Automatic Detection of Name Disambiguation and Extracting Aliases for the Personal Name

G. Tireesha Kumari[1], Mr. Saroj Kumar Gupta[2]
[1] M.Tech Scholar, CSEDepartment, Madanapalle
[2] Assistant Professor, CSEDepartment,
Madanapalle Institute of Technology & science, JNTUA, India

Abstract :
An individual can be referred by multiple name aliases on the web. Extracting aliases of a name is important in information retrieval, sentiment analysis and name disambiguation. We propose a novel approach to find aliases of a given name using automatically extracted lexical pattern based approach. We exploit set of known names and their aliases as training data and extract lexical patterns that convey information related to aliases of names and extract large set of candidate aliases from text snippets returned by web search engine. We define numerous ranking scores to evaluate candidate aliases using three approaches: lexical pattern frequency, word co-occurrences in an anchor text and page counts on the web. We introduce notion of a word co-occurrence graph to represent mutual relations between words that appear in anchor text, words in anchor text are represented as nodes in the co-occurrence graph and edge is formed between nodes which link to the same url. The drawback of the existing method is the extracted alias names may be a original of some other person. So we introduce Email id extraction, by this we can overcome the problem. To construct a robust alias detection system, we integrate ranking scores through support vector machines using a single ranking function. Moreover, the aliases extracted using the proposed method are successfully utilized in information retrieval task to improve recall by 20 percent in a relation detection task.

Keywords - Web mining, information extraction, Relation Extraction

1. Introduction

Searching for information about people in the web is one of the most common activities of internet users. Around 30 percent of search engine queries include person names [1], [2]. However, retrieving information about people from web search engines can become difficult when a person has nicknames or name aliases. For example, the famous Japanese major league baseball player Hideki Matsui is often called as Godzilla on the web. A newspaper article on the baseball player might use the real name, Hideki Matsui, whereas a blogger would use the alias, Godzilla, in a blog entry. We will not be able to retrieve all the information about the baseball player, if we only use his real name.Identification of entities on the web is difficult for two fundamental reasons: first, different entities can share the same name (i.e., lexical ambiguity); second, a single entity can be designated by multiple names (i.e., referential ambiguity). For example, the lexical ambiguities consider the name Jim Clark. Aside from the two most popular namesakes, the formula-one racing champion and the founder of Netscape, at least 10 different people are listed among the top 100 results returned by Google for the name.

On the other hand, referential ambiguity occurs because people use different names to refer to the same entity on the web. For example, the American movie star Will Smith is often called the Fresh Prince in web contents. Although lexical ambiguity, particularly ambiguity related to personal names has been explored extensively in the previous studies of name disambiguation [3], [4], the problem of referential ambiguity of entities on the web has received much less attention. In this paper, we specifically examine on the problem of automatically extracting the various references on the web of a particular entity. We propose a fully automatic method to discover aliases of a given personal name from the web. Our contributions can be summarized as follows:

We propose a lexical pattern-based approach to extract aliases of a given name using snippets returned by a web search engine. The lexical patterns are generated automatically using a set of real world name alias data. We evaluate the confidence of extracted lexical patterns and retain the patterns that can accurately discover aliases for various personal names. Our pattern extraction algorithm does not assume any language specific preprocessing such as part-of-speech tagging or dependency parsing, etc., which can be both inaccurate and computationally costly in web-scale data processing.

To select the best aliases among the extracted candidates, we propose numerous ranking scores based upon three approaches: lexical pattern frequency, word co-occurrences in an anchor text graph, and page counts on the Web. Moreover

Using real-world name alias data, We train a ranking support vector machine to learn the optimal combination of individual ranking scores to construct a robust alias extraction method.

2. Related Work

Alias identification is closely related to the problem of cross-document coreference resolution in which the objective is to determine whether two mentions of a name in different documents refer to the same entity. Bagga and Baldwin [10] proposed a cross-document coreference resolution algorithm by first performing within document coreference resolution for each individual document to extract coreference chains, and then, clustering the coreference chains under a vector space model to identify all mentions of a name in the document set. However, the vastly numerous documents on the web render it impractical to perform within document coreference resolution to each document separately, and then, cluster the documents to find aliases.

In personal name disambiguation the goal is to disambiguate various people that share the same name (namesakes) [3], [4]. Given an ambiguous name, most name disambiguation algorithms have modeled the problem as one of document clustering in which all documents that discuss a particular individual of the given ambiguous name are grouped into a single cluster. The web people search task (WePS)1 provided an evaluation data set and compared various name disambiguation systems. However, the name disambiguation problem differs fundamentally from that of alias extraction because in name disambiguation the objective is to identify the different entities that arereferred by the same ambiguous name; in alias extraction, we are interested in extracting all references to a single entity from the web.

3. Method

The method is outlined in Figure 1 and it comprises two main components: pattern traction and alias extraction and ranking. Using Seed list of name-alias pairs, we first extract lexical patterns that are frequently used to convey information related to aliases on the web. The extracted patterns are then used to find aliases for a given name. We define various ranking scores using the hyperlink structure on the web and page counts retrieved from a search engine to identify the correct aliases among the extracted candidates.

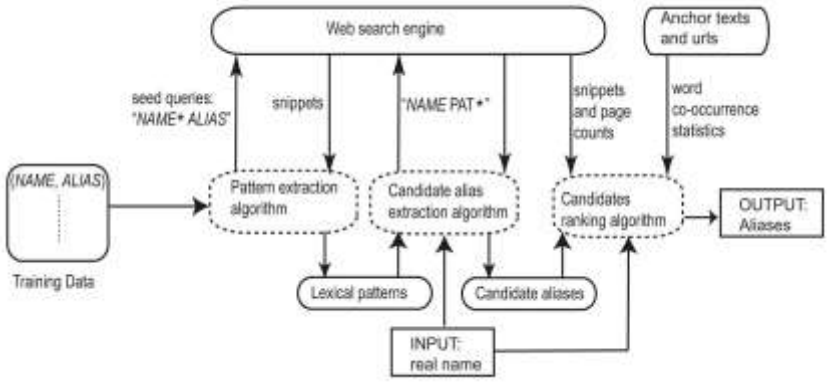

Figure 1. Outline of the Method

3.1. Extracting Lexical Patterns from Snippets

Many modern search engines provide a brief text snippet for each search result by selecting the text that appears in the web page in the proximity of the query. Such snippets provide valuable information related to the local context of the query. For names and aliases, snippets convey useful semantic clues that can be used to extract lexical patterns that are frequently used to express aliases of a name. For example, consider the snippet returned by Google for the query "Will Smith * The Fresh Prince. "Here we use the wildcard operator * to perform a NEAR query and it mateches with one or more words in a snippet. In Figure 2 the snippet contains aka(i.e., also known as), which indicates the fact that fresh prince is an alis for Will Smith. In addition to aka numerous clues exist such as, nicknamed, alias, real name is nee, which are used on the web to represent the aliases of a name. Consequently, we propose shallow lexical pattern extraction method in Fig.3 to capture the various ways in which information about aliases of names is expressed on the web.

Given a set of (NAME, ALIAS) pairs, the function ExtractPatterns returns a list of lexical patterns that frequently connect names and their aliases in web snippets. For each (NAME, ALIAS) pair in S, the GetSnippets function downloads snippets

from a web search engine for the query "NAME * ALIAS" . Then, from each snippet, the CreatPattern function extracts the sequence of words that appear between the name and the alias.

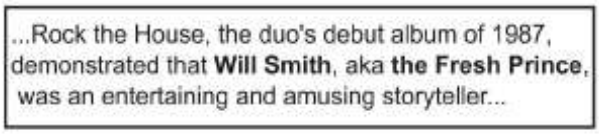

Figure 2. A snippet returned for the query "Will Smith * the Fresh Prince" by Google

Algorithm 3.1: EXTRACTPATTERNS(S)

comment: S is a set of (NAME, ALIAS) pairs

$P \leftarrow null$
for each $(NAME, ALIAS) \in S$
do $\begin{cases} D \leftarrow \text{GetSnippets}(\text{“}NAME \ * \ ALIAS\text{”}) \\ \textbf{for each} \text{ snippet } d \in D \\ \quad \textbf{do } P \leftarrow P + \text{CreatePattern}(d) \end{cases}$
return (P)

Figure 3. Given a set of (NAME, ALIAS) instances, extract lexical patterns.

Once a set of lexical patterns is extracted, we use the patterns to extract candidate aliases for a given name as portrayed in Figure 4 Given a name, NAME and a set, P of lexical patterns, the function ExtractCandidates returns a list of candidate aliases for the name. We associate the given name with each pattern, p in the set of patterns, P and produce queries of the form: "NAME p*." Then, the GetSnippets function downloads a set of snippets for the query. Finally, the GetNgrams function extracts continuous sequences of words (n-grams) from the beginning of the part that matches the wildcard operator *.

Moreover, we removed candidates that contain only stop words such as a, an, and the. For example, assuming that we retrieved the snippet in Figure 2 for the query "Will Smith aka*," the procedure described above extracts the fresh and the fresh prince as candidate aliases.

Algorithm 3.2: EXTRACTCANDIDATES($NAME, P$)

comment: P is the set of patterns

$C \leftarrow null$
for each pattern $p \in P$
do $\begin{cases} D \leftarrow \text{GetSnippets}(\text{“}NAME \ p \ *\text{”}) \\ \textbf{for each} \text{ snippet } d \in D \\ \quad \textbf{do } C \leftarrow C + \text{GetNgrams}(d, NAME, p) \end{cases}$
return (C)

Figure 4. Given a name and a set of lexical patterns, extract candidate aliases.

3.2. Ranking of Candidates

Considering the noise in web snippets, candidates extracted by the shallow lexical patterns might include some invalid aliases. From among these candidates, we must identify those, which are most likely to be correct aliases of a given name. We model this problem of alias recognition as one of ranking candidates with respect to a given name such that the candidates, who are most likely to be correct aliases are scores to measure the association between a name and a candidate alias using three different approaches: lexical pattern frequency, word co-occurrences in an anchor text graph, and page counts on the web.

3.3. Lexical Pattern Frequency

In Section 3.1 we presented an algorithm to extract numerous lexical patterns that are used to describe aliases of a personal name. The proposed pattern extraction algorithm can extract a large number of lexical patterns. If the personal name under consideration and a candidate alias occur in many lexical patterns, then it can be considered as a good alias for the personal name. Consequently, we rank a set of candidate aliases in the descending order of the number of different lexical patterns in which they appear with a name. The lexical pattern frequency of an alias is analogous to the document frequency (DF) popularly used in information retrieval.

Figure 5. A picture of Mahindra Singh Dhoni being linked by different anchor texts on the web

3.4. Co-Occurrences in Anchor Texts

Anchor texts have been used extensively in information retrieval and have been used in various tasks such as synonym extraction, query translation in cross-language information retrieval, and ranking and classification of web pages. Anchor texts are particularly attractive because they not only contain concise texts, but also provide links that can be considered as expressing a citation. We revisit anchor texts to measure the association between a name and its aliases on the web. Anchor texts pointing to a url provide useful semantic clues related to the resource represented by the url. For example, if the majority of inbound anchor texts of a url contain a personal name, it is likely that the remainder of the inbound anchor texts contain information about aliases of the name. Here, we use the term inbound anchor texts to refer the set of anchor texts pointing to the same url.

For example, consider the picture of Mahindra Singh Dhoni shown in Figure 5. Figure 5 shows a picture of Mahindra Singh Dhoni being linked to by four different anchor texts. According to our definition of co-occurrence, M.S. Dhoni , and spunky are considered as co-occurring.

3.5. Page Count Based Association Measure

However, not all names and aliases are equally well represented in anchor texts. Consequently, in this section, we define word association measures that consider co-occurrences not only in anchor texts but in the web overall. Page counts retrieved from a web search engine for the conjunctive query, “p and x,” for a name p and a candidate alias x can be regarded as an approximation of their co occurrences in the web. We compute popular word association measures using page counts returned by a search engine.

3.5.1. WebDice

We compute the Dice, WebDice(p,x)) between a name p and a candidate alias x using page counts as

$$WebDice(p,x) = \frac{2 \times hits(\text{"p AND x"})}{hits(p) + hits(x)} \quad (1)$$

Here, hits (q) are the page counts for the query q.

3.5.2. WebPMI

We compute the PMI, WebPMIðp; xÞ using page counts as follows:

$$WebPMI(p,x) = \log 2 \frac{L \times 2 \times hits(\text{"p AND x"})}{hits(p) + hits(x)} \quad (2)$$

Here, L is the number of pages indexed by the web search engine, which we approximated as L= 10^{10} according to the number of pages indexed by Google. It should be noted however that the actual value of L is not required for ranking purposes because it is a constant and can be taken out from the definition of WebPMI .

3.5.3. Conditional Probability

Using page counts, we compute the probability of an alias,given a name as

$$prob(x|p) = \frac{hits("p\ AND\ x")}{hits(p)} \quad (3)$$

Similarly, the probability of a name, given an alias is

$$prob(p|x) = \frac{hits("p\ AND\ x")}{hits(x)} \quad (4)$$

Unlike PMI and the Dice, conditional probability is an asymmetric measure.

4. Experiments and Results

4.1 Pattern Extraction

As shown in Figure 2 extracts patterns for the personal names. However, not all patterns are equally informative about aliases of a real name. Consequently, we rank the patterns according to their F -scores to identify the patterns that accurately convey information about aliases. F -score of a pattern s is computed as the harmonic mean between the precision and recall of the pattern. First, for a pattern s, we compute its precision and recall as follows:

$$Precision(s) = \frac{No.of\ correct\ aliases\ retrieved\ by\ s}{No.of\ total\ aliases\ retrieved\ by\ s}$$

$$Recall(s) = \frac{No.of\ correct\ aliases\ retrieved\ by\ s}{No.of\ total\ aliases\ in\ the\ dataset}$$

Then, its F-score can be computed as

$$F(s) = \frac{2 \times Precision(s) \times Recall(s)}{Precision(s) + Recall(s)}$$

Table 1
Lexical Patterns for the personal names

Pattern	F-score
* aka [NAME]	0.335
[NAME] aka *	0.322
[NAME] better known as *	0.310
[NAME] alias *	0.286
[NAME] also known as *	0.281
* nee [NAME]	0.225
[NAME] nickname *	0.224
* whose real name is [NAME]	0.205
[NAME] aka the *	0.187
* was born [NAME]	0.153

Table 1 shows the patterns with the highest F –scores extracted using personal names. As shown in Table 1 unambiguous and highly descriptive patterns are extracted by the proposed method.

4.2. Alias Extraction

Alias extraction can be done using the Mean reciprocal Rank . MRR is defined as follows:

$$MRR = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{Ri} \quad (5)$$

Therein: Ri is the rank assigned to a correct alias and n is the total number of aliases. AP is defined as follows:

$$AP = \frac{\sum_{r=1}^{k} Pre(r) \times Rel(r)}{No.of\ correct\ aliases} \quad (6)$$

Here, Rel(r) is a binary valued function that returns one if the candidate at rank r is a correct alias for the name. Otherwise, it returns zero. Furthermore, Pre(r) is the precision at rank r, which is given by

$$Pre(r) = \frac{no.of\ correct\ aliases\ in\ top\ r\ candidates}{r} \quad (7)$$

Table 2
Aliases Extracted by the Proposed Method

Real Name	Extracted Aliases
Sachin Tendulkar	**Little Master , Tendlya, Master Blaster , The Bombay**
Sourav Ganguly	**Maharaj , Dada , The Warrier Prince,** Lord Snooty
Virender Sehwag	**Veeru , Sultan of Multan , Loose Kartoose , Zen master of Modern Cricket**
Yuvraj Singh	**Grrovy Yuvi , Captain Kamal ,** UV , Bowlers Nightmare
Rahul Dravid	**The Wall , Jammy , Mr. Dependable ,** The Lc , The Great Wall of India

Table 2 presents the aliases extracted for some entities included in our data sets. Overall, the proposed method extracts most aliases in the manually created gold standard (shown in bold). It is noteworthy that most aliases do not share any words with the name nor acronyms, thus would not be correctly extracted from approximate string matching methods.

Table 3
Email Information Extracted by the Proposed Method

Real Name	Extracted Email Ids
Sachin Tendulkar	**Sachin.tendulkar@gmail.com,tendlya.sachin@yahoo.com**
Sourav Ganguly	**Maharaj519@yahoo.com, dada.spurav@rocketmail.com**
Virender Sehwag	**Veeru@ gmail.com, Indian.sehwag@gmail.com**
Yuvraj Singh	**Grrovy Yuvi@gmail.com, yuvraj.singh@gmail.com**
Rahul Dravid	**Rahul.dravid@rocketmail.com, ependable.india@gmail.com**

Table 3 presents the email ids extracted for some entities included in our data sets. Overall, the proposed method extracts most Email Information. The extracted Email information should be useful to identify the aliases for a personal name.

5. Implementation Considerations

We describe two independent approaches to efficiently extract all anchor texts pointing to a particular url.

5.1. Extracting Inbound Anchor Texts from a Web crawl

In the case where, we have a large web crawl at our disposal, then extracting all the inbound anchor texts of an url can be accomplished using a hash table, where the hash key is the url and the corresponding value contains a list of anchor texts that point to the url specified by the key. Fig. 6 illustrates the pseudocode to achieve this task. Given a web crawl, we extract the set L of links from the crawled data, where L contains tuples ðai; uiÞ of anchor texts ai and the urls ui pointed by the anchor texts. The function ANCHORS in Figure 6 processes the links in L and returns a hash H, where keys are urls ui and values, H[ui] are lists of anchor texts pointing to ui. H is initialized to an empty hash and for each tuple (ai; ui) in L, if H already contains the key ui, then the function AppendðH([ui], ai) appends the anchor text ai to the list H[ui]. If ui does not exist in H, then a new list containing ai is created and assigned to key ui. If the number of links to be processed is large, then H can be implemented as a distributed hash table.

5.2. Retrieving Inbound Anchor Text from a Web Search Engine

Crawling and indexing a large set of anchor texts from the web requires both computation power and time. However, there already exist large web indexes created by major web search providers such as Google, Yahoo, and MSN. Most web search engines provide query APIs and search operators to search in anchor texts. For example, Google provides the search operator inanchor to retrieve urls pointed by anchor texts that contain a particular word.Moreover, the search operator link returns urls which are linked to a given url. In algorit, we describe an algorithm to extract a set of anchor texts that points to urls, which are also pointed by a given name using the basic search operators provided by a web search engine

Algorithm 5.1: ANCHORS(L)

comment: Extract inbound anchor texts from L.

$H = \{\}$
for each $(a_i, u_i) \in L$
do { **if** $u_i \in H$
  then $Append(H[u_i], a_i)$
  else $H[u_i] \leftarrow [\,]$
return (H)

Figure 6. Extract inbound anchor texts from a web crawl

. Given a name p, the function ANCHORS (p) described in Figure 7 returns a hash H, where keys of H are urls and the corresponding values contain lists of anchor texts pointing to the url specified by the key. Moreover, for each url in H, at least one of the retrieved anchor texts contains the name p. Therefore, all the words that appear in anchor texts extracted by the algorithm contains candidate aliases of the name p. The algorithm first initializes the hash H to empty hash. Then, the function InAnchor(p) retrieves urls that are pointed by anchor texts that contain p. For example,in Google, the search operators inanchor or allinanchor can be used for this purpose. For each url, ui retrieved by the function. In Anchor, we initialize the list of anchor texts to empty list and retrieve the set of pages T that link to ui using the function Link(ui). For example, in Google, the search operator link provides the desired functionality. Then, for each page t in T the function ExtractLinks extracts anchor texts and links from t. Finally, we accumulate anchor texts that point to ui and store them as a list in H.

Algorithm 5.2: ANCHORS(p)

comment: Extract inbound anchor texts for p

$H = \{\}$
$S \leftarrow InAnchor(p)$
for each url $u_i \in S$
do { $H[u_i] \leftarrow [\,]$
  $T \leftarrow Link(u_i)$
  for each page $t \in T$
  do { $L \leftarrow ExtractLinks(t)$
    for each $(a_k, q_k) \in L$
    do { **if** $q_k = u_i$
      then $Append(H[u_i], a_k)$
return (H)

Figure 7. Get inbound anchor texts from a web search engine for a name p.

6. Conclusion

We proposed a lexical-pattern-based approach to extract aliases of a given name. We use a set of names and their aliases as training data to extract lexical patterns that describe numerous ways in which information related to aliases of a name is presented on the web. Next, we substitute the real name of the person that we are interested in finding aliases in the extracted lexical patterns, and download snippets from a web search engine. We extract a set of candidate aliases from the snippets. The candidates are ranked using various ranking scores computed using three approaches: lexical pattern frequency, co-occurrences in anchor texts, and page counts-based association measures. Moreover, we integrate the different ranking scores to construct a single ranking function using ranking support vector machines. In future we intend to apply this to extract the Email information about the person, a co-occurrence graph should be constructed to represent words in anchor texts and modeled the problem of alias extraction as a one of ranking nodes in this graph with respect to a given name. We intend to apply the proposed method to extract aliases for other entity types such as products, organizations and locations. Moreover, the extracted aliases significantly improved recall in a relation detection task and render useful in a web search task.

References

[1] R. Guha and A. Garg, "Disambiguating People in Search," technical report, tanford Univ., 2004.

[2] J. Artiles, J. Gonzalo, and F. Verdejo, "A Testbed for People Searching Strategies in the WWW," Proc. SIGIR '05, pp. 569-570,2005.

[3] G. Mann and D. Yarowsky, "Unsupervised Personal Name Disambiguation," Proc. Conf. Computational Natural Language Learning (CoNLL '03), pp. 33-40, 2003.

[4] R. Bekkerman and A. McCallum, "Disambiguating Web Appearances of People in a Social Network," Proc. Int'l World Wide Web Conf. (WWW '05), pp. 463-470, 2005.

[5] G. Salton and M. McGill, Introduction to Modern Information Retreival. McGraw-Hill Inc., 1986.

[6] M. Mitra, A. Singhal, and C. Buckley, "Improving Automatic Query Expansion," Proc. SIGIR '98, pp. 206-214, 1998.

[7] P. Cimano, S. Handschuh, and S. Staab, "Towards the Self-Annotating Web," Proc. Int'l World Wide Web Conf. (WWW '04),2004.

[8] M. Bilenko and R. Mooney, "Adaptive Duplicate Detection Using Learnable String Similarity Measures," Proc. SIGKDD '03, 2003.[13] T. Hokama and H. Kitagawa, "Extracting Mnemonic Names of People from the Web," Proc. Ninth Int'l Conf. Asian Digital Libraries (ICADL '06), pp. 121-130, 2006.

[9] M. Hearst, "Automatic Acquisition of Hyponyms from Large Text Corpora," Proc. Int'l Conf. Computational Linguistics (COLING '92),pp. 539-545, 1992.

[10] M. Berland and E. Charniak, "Finding Parts in Very Large Corpora," Proc. Ann. Meeting of the Assoc. for Computational Linguistics (ACL '99), pp. 57-64, 1999.

[11] S. Chakrabarti, Mining the Web: Discovering Knowledge from Hypertext Data. Morgan Kaufmann, 2003.

[12] T. Dunning, "Accurate Methods for the Statistics of Surprise and Coincidence," Computational Linguistics, vol. 19, pp. 61-74, 1993.

[13] K. Church and P. Hanks, "Word Association Norms, Mutual Information and Lexicography," Computational Linguistics, vol. 16, pp. 22-29, 1991.

[14] T. Hisamitsu and Y. Niwa, "Topic-Word Selection Based on Combinatorial Probability," Proc. Natural Language Processing Pacific-Rim Symp. (NLPRS '01), pp. 289-296, 2001.