Iterative Algorithm for Inferring Entity Types from Enumerative Descriptions

Qian Chen
Graduate School of Information, Production and Systems
Waseda University
Kitakyushu 808-0135, JAPAN
chenqian@asagi.waseda.jp

Mizuho Iwaihara
Graduate School of Information, Production and Systems
Waseda University
Kitakyushu 808-0135, JAPAN
iwaihara@waseda.jp

ABSTRACT
Entity type matching has many real world applications, especially in entity clustering, de-duplication and efficient query processing. Current methods to extract entities from text usually disregard regularities in the order of entities appearing in the text. In this paper, we focus on enumerative descriptions which enlist entity names in a certain hidden order, often occurring in web documents as listings and tables. We propose an algorithm to discover entity types from enumerative descriptions, where a type hierarchy is known but enumerating orders are hidden and heterogeneous, and partial entity-type mappings are given as seed instances. Our algorithm is iterative: We extract skeletons from syntactic patterns, then train a hidden Markov model to find an optimum enumerating order from seed instances and skeletons, to find a best-fit entity-type assignment.

Categories and Subject Descriptors
H.8 [**Database Management**]: Data mining

General Terms
Algorithms, Performance, Experimentation.

Keywords
RDF graph; Information Extraction; Hidden Markov Model

1. INTRODUCTION
We define *enumerative descriptions* as structured descriptions which contain listings of entities, where the order of listings follows a certain unknown order in the class hierarchy, characterized by a regular expression. There are many forms of web resources that contain such enumerative descriptions, such as product descriptions in e-commerce, Infoboxes of Wikipedia, tables coded by Media Wiki, and listings in web pages which start with entity names. Infoboxes, tables and listings contain parent types in areas like section titles and adjoining table cells. Recognizing entities from web resources and assigning their correct types are crucial for various applications, such as search result categorization and disambiguation. As an example of enumerative descriptions, real product descriptions in e-commerce sites are shown in Table 1. In the first line, Lumsing Battery is compatible with a list of smart devices of different classes. In the product title of ID 1 in Table 1, we notice that the substring starting from "Apple:iPad Mini, iPad 4, 3, 2, Android …" is listing entities, such as "Apple", "iPad mini", "iPad 3" in a certain order. Using external knowledge such that "Apple" is an entity of type "Company" and "iPad 3" is a list of entities "Brand" and "Model", we can surmise the types of the rest of entities. We call such texts enlisting entities, where these entities belong to a certain type hierarchy, as enumerative descriptions. If a list of enumerated entities are a mix of known and unknown entities, the type of the latter can be inferred from that of the former, by capturing the enumerating orders. As the example of Table 1 indicates, we can find ordering patterns from syntactic features such as punctuations and delimiters. We call a regular expression on types and delimiters a *description pattern*. In Table 1, we notice that enumerative descriptions are not unique, but a mixture of heterogeneous description patterns. However, we can employ an assumption that these enumerative descriptions arise from a single entity hierarchy, and thus we can deduce discovered facts across heterogeneous patterns. In summary, our problem on type inference over enumerative descriptions is unique and new in the sense that (1) finding both parent and child entities simultaneously from enumerative descriptions that obey certain hidden entity enumeration orders, and (2) finding an entity-type mapping from a mixture of heterogeneous patterns. Our application scenario is to find typing information from partially typed, structured information from web resources, such as e-commerce sites and Wikipedia.

RDF (Resource Description Framework) is a general standard for describing structured descriptions on Web resources and capturing their relationships. In RDF, entities are represented as sets of <subject, predicate, object> or <subject, property, object> triples. We assume that a type hierarchy represented as an RDF schema, is given for type inference. Figure 2 shows the RDF schema describing our running example.

Our algorithm can be implemented as automatically expanding RDF graphs by augmenting them with discovered entities and their types. Firstly, since not all type names are captured in the initial given graph, so the method should capture existing types and identify new types (classes). Secondly, since not all type names are listed in the product descriptions, the syntactic structure of the description patterns should be captured as skeletons. We employ a conventional assumption that an enumeration of types is according to a regular order, such as a navigation path in a RDF schema graph. But we need to reconstruct the most plausible navigation path from the enumerative description. At last, we need to improve the accuracy and efficiency of inferring such types on entities.

WWW'14 Companion, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04
http://dx.doi.org/10.1145/2567948.2579706

Table 1 Product descriptions

ID	Website	Title
1	Amazon	Lumsing 11000mAh 5 x USB External Battery Pack Charger Power Bank For Apple:iPad Mini, iPad 4 3 2, Android Tablets:Samsung Galaxy Tab 2, Note 10.1; Google Nexus 7,10; Acer B1; iPhone 5 4S 4 3GS, iPod; Android Smartphone: Samsung Galaxy S4, S3, S2, Ace,Note 2; HTC One, One X, Desire X; LG Optimus 4X HD, I7, I9;Nokia Lumia 920, Google Nexus 4, Blackberry Z10, Sony Xperia Z; MP3, MP4, GPS, Camera, Game Player
2	Amazon	Anker® Astro3E 10000mAh High Capacity Power Bank Pack Portable External Battery Charger for iPhone 5, 4S, 4, iPad 4, 3, 2, Mini, iPods; Samsung Galaxy S4, S3, S2, Note 2; HTC One, EVO, Thunderbolt, Incredible, Droid DNA; Motorola ATRIX, Droid; Google Nexus 4, Nexus 7, Nexus 10; LG Optimus; PS Vita, GoPro
3	Amazon	GTMax Black Touch Screen Styli Stylus For Samsung, Blackberry, HTC, LG, Pantech, Huawei, iPhone, Motorola, Nokia, Smartphone, iPad, Asus, Acer, Toshiba, Archos Tablet
4	Amazon	DandyCase White/Grey Waterproof Case for Apple iPhone 4, 4S, iPod Touch 3, 4, iPhone 3G, 3GS, & Other Smartphones

Exacting hierarchical and diverse entities along with implied relationships between them with high accuracy is the main goal of this work. With regards to this, we present an iterative pattern-based algorithm called HoverTyp (Horizontal-Vertical Type Extractor). The central paradigm used by HoverTyp is an iterative framework of starting with an initial labeled RDF graph and given RDF schema which is used to construct an initial Hidden Markov Model. They are in turn used to classify more instances from the dataset and then update the initial Hidden Markov Model until convergence. At every stage, captured entity types and candidate HMM are scored, reflecting their degree of likelihoods.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 describes our data representation framework. Section 4 presents each of the proposed techniques in detail. Experimental evaluations are presented in Sections 5. Section 6 concludes the paper with future work.

2. RELATED WORKS

RDF is a W3C standard to represent metadata, which has several equivalent formats, for example, triple store, column store, property tables, or graphs. In existing graph database researches, Zhao and Han [10] designed a neighborhood signature to directly store labels within *k* hops from each vertex. In contrast, our work involves uncertain candidate labels. Several previous works in the semantic web area considered uncertainty in the RDF data. Furthermore, during data integration [2], the integrated RDF data from different sources may often contain conflicting or duplicate information. Therefore, a new probabilistic RDF data graph model has been proposed. Huang and Liu [3] modeled uncertain RDF data by a probabilistic database. However, this model assumes that RDF triples have independent existence probabilities to appear in reality.

Lian and Chen [4] propose a different format of probabilistic RDF graphs. The node labels are not deterministic (multiple labels with probabilities). Probabilistic RDF graphs are related to our research, because RDF graphs extracted from product descriptions are not deterministic, in the sense that there can be different possible labels to tokens. However, our problem involves a different model of probabilistic RDF graphs which have uncertain vertex labels (but certain edge labels). We need to assign possible labels to tokens and calculate likelihoods by simultaneously aligning with given type hierarchies.

TYPifier [6] is a method proposed to infer the type semantics of structured data and build type hierarchy trees. They use complex pseudo-schema features to indicate missing type information and learn type hierarchies by proposed systems. RDF schema features are also used in our approach with different intention. Moreover, type hierarchies in our RDF graph are more complex and detailed.

Ontologies have played a central role in the development of semantic web. RDF Sentence Graph[9] has been used to automatically summarize ontologies which are widely used in understanding unstructured documents. An ontology typically includes concepts and hierarchical relationships. One of the approaches to learning an ontology from unstructured text is using lexico-syntactic patterns. LASER[5] is an iterative process starting with ISA and HASA seed patterns to effectively discover new patterns. However, this method only concentrates on the hierarchical relationships and ignores horizontal compatible relationships. On the other hand, seed patterns in our method are not pre-defined, but partially given in the initial instance graph. Our algorithm tries to infer types to unknown entities based on discovered patterns in enumerative descriptions.

In most of real-world extraction applications, a pattern-based algorithm is used in an iterative process: Starting with a relatively small set of seed tuples, these extractors iteratively learn patterns that can be instantiated to identify new tuples. I4E[7] is proposed as a graph-based framework that integrates tuples, patterns, and various trace information at each iteration. I4E assigns a confidence score to each pattern based on individual tuples that generate the pattern as well as the collective set of tuples produced by the pattern. In our method we also introduce confidence score on every iteration to update the patterns and prune candidate type labels. However, our problem involves a different model of unknown relationships, and enumerative descriptions contain a list of entities with heterogeneous entity types.

3. ASSUMPTION AND ARCHITECTURE

In this section we show assumptions used in our algorithms and outlines the architecture of our system.

3.1 Assumptions

We adopt the following assumptions in HoverTyp：

- An RDF schema for enumerative descriptions is given.
- Entities belonging to top-level types in the RDF schema, such as companies, operating systems, categories, are given in a table.
- Relationships connect two classes, but no schema for relationships is given. That is, any two classes can have a relationship.

- A portion of the RDF instance graph is given as seed instances. We try to grow this initial graph.

3.2 System Framework

Figure 1 shows the system framework of HoverTyp, where the circle indicates the main components. HoverTyp takes preprocessed descriptions with each word tokenized as input. In addition, it receives a given RDF schema and initial RDF instance graph. The system output is an RDF instance graph augmented with discovered entities and their types. The main part has two stages:

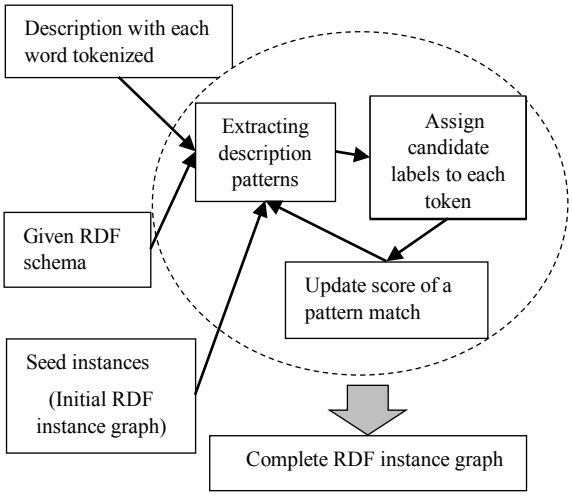

Figure 1 System framework of HoverTyp

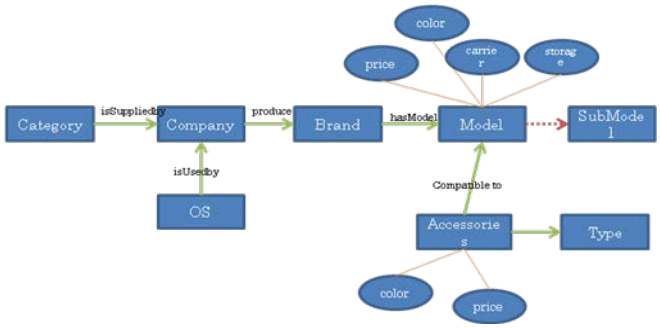

Figure 2 RDF schema of Smartdevices

Extracting description patterns: Skeletons are extracted by analyzing hierarchical relationships indicated by delimiters in the descriptions. Here, skeletons do not carry any type information. Then description patterns are generated by instantiating skeletons with types and delimiters, using the seed instances.

Assigning candidate labels: We apply the current set of description patterns to each description in the dataset and assign candidate labels on each token in the description. A Hidden Markov Model (HMM) is constructed from the seed instances, and type assignments are generated by dynamic programming on the HMM. The HMM is updated as we process succeeding descriptions.

4. APPROACH AND ALGORITHM

4.1 Preprocessing on tokens

- Certain values may have ambiguous meanings. Multiple candidate labels need to be assigned. Eg. "White" can be a color of phones or cases.
- There exist tokens not related to the RDF graph (eg. stop words, adjectives). We need to preprocess descriptions with each word tokenized and decide whether a token is an instance value.
- Tokens like "External Battery Pack" are not used in building the instance graph. But these tokens are characterizing the product itself.

4.2 Skeletons

We introduce *skeletons* that represent syntactical structures of enumerative descriptions of entities. Skeletons simply capture hierarchal structures of enumerative descriptions, containing no type information. In the next step, we consider mapping between skeletons and the RDF graph. The result is description patterns that can explain one enumerating pattern of types. Remember that our target descriptions consist of heterogeneous hidden patterns.

As an example of skeletons, let us consider the following description on an external battery pack with compatible information on smart devices:

PowerGen PGMPP12000 12000mAh External Battery Pack High Capacity Power Bank Charger Triple USB 3Amps output for Apple iPhone 5 4s 4 3Gs 3G, iPod Touch, iPad 1 2 3 4, The New iPad 3/ HTC sensation, XE, XL, One X S V, Thunderbolt, Inspire 4G , EVO 3D, EVO 4G, Desire S Z HD / Samsung Galaxy S3 S2 S 2 II ACE Mini, S Advance, Galaxy Nexus, Nexus 7, Tab / Motorola Atrix 2, Droid 3 X X2 Razr Maxx, Bionic, Triumph

In this description, the delimiters are " " (space), "," (comma) and "/" (slash). We observe that delimiters are hierarchically organized in the ascending order of 1: white space, 2: comma, and 3: slush. From the delimiters, we can construct a tree structure to extract a regular expression.

- We first replace each token with symbol "s". We record the original position of each "s" and the symbol s is not assigned any type. We should note that the proceeding part "… output for" is not considered. The beginning position of an enumerative description can be detected by occurrences of known types in the seed instances, such as companies and brands. The first part "s s s s s s s, s s, s s s s s, s s s s /" corresponds to "*Apple iPhone 5 4s 4 3Gs 3G, iPod Touch, iPad 1 2 3 4, The New iPad 3/*".
- We group the lowest level tokens which is separated by spaces and assign placeholder "X". The first part "X, X, X, X / X, X, X," corresponds to "*Apple iPhone…XE, XL,*". Here, the first "X" corresponds to the first seven tokens "s s s s s s s".
- Likewise, we group the sequences of X's, delimited by commas, into a placeholder "Y":Y / Y
- Finally, we group the sequences of Y's, delimited by slashes, into a placeholder "Z".

We obtain the skeleton Z—Y—X—s. The skeleton is not yet mapped to any types.

4.3 Description Patterns

We introduce description patterns to describe possible enumerating orders of types.

Description pattern: A description pattern is a regular expression that accepts possible enumerating orders of types, where the orders between types in a hierarchical relationship in the given RDF schema must be uniform.

Given a seed instance in which each token is marked with correct types in the RDF graph, we fold repetitions and subexpression to obtain a description pattern.

-The correct types for the tokens are given as assignment of a type to each symbol 's'.

Skeleton: s s s s s s s, s s, s s s s s, s s s s / s s, s, s, s s s s,

Pattern:c b m m m m m,b m,b m m m m m,# # b m/c b,m,m,b m m m,

c: Company b: Brand m: Model, #: non-type tokens ("The New")

- Fold repetitions: If a pattern contains a list of repeating identical types, we fold these repetitions.

Pattern: c b m^+*, b m, b* m^+*, b m/c b, m, m, b* m^+

(Note: '+' is one or more repetition. '*' is zero or more.)

- Further fold common subexpressions

Pattern: $[c[bm^*|m^+]^+]^+$

(Note: '|' is alternation.)

-We assign types to the skeleton. The skeleton can be translated into a regulation expression by the following mapping:

$X \rightarrow bm^*|m^+$

$Y \rightarrow cX^+$

$Z \rightarrow Y^+$

4.4 Constructing a HMM from seed instances and description patterns

Here we construct a Hidden Markov Model (HMM) having states on types. Each edge in the state machine of the HMM is labeled with a transition probability on types. From the seed instances, we obtain transition probabilities as follows: We count how many times each subexpression matches, and determine the probabilities on edges originating from the same vertex. For example, regarding the pattern '[c[bm*|m+]+]+', the subexpression 'bm*' appears six times in the seed instance, and m+ appears twice. Except the alternation '[bm*|m+]', the state machine has only one outgoing edge, so we can assign probability 1 to these edges. But for the alternation '[bm*|m+]', the ratio between the first and second terms is 3:1. From this ratio we assign probabilities 0.75 and 0.25, respectively, to the two outgoing edges corresponding to the alteration. The smaller probability 0.25 represents the irregular case of 'm,m' without 'b'.

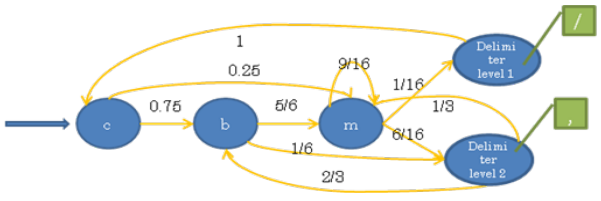

Figure 3 A pattern HMM M_0 with transition probabilities

We use this pattern automaton to construct an initial HMM M_0 (Figure 3) with transition probabilities by counting the proportion in the seed instances.

4.5 Generation of initial HMM

We will use Baum-Welch algorithm[1] to adjust the HMM for each new instance, and use Viterbi algorithm[8] to assign types to unknown entities. Baum-Welch algorithm is a well-known expectation maximization algorithm finding a maximum likelihood estimate of the parameters of a Hidden Markov Model given a set of observed sequences. It makes use of the forward-backward algorithm[1]. Viterbi algorithm finds most-likely sequences of hidden states by dynamic programming.

The initialization step needs to determine the transition and emission probabilities and the number of states. Although random transition and emission probabilities could work, a more-likely model can speed up the convergence. So we construct the following initial HMM M_0 as the input:

1) A HMM M_0 with state space S = $\{x_1,\ldots,x_k\}$, the initial probabilities π_i of being start at state i and transitional probabilities (i, j) of transitioning from state i to state j.

The initial HMM is constructed from the seed instances and description patterns. The initial probabilities π_i can be extracted from the seed instances as follows: If the seed instances always start from the root "Company", then $\pi_i = 1$ for "Company" and $\pi_j = 0$ for ($j <> i$). If there are multiple initial possible types, their relative frequencies shall be reflected on to their probabilities. Transitional probabilities are also from the seed instances through checking how many times each edge appear in the seed instances.

2) A set of enumerative descriptions $\{y_1, \ldots, y_T\}$. In our model, it corresponds to a set of product descriptions including seed instances and target (unknown) instances, like "*Apple iPhone 3/3s/4/4s, iPad 2/mini*". A subset of tokens are assigned types by the seed instances, and the reminders are delimiters and entities of unknown types.

3) Emission probabilities $\Pr(y_t|k)$. The emission probability $\Pr(y_t|k)$ is the probability of token y_t being observed in state k. We can calculate $\Pr(y_t|k)$ from the opposite direction $\Pr(k|\ y_t\ )$, namely, the possibility of in state k when y_t is observed. We apply Bayes' rule: $\Pr(y_t\ |k) = \Pr(k|\ y_t\ )\Pr(y_t\ )/\Pr(k)$.

Tokens included in the initial RDF graph (seed instances) should be given probability Pr(type|value) = 1. For example, since "Apple" is in the seed instance, we know that "Apple" type is "Company". This implies that Pr(Company|Apple) = 1.

We are approximating the distribution of the target instances by the distribution of the seed instances, since we do not know the exact distribution of the target. Pr(Apple) is calculated from in what ratio Apple occupies in the target instances. Likewise, Pr(Company) is calculated from in what ratio Company occurs in the seed instances. Then we can obtain the emission probability of Pr(Apple|Company).

There are tokens of unknown types in the target instances. We need to estimate emission probabilities of unassigned types. We also approximate the distribution of the target by the distribution of the seed instances. We assume that the state probabilities $\Pr(k)$ are identical between the target instances and the seed instances.

For example, in Figure 4, the description "*HTC sensation, XE, XL, One X S V*" has eight tokens. Each of the token probability

Pr(y_t) is calculated as 1/8 based on the target instance. We may merge target and seed instances to create a larger frequency distribution. Suppose that in the seed instances, c(Company), b(Brand) and m(Model) occur by the ratio: (0.2, 0,3, 0.5). We employ the ratio as the state probabilities Pr(k) for the target instances. Now we compute the emission probabilities Pr(y_t |k) by Bayes' rule. Here, there are two situations need to be considered. If we have no knowledge about the descriptions, Pr(k| y_t) = Pr(k) and Pr(y_t |k) = Pr(yt). Such as Pr(One|Company)=PR(One)= 1/8. But if we have partial knowledge, such as "HTC" is a company, then we can construct a better belief on Pr(k| y_t). We know that Pr(c|HTC) = 1, Pr(c) = 0.2 and Pr(HTC) = 1/8, then we can calculate that Pr(HTC|c) = 0.625. Furthermore, we can infer that Pr(b|HTC)=Pr(m|HTC)=0 and Pr(HTC|b)= Pr(HTC|m)=0.

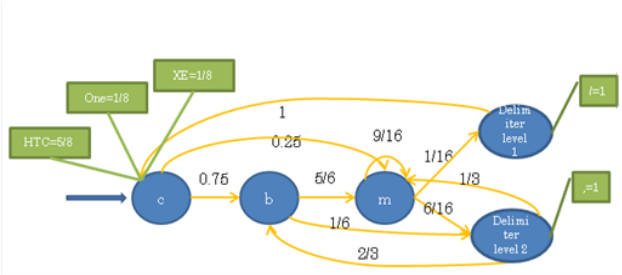

Figure 4 A HMM M_0 with emission probabilities

The target instances follow the hierarchical syntactic structure represented by the skeletons. But delimiters can be used in different levels among different seed instances, because descriptions may use different notations, depending on their origins. For example, only in the title of ID 1 in Table 1, colon ":" is used. We use skeletons to absorb such different usage of delimiters, because sometimes a placeholder in a different description is coupled with a different delimiter level. As shown in Figure 4, we first search a matching skeleton to find that the delimiter "/" is in level 1 and delimiter "," is in level 2. Then we assign the delimiter emission probabilities as Pr("/"|delimiter level 1)=1 and Pr(","|delimiter level 2)=1

After calculating all target token emission probabilities on one state, we normalize them and use this initial HMM in the initialization step of Baum-Welch algorithm.

In the expectation step, we need to compute the expected number of occurrences of each state transition and the expected number of occurrences of each symbol emission. Then we adjust the parameters to maximize the likelihood of these expected values.

We apply the forward and backward algorithms to compute Pr(q,i | S). Here, S is the token sequence in the product descriptions and q is a state in M_0. Pr(q,i | S) is the probability such that at *i-th* token s_i in S, the type of s_i is q. In the algorithm, we define:

$$\Pr(q, i \mid s) = \Pr(q, i, S) / \Pr(S)$$

$$\Pr(q, i, S) = \Pr(s_1 \ldots s_i, \text{state} = q) \Pr(s_{i+1} \ldots s_n \mid \text{state} = q)$$

For each type q and each token s_i, we compute Pr($s_1 \ldots s_i$, state = q) by the forward algorithm and we apply the backward algorithm to compute Pr($s_{i+1} \ldots s_n$ | state = q).

In our problem, the seed instances give constraints such that particular tokens are always assigned the types specified by the seed instances. Thus in the backward and forward algorithms, we apply Pr(q, i, S) = 1, if an seed instance gives that the correct type of s_i is q. The output of Baum-Welch algorithm is a converged HMM M_c.

4.6 Assigning types to unknown tokens

Now we discuss assigning types to unknown tokens. Let us consider the example: "*Samsung:c Galaxy:b 3:x,4:x*". Here, 'Samsung:c' means that the type of 'Samsung' is 'c' and the symbol 'x' means an unknown type. We need to find a type for each occurrence of 'x'.

In the final step of type assignment, we apply Viterbi algorithm to the converged HMM M_c to compute a type assignment on the target instances. Viterbi algorithm[8] solves the problem of finding an optimal state sequence (in our case, type assignment) for a given sequence (in our case, description) by dynamic programming.

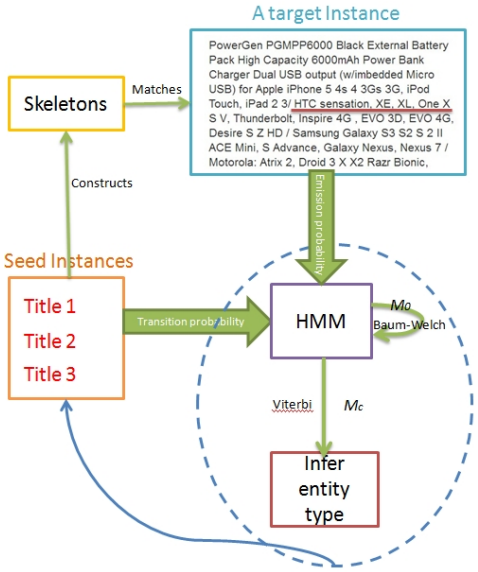

Figure 5 The whole process of inferring entity types

Figure 5 shows the picture of the whole process. We construct a basic HMM M_0 from a set of seed instances and apply Baum-Welch algorithm to update the parameters of the HMM. Then we apply Viterbi algorithm to infer entity types. After each iteration, we add the new type assignments with confidence score to the seed instances. We execute this process at each arrival of new target instances.

5. EXPERIMENTS

5.1 Benchmark data set

We collect smart device and accessory descriptions from various e-commerce websites randomly, and extract product titles from these web pages to construct a test data set of product descriptions. We collected 2892 product descriptions and the number of descriptions at each website is shown in Table 2.

Table 3 shows the distribution of production descriptions in different product categories. We collect 2636 product descriptions of popular phones and tablets appearing in the default sort order in e-commerce websites. We use these phone and tablet product descriptions to extract phone and tablet entities as the ground truth, to be used to verify inferred types. Besides, we construct phone and tablet type hierarchies from the entity listing orders in these descriptions. On the other hand, we use the other 256 smart device

accessories, which are randomly collected from the e-commerce websites, as the target instances. Each accessory description contains an enumerative description consisting of compatible phone entities.

We identified 68 distinct entities from 256 accessory descriptions. On the contrary, we identified 47 distinct entities from 2623 phone and tablet descriptions. Since phone and tablet descriptions in the e-commerce sites are dealing with popular and best-selling models, the entities found in these descriptions are heavily duplicated. Also, these descriptions contain one phone/table entity and not enumerative. We manually assign (entity, type) pairs extracted from the phone and tablet descriptions and use them as ground truth. Although there are two entities in the phone descriptions that are missing from the accessory descriptions, popular and general models have been found in the accessory descriptions.

Table 2 Website distribution of product descriptions

Website	#of descriptions
Amazon	1045
Ebay	1428
Rakuten	134
Walmart	23
BestBuy	262
Total	2892

Table 3 Category distribution of product descriptions

Category	#of descriptions
Phone	1415
Tablet	1221
Battery pack	78
Case	53
Stylus pen	33
Headset	61
USB cable	31
Total	2892

5.2 Result Analysis

We compare the results inferred by our method from accessories with the ground truth. Our current implementation for extracting skeletons and generating description patterns requires manual supervision. In the 256 accessory descriptions, seven skeletons matching different description structures are found. The number of entities in one description ranges from 1 to 40, and its average is 17. Each skeleton is assigned with one to three description patterns.

We compared our proposed method with two baseline methods. One is such that all delimiters in the enumerative descriptions are replaced with white spaces, so no syntactic feature can be utilized. The other baseline prohibits update of the HMM in BW algorithm, so that common traits across skeletons are not propagated.

We change the size of the seed instances to evaluate accuracy, which is defined as the fraction of correct type assignments over the whole type assignments. We executed five times for each size and took the average. As the size of seed instances increases, in order of 10%, 20%, 30%,... 90%, the accuracy is expected to become higher. On the contrary, the efficiency of discovering new entities in terms of the size of the seed instances is expected to become lower. We need to find the appropriate proportion of the seed instances that realizes a reasonable accuracy.

Figure 6 shows the accuracy results. All the three methods show a rising trend with the increase of the proportion of the seed instances. The green curve is the proposed method, showing the highest accuracies. The proposed method achieves a significant rise from 0 to 0.6, and then the curve becomes almost flat. There is a trade-off between accuracy and efficiency. We obtained an acceptable result at 0.6 of accuracy around 95%. The baseline which ignores the delimiter (skeleton) information is shown as the red curve, which is close to linear. The blue curve is the baseline with no HMM update in BW algorithm. The accuracies of these baselines are much lower than our method, especially when the seed instance size is larger than 50%.

Figure 6 Accuracy vs. seed instance size

6. CONCLUSION

In this paper, we proposed an iterative process for entity type resolution on enumerative descriptions. Regularities of enumerative descriptions are captured by skeletons, and description patterns are generated to give candidate type assignments. Then a HMM is constructed from description patterns and seed instances, to generate maximum-likelihood type assignments. As future work, we aim to extend performance evaluation of the proposed method over different types of enumerative descriptions, such as tables in Wikipedia, and study application to various types of partially-typed web resources.

7. REFERENCES

[1] L. E. Baum. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a Markov process. Inequalities, 3, 1972.

[2] X. L. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. VLDBJ, 18(2), 2009.

[3] H. Huang, C. Liu. Query evaluation on probabilistic RDF databases. In WISE, 2009.

[4] X. Lian and L. Chen. Efficient query answering in probabilistic RDF graphs. SIGMOD'11, 2011.

[5] T. Y. Li, P. Chubak, L.V.S. Lakshmanan. Efficient extraction of ontologies from domain specific text corpora. CIKM'12, 2012.

[6] Y. Ma, T. Tra, V. Bicer. TYPifier: inferring the type semantics of structured data. ICDE '13, 2013.

[7] A.D. Sarma, A.Jain, D. Srivastava. I4E: interactive investigation of iterative information extraction. SIGMOD'10, 2010.

[8] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," IEEE Trans. Inform. Theory,IT-13:260-269, Apr 1967.

[9] X. Zhang, G. Cheng, Y. Z. Qu. Ontology summarization based on RDF sentence graph. In IW3C2, 2007.

[10] P. Zhao and J. Han. On graph query optimization in large networks. PVLDB, 3(1), 2010.