Extracting structured information from Wikipedia articles to populate infoboxes

Article · November 2010

DOI: 10.1145/1871437.1871698

3 authors, including:

Dustin Lange
Universität Potsdam

Felix Naumann
Hasso Plattner Institute

Extracting Structured Information from Wikipedia Articles to Populate Infoboxes *

Dustin Lange Christoph Böhm Felix Naumann
Hasso Plattner Institute, Potsdam, Germany
firstname.lastname@hpi.uni-potsdam.de

ABSTRACT

Roughly every third Wikipedia article contains an infobox – a table that displays important facts about the subject in attribute-value form. The schema of an infobox, i.e., the attributes that can be expressed for a concept, is defined by an infobox template. Often, authors do not specify all template attributes, resulting in incomplete infoboxes.

With iPopulator, we introduce a system that automatically populates infoboxes of Wikipedia articles by extracting attribute values from the article's text. In contrast to prior work, iPopulator detects and exploits the structure of attribute values to independently extract value parts. We have tested iPopulator on the entire set of infobox templates and provide a detailed analysis of its effectiveness. For instance, we achieve an average extraction precision of 91% for 1,727 distinct infobox template attributes.

Keywords

Information Extraction, Linked Data, Wikipedia

1. WIKIPEDIA INFOBOXES

Wikipedia is a free, collaborative encyclopedia with a huge impact. Since its foundation in 2001, Wikipedia has become one of the most popular web sites in the world. As of May 2010, the English version of Wikipedia contained almost 3.3 million articles. Wikipedia articles are expected to offer an overview of its subject at the beginning of the article. Thus, the article text usually starts with a definition, a summary, or a short description of the subject. Often, a box next to the summary offers structured information about the article's subject in table form. These so-

An extended version of this paper is available [2].

CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.
Copyright 2010 ACM 978-1-4503-0099-5/10/10 ...$10.00.

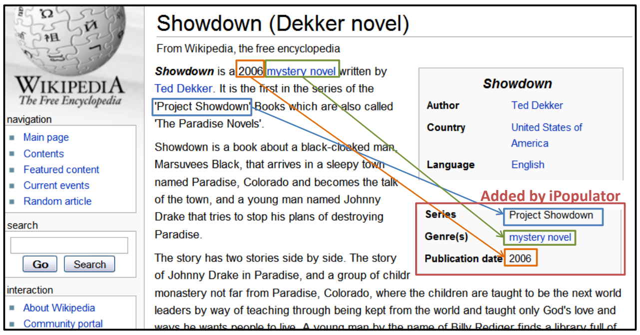

Figure 1: Example for infobox and for infobox attribute values extracted by iPopulator

called infoboxes contain facts about the described subject, displayed as attribute-value pairs. Figure 1 shows an example. The text summary and the infobox allow readers to rapidly gather the most important information about the article's subject.

Infobox creation is based on templates. In Wikipedia, a template works similar to a function: it receives parameters that can be viewed as values of attributes, and it has a well-defined return value, namely the Wikipedia source text. The template's attributes describe information about instances of a specific concept, and the template's return value contains the source text necessary to display the box and its content in table form.

Often, an infobox does not contain as much information as possible, i.e., the infobox template call does not specify values for all of the template's attributes. For example, the original infobox shown in Fig. 1 contains only few values for the `infobox_book` template, namely `author`, `country`, and `language`. Among others, no values for the attributes `genre` and `publication date` have been specified.

To complete missing infobox attribute values, we propose to examine the article text. Often, article texts contain some of the values specified in the infobox. Figure 1 shows an example: For several infobox attributes, the specified values are contained in the article text, such as the title and author of the book, its genre, and its publication year.

Problem Statement. Given a Wikipedia article containing an incomplete infobox template call, the *Infobox Population Problem* is to extract as many correct attribute values from the article text as possible. We say an infobox template call is *incomplete* if some attributes are unspecified. The problem is restricted to the extraction from Wikipedia article texts; no external sources are used. Note that there is no limitation to a specific set of infobox templates or to a specific domain. A system should be able to adapt to any given infobox template, i.e., the system should be able to extract attribute values for all attributes in all infobox templates.

The remainder of this paper is structured as follows: Section 2 gives an overview of related work. Section 3 presents our extraction system including structural analysis, training data creation, CRF training, and the final extraction step. An evaluation is given in Sec. 4. Finally, Sec. 5 concludes this paper. An extended version of this paper is [2].

2. RELATED WORK

We focus on similar approaches here; other related work is discussed in [2].

Wu and Weld [6] propose Kylin as a system for automatically generating infoboxes from Wikipedia articles. Our system is in parts similar to Kylin, but offers important improvements. Kylin merely labels exact occurrences of attribute values for training. By applying fuzzy matching to infobox attribute values and their text occurrences, iPopulator can find more occurrences (+23%). Dividing the attribute value into significant substrings allows even more occurrences to be found (+31%). To select the input of the attribute value extractor, our system reads only the first paragraphs of an article as a basic heuristic, while Kylin uses sentence classifiers. Both Kylin and iPopulator employ CRFs for the extraction process: Kylin uses CRFs to extract entire attribute values, while iPopulator uses them to extract attribute value parts according to prior discovered value structures. In contrast to Kylin, iPopulator is able to reconstruct the structure of attribute values by aligning the extracted attribute value parts and inserting structural elements. Wu and Weld chose four specific infobox templates for their experiments, in which Kylin achieved precision of 0.74 to 0.97 and recall of 0.61 to 0.96. In our work, we evaluated extraction for *all* Wikipedia templates and achieve average precision of 0.91 and average recall of 0.66.

In a later work, Wu et al. [5] present an improved version of Kylin that considers shrinkage, ontologies, and web search results. These improvements result in an increase in recall of up to 50.8% while maintaining precision. In Sec. 4.2, we compare the results of Kylin and its successor with those of our work in more detail.

3. EXTRACTING ATTRIBUTE VALUES

iPopulator's extraction process is shown in Fig. 2. For each infobox template, the following steps are applied using articles that contain an infobox of this type as training data:

(1) Structure Analysis: For each attribute of the infobox template, we analyze its values given in the training articles' infoboxes to determine a structure that represents the attribute's syntactical characteristics.

(2) Training Data Creation: For this step, we use articles that specify a value for an attribute as training data. Occurrences of attribute values within the training article texts are labeled.

(3) Value Extractor Creation: The labeled training data are used to generate extractors for as many attributes as possible. We employ Conditional Random Fields (CRFs) to generate attribute value extractors. These extractors are automatically evaluated, so that ineffective extractors can be discarded.

(4) Attribute Value Extraction: The extractors can then be applied to all articles to find missing attribute values for existing infoboxes.

In the following, we provide details on the different steps.

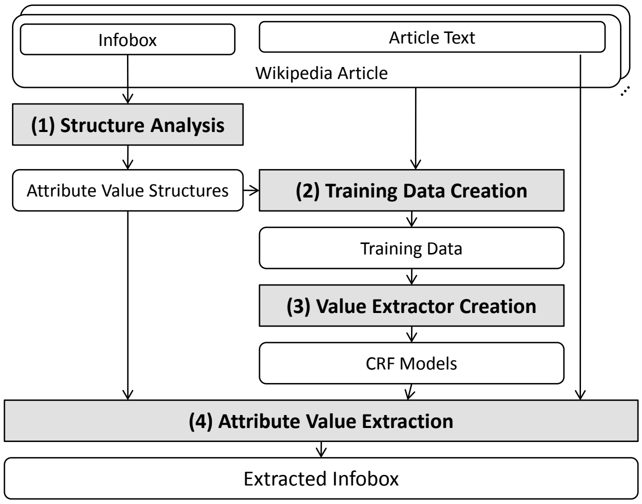

Figure 2: iPopulator extraction process

3.1 Structure Analysis

Many attributes have a characteristic structure. For example, a value for the `infobox_company` attribute `number_of_employees` might be `12,500 (2003)`, which means that 12,500 people were employed in 2003. Many other values of this particular attribute have a similar structure of the form (Number '(' Number ')'). Further, many attributes are multi-valued, such as `Bill Gates, Paul Allen` for the `founder` attribute. iPopulator discovers attribute value structures and exploits them both for training data and attribute value construction.

In the following, we present an algorithm that analyzes available values of an attribute and discovers a structure that represents most of these values and is still easy to process, i.e., simple, but powerful enough to split values and to combine value parts.

Structure discovery algorithm. We have developed a new algorithm that addresses shortcomings of regular expression learning algorithms. The main steps are shown in Fig. 3 and explained in more detail in the following. Tables I to IV in Fig. 4 illustrate the algorithm by means of the attribute `number_of_employees` from `infobox_company`. At first, the algorithm determines patterns for all values of an attribute by parsing them (Step (a)). These patterns are then counted and sorted by their frequency (Step (b)). Then the important patterns are merged into the result expression (Step (c)), starting with the most frequent ones.

3.2 Training Data Creation

Training data are a prerequisite for any machine learning technique. In our case, we need to spot and label attribute value occurrences in Wikipedia article texts. Such a search is not an easy task, because attribute values usually do not

```
DiscoverAttributeValueStructure(attribute values V)
  (a) Parsing
  For each value v in V:
      Create pattern p = Parse(v)
      Add pattern to pattern list L
  (b) Counting & Sorting
  Group same patterns in L and sort by frequency (descending order)
  (c) Merging
  Initialize result expression r with most frequent pattern
  For each remaining pattern p in L:
      If IsImportantPattern(p):
          Merge(r, p) next frequent pattern p into current result r
  Return r
```

Figure 3: Structure discovery algorithm

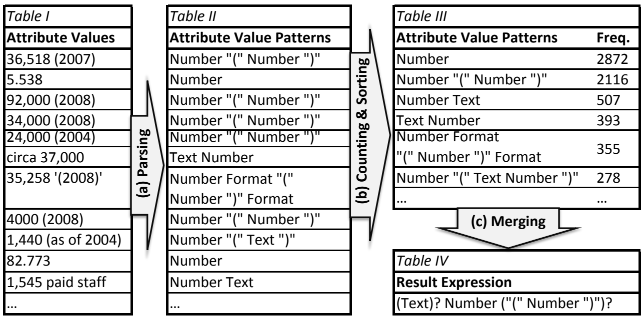

Figure 4: Example for applying the structure discovery algorithm (Fig. 3) to values of the attribute `number_of_employees` **from** `infobox_company`

occur verbatim in texts. We tackle this problem with the following ideas (see [2] for details):

- *Article Paragraph Filtering:* Many Wikipedia articles are rather long and contain much information that is irrelevant for the Infobox Population Problem. We conducted an occurrence analysis and concluded that the first paragraphs of an article are sufficient for extraction of many infobox attributes. To restrict the corpus size on the one hand, but also examine only useful text passages, we choose only the first few paragraphs.

- *Fuzzy Matching:* We apply a simple heuristic to discover fuzzy matches of attribute values. We distinguish between numbers and all other strings.

- *Labeling Value Parts:* All attribute values are divided into several parts according to the corresponding attribute value structure. Each part of the value structure is labeled separately. To retain the identity of the value part that is being labeled, a number is assigned to each structure part and used as the actual label.

3.3 Value Extractor Creation

With labeled training data at hand, we can now create and apply attribute-specific extractors. Extracting attribute values from Wikipedia articles is an information extraction problem [4]. We chose Conditional Random Fields (CRFs) [1] as extraction method, because they have proven a strong performance in labeling tasks [4].

CRFs learn to label tokens based on features. The features should represent key characteristics of a token in the domain under consideration. Token labels have been determined in the previous step and also represent the position of the labeled token in the attribute value structure. By using position numbers as labels, we can exploit the dependencies of value parts in article texts; e.g., if part 1 is often followed by part 3, then the CRF can recognize this dependency.

For each infobox template attribute, we determine the labels and feature values of the tokens in the training articles. These data represent the input of the CRF learner that determines attribute-specific weights for the features. Applied to an unseen article, the CRF predicts labels based on calculated feature weights. We chose CRFsuite [3] as CRF implementation.

Attribute value extractors are selected according to their extraction performance. We automatically evaluate the performance of all generated extractors. Details are presented in Sec. 4.1. Since precision is more important than recall for automatic content generation in Wikipedia, we select all attributes for which we achieve a precision of at least 0.75. For all other attributes, the extractors are discarded.

3.4 Attribute Value Extraction

The generated attribute value extractors can now be applied to unseen articles. For this task, iPopulator determines the infobox template specified in the article. Then, all attribute value extractors that have been learned for this template are applied to the article. The result of this step is a labeled article text where the labels represent identified attribute value parts according to the attribute value structure as described in Sec. 3.1.

When constructing an attribute value from labeled tokens, the learned attribute value structure is used in various ways:

- *Align value parts:* Assigned token labels represent the value part positions in the value structure. Hence, extracted value parts in the result value are aligned according to value part positions. For repetitions, the order of appearance in the article text is retained.

- *Insert structural elements:* Structural elements, such as brackets and commas, are not extracted from article texts, but pre-defined for the entire attribute.

- *Avoid meaningless values:* Optional tokens (marked with `?`) often have no meaning without related mandatory tokens; hence, attribute values must consist of at least one mandatory token. For this reason, if only optional tokens could be extracted from the article text, no attribute value is constructed.

4. EVALUATION

After clarifying evaluation measures, we present the results of our evaluation. We evaluate performance over the entire set of infoboxes. Finally, we contrast our results with those of Kylin [6] and its successor [5]. We also provide a detailed analysis of selected infobox templates [2].

Note that values extracted by iPopulator are often not entirely true or false. By dividing a value into parts, we can specify more fine-grained evaluation measures. The correctness regarding *value parts* is determined by the proportion of correct value parts. We calculated precision P, recall R, and F-measure F of the extracted value parts.

4.1 Evaluation of all Infobox Templates

In this experiment, we apply iPopulator to all infobox templates. On average, an infobox template is used by 311 articles (minimum: 1 article, maximum: 55.300 articles). For performance reasons we select for each template 50% of all articles containing a call to this template for evaluation. From each article, the first five paragraphs have been considered. Each attribute is evaluated using 3-fold cross validation. On a 64-bit Linux 2.6.18 system with 8-core CPU and 16 GB RAM, this test took about 18 hours. The goal of this experiment is to specify precisely for which infobox templates and attributes therein we want to actually apply extraction. Only promising attributes will be chosen. We calculated precision of extraction results for all created extractors.

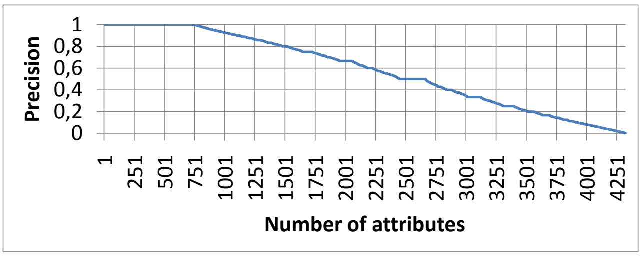

Figure 5: Precision of attribute value extractors for all attributes with $P > 0$ of all infobox templates, sorted by precision

Figure 5 shows iPopulator's attribute extraction precision. For example, we achieve $P \geq 0.8$ for 1521 attributes and $P \geq 0.9$ for 1127 attributes.

For the following, we select only those 1,727 attributes with $P \geq 0.75$. The resulting average extraction results for all infobox templates are shown in Fig. 6. The overall average measures for all selected attributes are $F = 0.73$, $P = 0.91$, and $R = 0.66$.

4.2 Comparison with Related Work

The results of iPopulator and related systems presented in Sec. 2 are difficult to compare due to different evaluation settings. Relevant differences between the evaluation methodologies of iPopulator and Kylin [6] as well as its successor [5], dubbed K2 in the following, are discussed in [2].

Despite these differences in evaluation, we offer a comparison of extraction results for the domains for which results of Kylin/K2 are known (Table 1). For K2, the authors did not state overall precision and recall numbers; thus, we eyeballed presumably optimal precision and recall values for each domain from their P/R-graphs.

The results show that iPopulator competes with Kylin and K2; in some domains, iPopulator even outperforms Kylin's and K2's results. Especially precision is iPopulator's strength, one reason being its ability to restrict extraction to promising attributes. Kylin and K2 cannot perform such restriction automatically, because their ground truth is manually extracted whereas we determine it automatically. Since iPopulator uses a similarity measure and divides attribute values into parts for labeling article texts, one could expect higher recall as well as lower precision values. However, since we use the same techniques for training as for evaluating the system, we argue that the calculated precision and recall values are not affected by these differences.

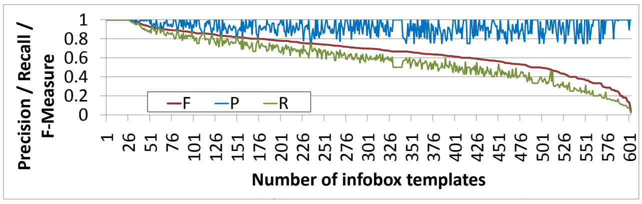

Figure 6: Performance measures for all infobox templates with $F > 0$ (for attributes with $P \geq 0.75$), sorted by F

Infobox templ.	Extraction performance					
	Kylin			iPopulator		
	P	R	#	P	R	#
Actor	0.88	0.86	50	0.93	0.81	4470
Airline	0.87	0.64	50	0.77	0.69	546
County	0.97	0.96	50	0.94	0.77	329
University	0.74	0.61	50	1.00	0.55	2368
	K2			iPopulator		
	P	R	#	P	R	#
Baseball Stadium	0.53	0.45	40	0.84	0.30	55
Irish Newspaper	0.75	0.46	20	1.00	0.42	9
Performer	0.65	0.40	44	0.95	0.25	19
Writer	0.60	0.35	40	1.00	0.23	507

Table 1: Comparison of Kylin's, K2's, and iPopulator's extraction results and numbers of evaluated articles

5. CONCLUSION AND FUTURE WORK

By automatically extracting infobox attribute values, iPopulator supports readers, authors, as well as external applications that access Wikipedia content. Exploiting the structure of attribute values improves the training quality as well as the data quality of the extracted values. Resulting values are structured similarly to the majority of attribute values in the training data. Homogeneously structured attribute values help maintain high data quality and support external applications that rely on a specific structure of infobox attribute values.

6. REFERENCES

[1] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proc. of the 18th Intl. Conf. on Machine Learning*, pages 282–289, 2001.

[2] D. Lange, C. Böhm, and F. Naumann. Extracting Structured Information from Wikipedia Articles to Populate Infoboxes. Technical Report 38, Hasso Plattner Institute, Potsdam, 2010. ISBN 978-3-86956-081-6.

[3] N. Okazaki. CRFsuite: a fast implementation of Conditional Random Fields (CRFs), 2007. `http://www.chokkan.org/software/crfsuite/`.

[4] S. Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 1(3), 2008.

[5] F. Wu, R. Hoffmann, and D. S. Weld. Information Extraction from Wikipedia: Moving Down the Long tail. In *Proc. of the 14th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 731–739, 2008.

[6] F. Wu and D. S. Weld. Autonomously Semantifying Wikipedia. In *Proc. of the 16th Conf. on Information and Knowledge Management*, pages 41–50, 2007.