections, such as product catalogs. Dictionaries provide the supervised data for learning product or entity descriptions. In this demonstration, we will present *AutoDict*, a system that analyzes input data records, and discovers high quality dictionaries using information theoretic techniques. To the best of our knowledge, AutoDict is the first end-to-end system for building attribute dictionaries. Our demonstration will showcase the different information analysis and extraction features within AutoDict, and highlight the process of generating high quality attribute dictionaries.**

I. Introduction

Web data is often in an unstructured text format or in a semi-structured record format. To support effective structured querying of this data, the information needs to be converted into a structured format. Consider the following example of TV descriptions. To understand, maintain, clean, or query this information, it is helpful to understand what parts of a product description represent properties such as manufacturer, model, size, refresh frequency, etc.

- t_1 : Samsung LN-52A650 52in 1080p LCD Widescreen 60Hz
- t_2 : Mitsubishi WD65835 65" 1920x1080 Projection LCD TV Widescreen
- t_3 : Sony XBR 46" 1080p LCD HDTV 120Hz

Manually identifying the inherent structure and appropriate attribute values in such records is a laborious task requiring highly specific domain knowledge. Users would need to identify the desired attributes, and have knowledge of the syntax and semantics of these attributes. Identifying the specifications of a bicycle (e.g., manufacturer, frame size, type) is different than the specifications of a digital camera (e.g., manufacturer, memory, zoom). Current techniques rely on identifying basic attributes, such as manufacturer, and having a domain expert manually identify the remaining descriptive attributes (e.g., memory size), and then creating taxonomies that categorize the entities and their properties. In this demonstration, we will present a tool that automates this process for a user, by identifying core attribute values in a record, and then automatically identifying values that belong to the same attribute. Unlike previous methods, we do not assume a priori knowledge of any record structure or a catalog. Instead, we make use of only a sample of marked values from the user.

Our demonstration will focus on product data. Many vendors such as amazon.com which sell a variety of products from different suppliers must be able to identify which parts of a product description are associated with which attribute. Given the heterogeneous data formats from different suppliers, that may contain missing fields, incorrect values, and inconsistent formats, the task of segmenting these records into attribute values is challenging. To extract values for attributes such as year, regular expressions can be used. However, for categorical attributes with no clear distinguishing formatting, such as TV manufacturer and model, a table of possible attribute values (i.e., a set of *dictionaries*) is most useful. Often users know which attributes they're interested in (for a given querying or cleaning task). So given a user provided set of attributes, we would like to segment records into values of those attributes. In AutoDict, we model attributes using discovered lists of attribute values.

Previous work on segmentation has attempted to leverage the inherent structure in the data records (given as strings) by expanding models such as Hidden Markov Models (HMMs) to include references to dictionaries [1], where the dictionaries are already given as input and not learned. Models such as Conditional Random Fields (CRFs) [2] and hierarchical HMMs [1] have also been used to segment records into entities, but the focus here is based on learning the models, assuming a partial dictionary is given. Work in query segmentation [3] and keyword tagging [4] has focused on using generative models to maximize the probability that a candidate segmentation is the correct one. Most of these models either assume a fixed attribute order (making them inflexible to handle string records where attribute values may appear in different orders or where a single value may be composed of non-sequential tokens in the string), or assume a dictionary is given to facilitate the tagging process [5]. A semi-automatic dictionary discovery tool is proposed by Godbole et al. [6] that assumes the attributes, and an initial set of values to start the search process, are given. Their discovery algorithm groups words into the same dictionary based on a frequency, tf-idf based similarity model. In our work, we discover frequently co-occurring sets of values, and present these to the user, ensuring that only relevant values are tagged. Furthermore, our discovery algorithm applies information theoretic techniques that capture both the frequency and information content between values.

Previous work has assumed that the attribute dictionaries are given, and has not addressed the question as to how these dictionaries are obtained. In some domains, standard tables such as cities, states, and postal codes are readily available. However, for domains such as televisions or children's

1084-4627/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDE.2012.126

clothing, although this information exists online, finding a single authoritative source listing all manufacturers, tv lines, or clothing lines, that is accurate and complete, is difficult. In addition, customizing this list according to the needs of the application and schema often requires highly specific domain knowledge.

In this demonstration, we present *AutoDict*, a tool that simplifies the segmentation and tagging tasks, and generates dictionaries from semi-structured data. AutoDict is able to achieve high quality dictionaries using only a small sample of input entries. We will show in this demonstration how a user can interactively create a dictionary according to their application requirements.

II. System Overview

Figure 1 illustrates the architecture of AutoDict consisting of modules for identifying and expanding segments, and tagging them to the appropriate attribute. A segment is an ordered set of values. The system takes a raw data file containing a list of records (where each string record contains a list of values describing a product), a set of user specified threshold parameters, and a schema containing k attributes as input, and produces k populated dictionaries according to these attributes. The first phase, *seed identification*, consists of identifying all the frequent sets of values occurring in the dataset which we call *seeds*. These seeds are ranked and refined according to the amount of information they capture, which is determined in the *information content quantification* module. At this point, we have a basic set of candidate segments. Since the quality of a dictionary is subjective according to the application needs, we allow the user to associate particular segments to specific dictionaries to produce more relevant dictionaries. This feedback will be used in the segment tagging module when deciding the segment to dictionary mapping. We expand upon this basic set of candidates in the *candidate generation* module by considering candidate values that may be composed of non-sequential values in the record. Finally, in the *segment tagging* component, for each record in the dataset, we tag each segment in a record with one of the k dictionary labels, thereby producing k populated dictionaries. We describe each of the four modules next.

A. Seed Identification

The first step in the dictionary discovery process is to identify the segments that are the core of a record. We call these initial segments *seeds*, since they form the basic set of segments which we will expand upon in the Candidate Generation phase. Due to their importance, we expect their frequency of occurrence in the data to be high. Given a raw dataset I, we first compute and store the frequency of all segments s_l of length l (i.e., containing l values), $l \in [1, n]$ for a given n. We add candidate segment s_l to an initial, single dictionary D if it satisfies our cost function (we do not assume a set of k attributes are known at this point). The cost function is based on finding the smallest and simplest model (i.e., dictionary) D to represent the data instance I. This idea can be quantified using the *Minimum Description Length (MDL) Principle* [7], [8]. MDL defines the description length DL for D as the length of the model $L(D)$, plus the length to encode the data values in I given the model $L(I|D)$. Our objective is to minimize DL. We seek to add segments to D that occur frequently in I. The idea is that if the frequency of s_l is sufficiently high, then adding s_l to D will increase $L(D)$ and decrease $L(I|D)$, such that the DL is further reduced. By doing this, we find the frequent segments in I that are now modeled in the dictionary D. After evaluating all s_l, the resulting D contains a basic candidate set of segments. We refine these seeds by considering their information content.

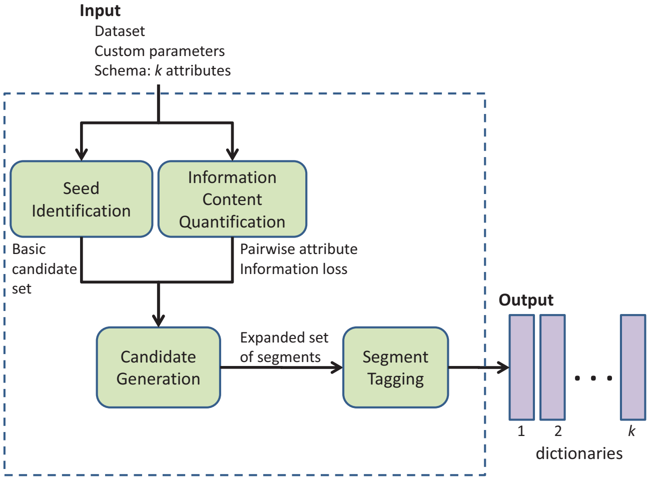


Fig. 1. Dictionary discovery framework

B. Information Content Quantification

As mentioned earlier, one of the key restrictions of current generative models is the inability to handle attribute values that consist of values in a different order across records, or values that do not appear consecutively. Our *Seed Identification* algorithm shares this limitation. For example, it is not able to identify the bi-gram ”LCD Widescreen” in t_2 because these keywords do not occur next to each other. To overcome this issue and quantify the information content of naturally co-occurring values, we use an information theoretic approach. In prior work on schema discovery, the *Agglomerative Information Bottleneck, (AIBM)* clustering algorithm has been used to quantify redundancy in large data sets [9]. The outcome of AIBM is a hierarchy of clusters of co-occurring values. To create attribute dictionaries, we cluster values and consider their distribution according to how they appear in the data set. We compute the information content between values and records, captured by the information-theoretic measure *mutual information*.

Since we would like to identify the strongest seeds based on the natural co-occurrence of values, we employ the *Information Loss* function as originally introduced in the *Information Bottleneck Method* [10]. Intuitively, information loss measures the change in information content when replacing a set of values represented by u with a set of values represented by v. We would like to find sets of values with minimal information loss.

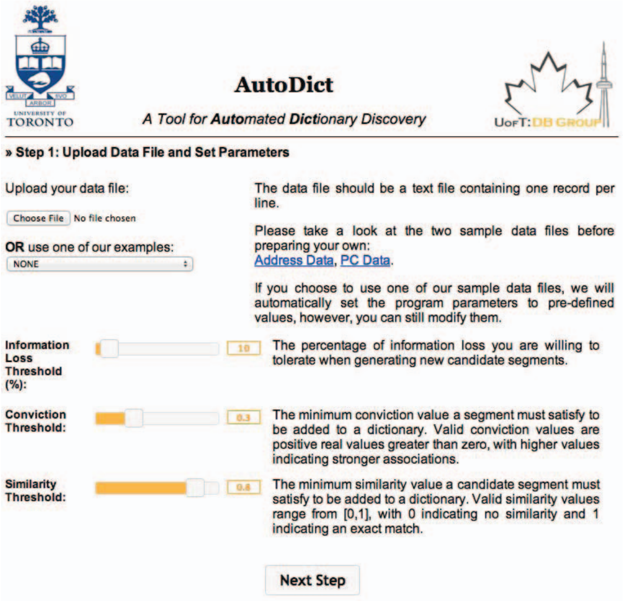
Our AIBM clustering computes the information loss for every pair of values that appear in the dataset. At each step, AIBM merges the values that incur the smallest loss of information. These are values that appear either exclusively (information loss equals zero), or almost exclusively in the data. This procedure assumes a bag-of-word semantics, and helps to identify values that are out of order in a record. At the end of the routine, we have quantified the information content in the dataset in two ways: (1) we know the order in which values merge with other values, and (2) we know the information loss incurred at each step. This information will be used to identify extensions of candidate segments in the candidate generation module.

C. Candidate Generation

Using the pairwise information loss values, we expand each candidate in the basic candidate set. Each candidate is a segment s_l also known as a *seed*. A seed serves as a core component of the record due to its high frequency of occurrence. We hope to extend the seeds with descriptive values while incurring minimal information loss, since our goal is to find strongly correlated values describing an attribute. We do this by considering candidate values $v_1 \ldots v_i$ within a window w of the seed, where $i \in [1, w]$. We expand s_l by adding i values to s_l. These i values occur on either side of s_l, possibly excluding values $v_j, j \in [1, w]$, to account for values that may not appear sequentially. In order for an expanded segment $s_l^{'}$ to be added to the initial dictionary D, two conditions must be satisfied: (1) the amount of information loss between s_l and $(v_1 \ldots v_i)$ (the descriptive values) must be less than the loss percentage we are willing to tolerate; and (2) the conviction value[1] between s_l and $(v_1 \ldots v_i)$ must be greater than the conviction threshold set by the user, since we only want to expand the segment with values that have strong association with s_l. Finally, the resulting set of $s_l^{'}$ segments are added to D, which will serve as the starting point for the segment tagging to generate the final set of dictionaries.

D. Segment Tagging

To allow a user to customize the dictionaries to their application requirements, we present a sample set of $s_l^{'}$ segments to the user, and ask the user to mark each segment with one of the k attribute labels. This allows our tagging algorithm to produce more relevant dictionaries based on the user's starting preferences. Segments marked with label $u \in [1, k]$, will compose the initial entries of dictionary d_u. A dictionary d_u contains values describing a specific attribute, and we define a property called *validity* that each d_u must satisfy with respect to all tuples in I. Specifically, the validity property states that a tuple can only contain at most one value from a dictionary d_u. For example, a tuple describing digital cameras cannot contain both `Nikon` and `Canon` as manufacturers. We will reference the validity property as we populate the dictionaries.



AutoDict

A Tool for Automated Dictionary Discovery

» Step 1: Upload Data File and Set Parameters

Upload your data file:

Choose File No file chosen

OR use one of our examples:

NONE

The data file should be a text file containing one record per line.

Please take a look at the two sample data files before preparing your own:
Address Data, PC Data.

If you choose to use one of our sample data files, we will automatically set the program parameters to pre-defined values, however, you can still modify them.

Information Loss Threshold (%): 10

The percentage of information loss you are willing to tolerate when generating new candidate segments.

Conviction Threshold: 0.3

The minimum conviction value a segment must satisfy to be added to a dictionary. Valid conviction values are positive real values greater than zero, with higher values indicating stronger associations.

Similarity Threshold: 0.8

The minimum similarity value a candidate segment must satisfy to be added to a dictionary. Valid similarity values range from [0,1], with 0 indicating no similarity and 1 indicating an exact match.

Next Step

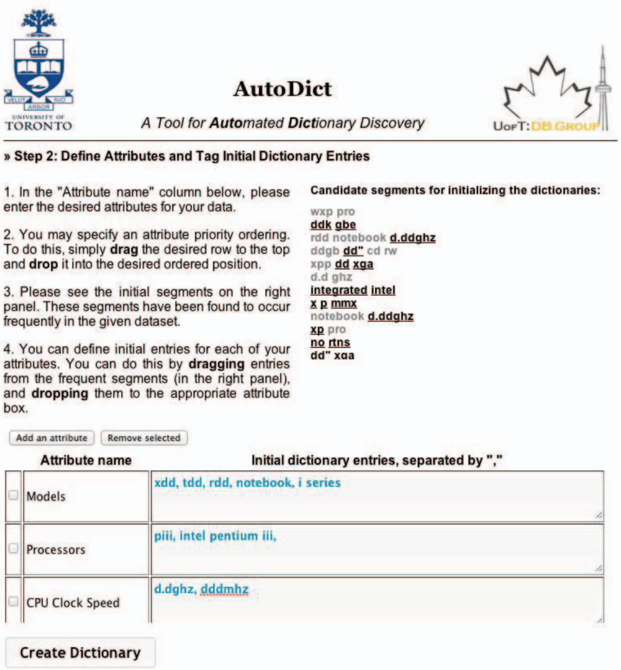
Fig. 2. Define the input parameters

We apply the MDL Principle to find a model M (a set of d_u's) that can represent the data instance I as succinctly as possible. We initialize M to the initial entries in $d_u, \forall u$. The initial $L(M)$ consists of counting the segments in d_u for all u, these values are considered *tagged* in I. From this, we compute $L(I|M)$ by counting those values not in M, these values are considered *untagged*. The instance I consists of tuples that may contain both tagged, and untagged values. For each tuple, we consider the possible assignments to tag the untagged values, and compute the associated DL cost. We select the assignment with lowest cost. If there is a tie, we compare the similarity between the candidate values and the dictionary values of the assigned attribute, and we select the attribute assignment with the highest similarity. If a candidate assignment leads to a violation of the validity property, that is, a tuple contains two values from the same d_u, then that candidate is disregarded. We process the tuples in increasing order of the number of untagged values. After evaluating all the tuples, the set of k dictionaries is returned to the user for review.

III. Demonstration

AutoDict has been implemented as a working prototype using Python and Perl, and offers an interactive web interface implemented using the jQuery UI Javascript library running on an Apache server. In the demonstration, we will highlight features of the AutoDict tool using real datasets, and guide a user as to how they can discover relevant dictionaries from their data. In particular, a user can upload their own data or use one of the provided sample datasets. We allow users to control the thresholds of the qualitative measures that our discovery algorithm uses, as shown in Figure 2. For example, if the given

[1] Conviction is an association measure based on co-occurrence frequency used to quantify attribute value relationships [11].



AutoDict

*A Tool for **Auto**mated **Dict**ionary Discovery*

» **Step 2: Define Attributes and Tag Initial Dictionary Entries**

1. In the "Attribute name" column below, please enter the desired attributes for your data.

2. You may specify an attribute priority ordering. To do this, simply **drag** the desired row to the top and **drop** it into the desired ordered position.

3. Please see the initial segments on the right panel. These segments have been found to occur frequently in the given dataset.

4. You can define initial entries for each of your attributes. You can do this by **dragging** entries from the frequent segments (in the right panel), and **dropping** them to the appropriate attribute box.

Candidate segments for initializing the dictionaries:

wxp pro
ddk gbe
rdd notebook d.ddghz
ddgb dd" cd rw
xpp dd xga
d.d ghz
integrated intel
x p mmx
notebook d.ddghz
xp pro
no rtns
dd" xqa

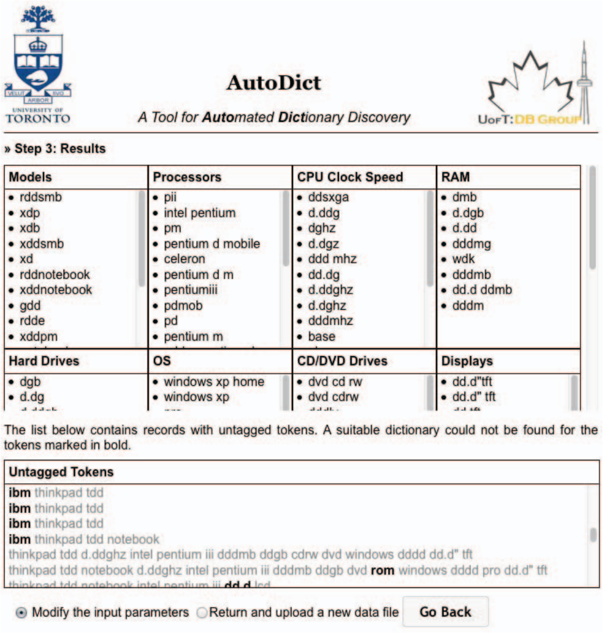
	Attribute name	Initial dictionary entries, separated by ","
□	Models	xdd, tdd, rdd, notebook, i series
□	Processors	piii, intel pentium iii,
□	CPU Clock Speed	d.dghz, dddmhz

Fig. 3. Define the attributes

dataset contains important syntactic formatting, *i.e.*, a user would like the data values to be matched based on distinctive data patterns, then a high similarity threshold should be used. Alternatively, if a user would like to obtain seed segments containing correlated values, then a higher conviction value should be specified. These parameter values can be adjusted using the sliding bars as shown in Figure 2.

In the next step, shown in Figure 3, users define their schema by specifying a list of attributes, where a dictionary will be created and populated for each attribute. Users can specify a priority ordering over the attributes by dragging and dropping the attribute to the desired ordered position. Attributes at the top have highest priority, indicating that the discovery algorithm will try to maximize the quality of these attribute dictionaries over attributes lower in the ordered list. In our example shown in Figure 3, we use a real laptops dataset focusing on `IBM Thinkpads` and define eight attributes, with the `Models` attribute having the highest priority. If users would like to customize the dictionaries towards their application requirements, they can define initial entries for each dictionary. We present a list of high frequency segments in the data (top right panel of Figure 3), and ask the user to assign these segments to a dictionary by dragging and dropping the segment into the appropriate attribute box.

After defining the schema and initial entries, Figure 4 shows the resulting set of populated dictionaries. There may be cases where not all the values in a record have been matched to a dictionary. We display these values (shown in bold) in their respective record, in an *Untagged Tokens* list, shown at the bottom of Figure 4. In our example, the value `ibm` was not matched to a dictionary since there was no suitable `Manufacturer` attribute. Users may download the dictionaries as a text file, or modify the parameters to generate a new set of dictionaries according to their application requirements.



AutoDict

*A Tool for **Auto**mated **Dict**ionary Discovery*

» **Step 3: Results**

Models	Processors	CPU Clock Speed	RAM
• rddsmb • xdp • xdb • xddsmb • xd • rddnotebook • xddnotebook • gdd • rdde • xddpm	• pii • intel pentium • pm • pentium d mobile • celeron • pentium d m • pentiumiii • pdmob • pd • pentium m	• ddsxga • d.ddg • dghz • d.dgz • ddd mhz • dd.dg • d.ddghz • d.dghz • dddmhz • base	• dmb • d.dgb • d.dd • dddmg • wdk • dddmb • dd.d ddmb • dddm
Hard Drives	**OS**	**CD/DVD Drives**	**Displays**
• dgb • d.dg	• windows xp home • windows xp	• dvd cd rw • dvd cdrw	• dd.d"tft • dd.d" tft

The list below contains records with untagged tokens. A suitable dictionary could not be found for the tokens marked in bold.

Untagged Tokens

ibm thinkpad tdd
ibm thinkpad tdd
ibm thinkpad tdd
ibm thinkpad tdd notebook
thinkpad tdd d.ddghz intel pentium iii dddmb ddgb cdrw dvd windows dddd dd.d" tft
thinkpad tdd notebook d.ddghz intel pentium iii dddmb ddgb dvd **rom** windows dddd pro dd.d" tft
thinkpad tdd notebook intel pentium iii **dd.d** lcd

● Modify the input parameters ○ Return and upload a new data file

Fig. 4. The final populated dictionaries

IV. Conclusions

In this demonstration, we present AutoDict, a novel dictionary discovery tool that incorporates a set of measures including information content, similarity, and conviction, to produce relevant and accurate dictionaries. We show the usefulness of our approach by demonstrating our tool using real data in the products domain.

References

[1] V. Borkar, K. Deshmukh, and S. Sarawagi, "Automatic segmentation of text into structured records," *SIGMOD Rec.*, vol. 30, no. 2, pp. 175–186, 2001.

[2] S. Sarawagi and W. W. Cohen, "Semi-markov conditional random fields for information extraction," in *NIPS*, 2004, pp. 1185–1192.

[3] B. Tan and F. Peng, "Unsupervised query segmentation using generative language models and wikipedia," in *WWW*, 2008, pp. 347–356.

[4] N. Sarkas, S. Paparizos, and P. Tsaparas, "Structured annotations of web queries," in *SIGMOD Conference*, 2010, pp. 771–782.

[5] E. Cortez, A. S. da Silva, M. A. Gonçalves, and E. S. de Moura, "Ondux: on-demand unsupervised learning for information extraction," in *SIGMOD '10*, 2010, pp. 807–818.

[6] S. Godbole, I. Bhattacharya, A. Gupta, and A. Verma, "Building reusable dictionary repositories for real-world text mining," in *CIKM*, 2010, pp. 1189–1198.

[7] J. Rissanen, "Modeling shortest data description," in *Automatica*, 1978.

[8] T. Cover and J. Thomas, "Elements of information theory."

[9] P. Andritsos, R. J. Miller, and P. Tsaparas, "Information-theoretic tools for mining database structure from large data sets," in *SIGMOD*, 2004, pp. 731–742.

[10] N. Slonim and N. Tishby, "Agglomerative information bottleneck," in *NIPS*, 1999, pp. 617–623.

[11] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," in *SIGMOD'97*, pp. 255–264.