Extracting Structured Information from Wikipedia Articles to Populate Infoboxes

Dustin Lange, Christoph Böhm, Felix Naumann

Technische Berichte Nr. 38

des Hasso-Plattner-Instituts für
Softwaresystemtechnik
an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Softwaresystemtechnik an der Universität Potsdam | 38

Dustin Lange | Christoph Böhm | Felix Naumann

Extracting Structured Information from Wikipedia Articles to Populate Infoboxes

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Universitätsverlag Potsdam 2010
http://info.ub.uni-potsdam.de/verlag.htm

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 4623 / Fax: 3474
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Softwaresystemtechnik an der Universität Potsdam.

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
URL http://pub.ub.uni-potsdam.de/volltexte/2010/4571/
URN urn:nbn:de:kobv:517-opus-45714
http://nbn-resolving.org/urn:nbn:de:kobv:517-opus-45714

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-081-6

Extracting Structured Information from Wikipedia Articles to Populate Infoboxes

Dustin Lange Christoph Böhm Felix Naumann
firstname.lastname@hpi.uni-potsdam.de

Hasso Plattner Institute, Potsdam, Germany

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 for independently extracting 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.

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-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

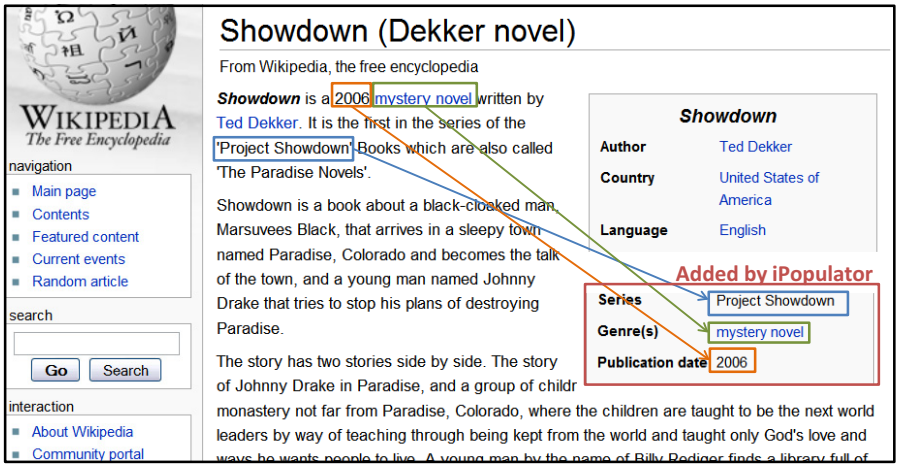


Fig. 1. Example for infobox and for infobox attribute values extracted by iPopulator

the `infobox_book` template, namely `author`, `country`, and `language`. Among others, no values for the attributes `genre` and `publication date` have been specified. There are several reasons for empty infobox attributes:

- *Unknown value:* The value of an attribute might be unknown to the author of the Wikipedia article.
- *Inappropriate attribute:* The attribute could be inappropriate for the specific subject. For example, the novel `War and Peace` is not part of a series, so it is impossible to specify any predecessors or successors.
- *Intentionally empty value:* Authors might deliberately omit attribute values, because they might be deemed uninteresting for this subject, or out of laziness.
- *Unknown attribute:* Many infoboxes are created by copying the source code from another article and replacing the attribute values. If an attribute is missing in the copied article, it will thus be missing in the newly created article.

To overcome several of these reasons for 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 on a specific set of infobox templates or on a specific domain. A system should be able to adapt to any given infobox template, i.e., the system should extract attribute values for all attributes in all infobox templates. Apart from the general difficulty of text extraction, we face the following additional challenges:

- *Large number of diverse infobox templates:* Wikipedia contains a huge set of diverse infobox templates. In the analyzed data set, more than 3,000 different infobox templates could be distinguished, although some of them are used by only a small set of articles. Altogether, the infobox templates contain more than 20,000 different attributes. The developed system should cover as many of these attributes as possible instead of focusing on a selected subset of templates.
- *Data size:* The data set itself is quite large. The used Wikipedia dump as of August 2009 contains about three million articles and has a size of 23 GB.
- *Composite infobox attribute values:* Many infobox attribute values are composed of several parts, or are lists of values, e.g.:
  - `key_people`: `Bill Gates (Chairman)`
  - `number_of_employees`: `93,000 (2009)`
  - `genre`: `Dystopian, Political novel, Social science fiction`

  Thus, iPopulator should be able to extract and construct such values accurately.
- *Diverse writing styles:* Wikipedia is a collaborative encyclopedia, i.e., most Wikipedia articles can be edited by anyone. Consequently, articles are written in different styles. The lack of regularity makes the extraction process more challenging.

Contributions With iPopulator we present a system that automatically extracts infobox attribute values from Wikipedia articles. The system automatically creates training data, which is used to learn Conditional Random Fields (CRF) models for as many infobox attributes as possible. These models can be applied to other articles to extract additional, missing attribute values. Within this overall system we make the following contributions:

- *Detection of Attribute Value Structures:* We present an algorithm that determines the internal structure of infobox attribute values. The algorithm analyzes values for each attribute to detect a structure that represents the syntactic contents of the majority of these values.
- *Exploitation of Attribute Value Structures:* We use the detected structures to improve the attribute value extraction process. The structures allow splitting values into different parts that can be extracted independently from each other. Afterwards, the structures can be used to align the extracted attribute value parts.
- *Comprehensive Evaluation:* To evaluate our ideas we tested on *all attributes* of *all infobox templates* used by at least ten articles in Wikipedia. To the best of our knowledge, this is the most comprehensive study of an infobox extraction system.

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. A detailed evaluation is given in Sec. 4. Finally, Sec. 5 concludes this paper.

2 Related Work

Related research ranges from the more general area of information extraction from the Web and Wikipedia in particular to the explicit extraction of infobox attribute values from Wikipedia.

Extraction of Relations from Web and Wikipedia KnowItAll [7] and TextRunner [2] extract knowledge in form of relations from general web documents. Both systems do not focus on the characteristics of Wikipedia. In contrast, our system focuses on creating domain-specific extractors for all attributes in all infoboxes. By specializing on Wikipedia articles, extractors can exploit the encyclopedic style of these articles. Catriple [13] analyzes category names and co-occurrences as well as the category hierarchy from Wikipedia articles to extract triples. These generated triples may contain information that is useful to fill infobox attribute values. However, the focus on categories does not overlap with our focus on article text.

Ruiz-Casado et al. [17] present a system that uses lists of term pairs as training basis and that identifies relations between these terms using Wikipedia articles. Their system uses a rule-based extraction process and needs a list of manually tagged training data items. Wang et al. [19] also use a rule-based approach, but they add selectional constraints on the extracted patterns. Herbelot and Copestake [8] propose a method to create minimal semantic representations of sentences and to extract ontological is-a relations from them. The procedure focuses on is-a relations; on the contrary, iPopulator does not restrict the set of extracted relations.

Finally, Nguyen et al. [15] extract relations by tagging entities and analyzing dependencies between them. Their system is restricted to 13 relation types, i.e., manual work is needed to add another relation type, which is not the case for iPopulator. None of these systems deal with the characteristics of Wikipedia infobox attribute extraction.

Extraction of Wikipedia Infobox Attribute Values Wu and Weld [21] 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. By applying a similarity measure to the compared infobox attribute values and 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 identified 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. [20] present an improved version of Kylin that considers shrinkage, ontologies, and web search results. These improvements result in an increase in recall of 5.8% to 50.8% while maintaining precision. In Sec. 4.4, we compare the results of Kylin and its successor with those of our work in detail.

Hoffmann et al. [9] show a further improvement of Kylin that incorporates user feedback. Their system visualizes contradictions between infobox attribute values and article text occurrences. The user is offered the possibility to correct the information. This feedback is used to improve the performance of the attribute value extractors. Finally, Adar et al. [1] present Ziggurat, a system that exploits differences between the different language versions of Wikipedia articles. For this purpose, Ziggurat matches attributes of different language versions of infobox templates by analyzing, among others, equal values, contained values, or values in the same cluster. Compared to iPopulator, both works require additional input, namely user input or versions of the same article in a different language.

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 article text 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. 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 sections, we provide details on the different steps.

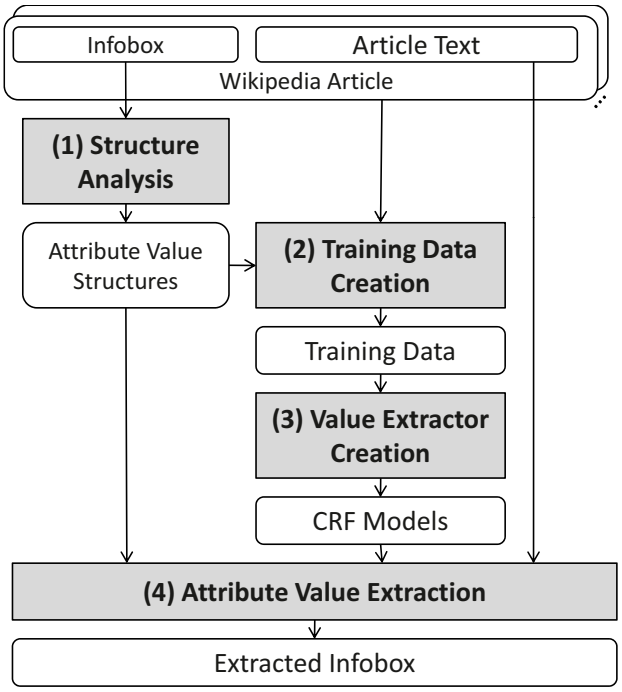


Fig. 2. iPopulator extraction process

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 ')'). Other typical examples are:

- `key_people`: `Samuel J. Palmisano (Chairman, President and CEO)`
- `revenue`: {{`loss`}} `US$ 95 billion (2009)`
- `residence`: `The White House (official)`

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 this section, we present our algorithm to discover the structure of attribute values. The goal of the algorithm is to analyze available values of an attribute and to discover 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.

The resulting structure is expressed similarly to a regular expression; hence, determining the pattern is similar to learning a regular expression from examples. There are several approaches to this task [3, 4, 12]. These approaches usually address the problem of finding a preferably small regular expression that covers every single value. Since attribute values are quite diverse, resulting expressions using such approaches would become rather complex. Additionally, these approaches do not consider the varying importance of examples. In the case of attribute values, we argue that more frequent structures should have a higher influence on the result pattern, and rare structures need not be reflected in the result pattern at all. Thus, these regular expression learning approaches are not well applicable for the problem of discovering attribute value structures.

Structure discovery algorithm We have developed a new algorithm that addresses the 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`. Table I shows some example values for this attribute. At first, the algorithm determines patterns for all values of an attribute by parsing them (Step (a)). The patterns for the examples in Table I are shown in Table II. These patterns are then counted and sorted by their frequency (Step (b)). Table III shows the most frequent patterns for `number_of_employees`. After that, the important patterns are merged into the result expression (Step (c)), starting with the most frequent ones. The result expression for the example attribute is shown in Table IV. Subsequently, we discuss the functions Parse(), IsImportantPattern(), and Merge().

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*

Fig. 3. Structure discovery algorithm

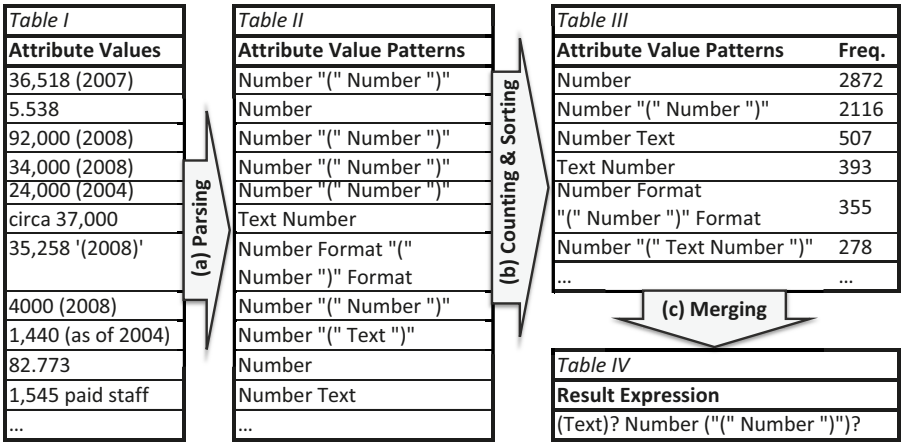


Fig. 4. Example for applying the structure discovery algorithm (Fig. 3) to values of the attribute `number_of_employees` from `infobox_company`

Function Parse() A pattern should divide an attribute value into meaningful parts. This allows us to label parts independently, as shown in Sec. 3.2. Text tokens can be distinguished from number tokens. Additionally, special Wikipedia syntax elements, such as links and template calls, can be identified. Furthermore, structural elements can be recognized and used during attribute value construction, as described in Sec. 3.4. For this purpose, brackets, commas, and other formatting symbols are identified. The Parse function is implemented using regular expressions for each of the mentioned types.

Function IsImportantPattern() For each attribute, there is a variety of different patterns that represent the format of the attribute's values. If all these patterns had to be covered by a single structure, the resulting expression would become overly complex. If only the most frequent pattern would be considered, the result structure would cover only the respective fraction of an attribute's values. For these reasons, iPopulator considers all frequent patterns; rare patterns are ignored. The function IsImportantPattern returns *True* only if the pattern covers at least 20 values and at least one percent of the values. We empirically determined these thresholds on a representatively chosen set of attributes from different infobox templates.

If there are not enough attribute values available to determine an attribute structure, we do not consider the attribute at all. In this case, creating a learner for the few and diversely specified values would be unpromising, anyway.

Function Merge() The merging algorithm is responsible for combining the current result expression (basic pattern) and another pattern (merging candidate). To merge the two patterns, we apply the following two rules.

Rule (1): Basic pattern extension

- *Condition*: The merging candidate includes and extends the basic pattern.
- *Action*: Additional value parts from the merging candidate are added to the basic pattern and marked as optional (with a question mark).
- *Example*: Merging the basic pattern (Text) with the merging candidate (Text "(" Number ")") results in (Text ("(" Number ")")?).

Rule (2): List recognition

- *Condition*: The current result expression ends with at least three repetitions of the same token sequence and at least two repetitions are marked as optional. Separator tokens (e.g., commas) are used to separate the repeated token sequences.
- *Action*: The list is summarized and repeated parts are marked (with an asterisk).
- *Example*: The attribute `starring` from `infobox_film` comprises a list of actors, such as `[[Humphrey Bogart]], [[Ingrid Bergman]], [[Paul Henreid]]`. In this case, the corresponding pattern is (Link Separator Link Separator Link). Other value patterns for this attribute end with even more sequences of Separator and Link tokens. The rule merges this list into the result expression (Link (Separator Link)*).

If none of the rules can be applied, the pattern cannot be merged into the result structure and is ignored.

3.2 Training Data Creation

Training data are a basic 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 occur verbatim in texts. We define a simple heuristic to discover fuzzy matches of attribute values. Additionally, the volume of text under consideration is quite large. To restrict the corpus size on the one hand, but also examine only useful text passages, we first filter article paragraphs.

Article Paragraph Filtering Many Wikipedia articles are rather long and contain much information that is irrelevant for the Infobox Population Problem. For example, the `infobox_book` attribute `name` refers to the book name that is usually mentioned in the very first sentence of the corresponding article. Figure 5 shows an analysis of the 15 most frequent attributes of `infobox_book`. It plots in which paragraph attribute values occur the first time (if contained at all). For example, to find out the book name, reading only the first paragraph is sufficient for 92% of all book articles. This tendency holds for most attributes in most templates, i.e., most graphs have a relatively high slope for the first paragraphs, but level out for following paragraphs.

In conclusion, examining more than a few initial paragraphs does not significantly add attribute value occurrences. Moreover, experiments have shown

that examining more paragraphs may even cause worse extraction results due to the consideration of much useless information, which makes the extraction model more noisy and thus error-prone. For several different infobox templates, we evaluated different amounts (1-10 paragraphs) and determined a length of five paragraphs as an optimal compromise between precision and recall. Figure 5 underpins this discussion.

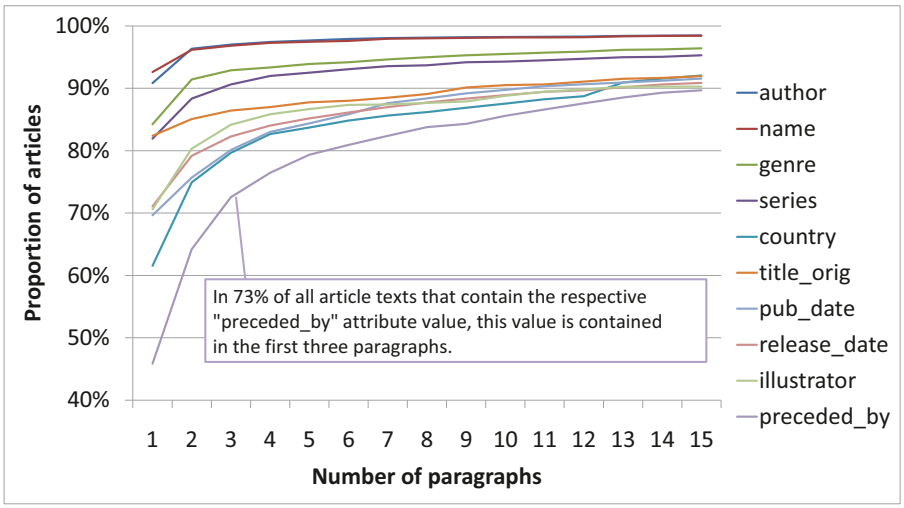


Fig. 5. Proportion of articles containing the respective attribute value in the specified number of paragraphs (15 most frequent attributes in `infobox_book`)

Labeling with Similarity Measure To create training data for CRF learning, we now label occurrences of infobox attribute values in article texts. Often times, an attribute value and its occurrence in the text do not exactly match. For example, the value of the attribute `company_name` might be `Fantastic Holdings Ltd`; but the article text mentions `Fantastic Holdings Limited`. To handle such differences, we define a similarity measure that determines whether an article text token is a fuzzy match of an attribute value or not. The measure combines an approximation measure for numeric values and edit distance for all other strings. Functions 1 and 2 define the similarity for numbers and strings. If only one of two compared tokens is a number, they are evaluated as dissimilar. We chose the Levenshtein edit distance [11] for the *ed (edit distance)* function. Although there are distance measures that are better suited for specific string matching tasks [6], Levenshtein distance with equal weights for insertion, deletion, and update of characters is an adequate choice for the generic setting of matching all kinds of strings in Wikipedia. The similarity thresholds were determined empirically using 20 representatively chosen attributes in 5 frequently

used infobox templates.

$$s_{num}(s1, s2) := \begin{cases} True & |s1 - s2| \leq \frac{1}{1000} s1 \\ False & \text{otherwise} \end{cases} \quad (1)$$

$$s_{other}(s1, s2) := \begin{cases} True & \text{ed}(s1, s2) \leq \frac{1}{4} \text{ length}(s1) \\ False & \text{otherwise} \end{cases} \quad (2)$$

In our evaluation, we compare the occurrence rates for exact matching and fuzzy matching and observe an improvement for all considered infobox templates: More occurrences can be found by applying the similarity measure. Using similar matching, we achieve an average occurrence rate of 26.0%, which is an increase of 23%.

Labeling Value Parts Besides fuzzy matching, we leverage attribute value structures to further improve training data creation. The structure that is determined for each attribute in each infobox is useful to improve training data creation. All attribute values are divided into several parts according to the corresponding attribute value structure. Each part of the value structure is labeled separately. For example, the value of `number_of_employees` in `infobox_company` can be `54,400 (2008)`. On the other hand, the corresponding article might state `In 2008, the company had 54,400 employees`. Although both numbers occur in the sentence, no token would be found if only entire values were labeled. Labeling value parts allows us to individually label the parts `54,400` and `2008`.

A statistical analysis shows that searching value parts independently significantly increases the probability to find an attribute value occurrence. Figure 6 shows a comparison between occurrence rates of complete attribute values and value parts. Here we show the average of value occurrence rates for each infobox template. On average, searching for value parts increases the rate of found occurrences from 26.0% to 33.9%; an improvement of 30.5%.

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. Only the value part occurrences in the sentence that contains the most individual parts are labeled. A sentence that contains only such parts of an attribute value that have been marked as optional is disregarded. Thus, for instance spurious occurrences of year values are ignored.

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 [18]. To tackle this, we employ learning-based methods rather than hand-coded techniques, since our goal is to build a system that extracts attribute values for *all attributes of all infoboxes*. Further, Wikipedia allows anyone to add new content, which causes articles to show many

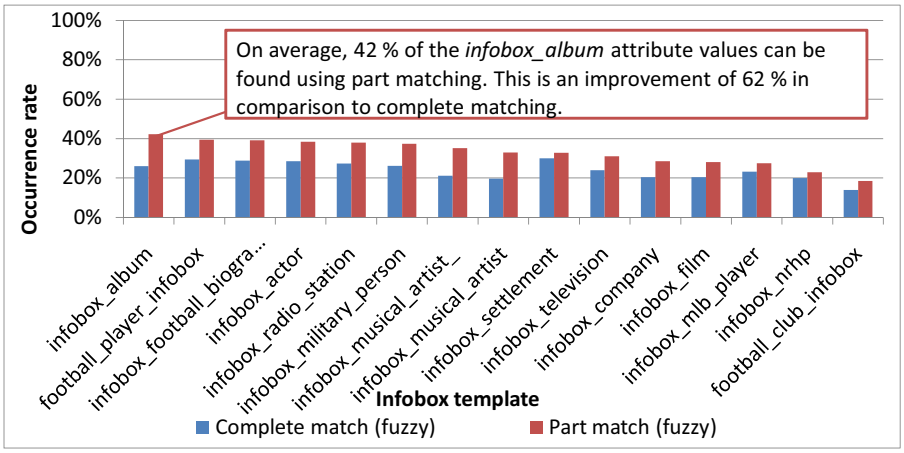


Fig. 6. Comparison of complete attribute value occurrences and average attribute value part occurrences, i.e., average fraction of token parts that occur in the article text

different writing styles. Due to a lack of common sentence structures, we prefer statistical methods over extraction rules. Specifically, we chose Conditional Random Fields (CRF) [10] as extraction method, because they have proven a strong performance in labeling tasks [18].

CRFs learn to label tokens based on features. The features should represent key characteristics of a token in the analyzed domain. Our features are shown in Table 1. Token labels have been determined in the previous step and 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. As CRF implementation, we used CRFsuite [16] with L-BFGS [5] as feature weight estimation method.

Attribute value extractors are selected according to their extraction performance. We automatically evaluate attribute extraction performance of all generated extractors. Details are presented in Sec. 4.3. 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.

Description
Length
Small token (length < 10)
Within long paragraph (#sentences > 2)
Relative token positions
Position in article: in 1st, 2nd, 3rd third
Position in paragraph: in 1st, 2nd, 3rd third
Position in sentence: in 1st, 2nd, 3rd third

(a) Features for the currently analyzed token only.

Description	Example
General	
Value of the token	2010
Type	Number, String
Part-of-speech tag	NN
Enclosed by formatting symbols?	[”] Hi [”]
Enclosed by the formatting symbols "'?	[”’] Title [”’]
Structure	
Two-digit number	17
Four-digit number	2010
Number	1511
Formatted numbers (thousand sign)	1,121
Formatted numbers (thous., mill./bill.)	1 mill.
Alphanum. char. (start: letter)	Company
Alphanum. char. (start: letter, end: dot)	end.
Check for Occurrence	
Token km/mi/miles	km
Token contains “http”	http://faz.de
Token contains “?”	Who?
Token contains “.”	him.

(b) Features for all tokens inside a window of five tokens before and after the currently analyzed token.

Table 1. CRF features determined for each token

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. 3.2.

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. In the case of opening and closing brackets, the actual bracket symbol is encoded in the value structure part. In the case of Separator and Format tokens, the most frequent value for this token is determined by analyzing the training data for the considered attribute.
- *Avoid meaningless values:* Optional tokens 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.

For instance, the text "`IBM's key people are Sam Palmisano, who serves as CEO, and Mark Loughridge as SVP.`" is used to construct `Sam Palmisano (CEO), Mark Loughridge (SVP)` for the `key_people` attribute.

4 Evaluation

After clarifying evaluation measures, we present the results of our evaluation. First, we analyze selected infobox templates in detail, then we evaluate performance over the entire set of infoboxes. Finally, we contrast our results with those of Kylin [21] and its successor [20].

4.1 Evaluation Measures

For the evaluation of extracted attribute values, we distinguish the correctness of a complete value and the correctness of value parts. In both cases, the similarity measure defined in Sec. 3.2 is used.

To evaluate *complete values*, we compare the values extracted by iPopulator with the expected attribute values in existing infoboxes. $F_{complete}$ is the harmonic mean (F-measure) of precision ($P_{complete}$) and recall ($R_{complete}$) regarding complete values.

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. The usage of value part numbers as labels allows the alignment of extracted and expected value parts. The aligned parts are classified as

- Correct (C), if expected and extracted value parts match,
- Substituted (S), if expected and extracted value parts do not match,
- Deleted (D), if no value has been extracted for an expected value even though the expected value appears in the article text, or
- Inserted (I), if an additional value part has been extracted.

This notation is based on the work by Makhoul et al. [14]. The performance measures are defined as follows: F_{part} is the harmonic mean of precision P_{part} and recall R_{part} with

$$P_{part} = \frac{|C|}{|C| + |S|} \qquad R_{part} = \frac{|C|}{|C| + |S| + |D|}$$

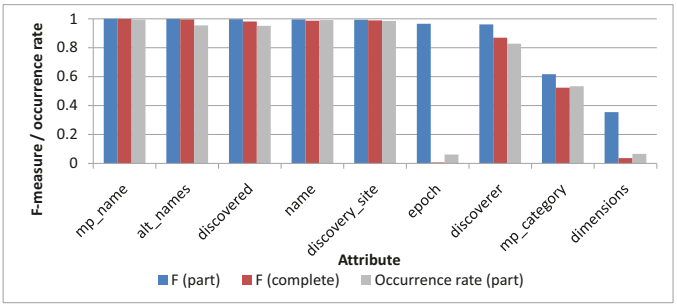
Insertions are not considered incorrect, since it is desired to extract additional value parts. But because their correctness cannot be evaluated automatically, insertions are ignored in this measure.

To further illustrate the difference between value parts and complete values, we use the following example: An infobox contains a single value that consists of three parts $p_1p_2p_3$ and the article text contains all three desired value parts. If iPopulator extracts $p_1p_2p_3$, then $P_{complete} = P_{part} = 1$. If iPopulator extracts $p_1p_2p_4$ with $p_3 \neq p_4$, then $P_{complete} = 0$, since the complete values are too different, but $P_{part} = \frac{2}{3}$, since only one of the three value parts is not correct.

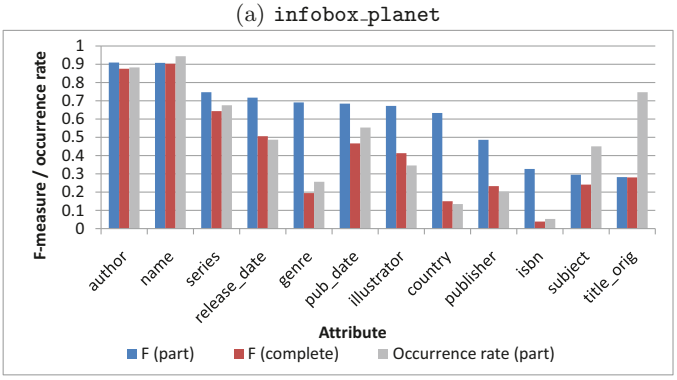
4.2 Experiment 1: Selected Infobox Templates

Methodology Four different infobox templates have been chosen for a detailed discussion: `infobox_actor`, `infobox_book`, `infobox_company`, and `infobox_planet`. These templates are among the 50 most frequently used templates, but they differ in their domains. For each infobox template, 50% of all articles having an infobox of this type were randomly selected and used for training and evaluation by means of 3-fold cross-validation. Table 2 summarizes the overall results and Fig. 7 shows results for different attributes within the analyzed templates. Note that this experiment covers *all* respective template attributes to allow a detailed analysis. Therefore, the results include attributes for which it is known that extraction yields poor results. For the actual use, iPopulator excludes such poor extractors.

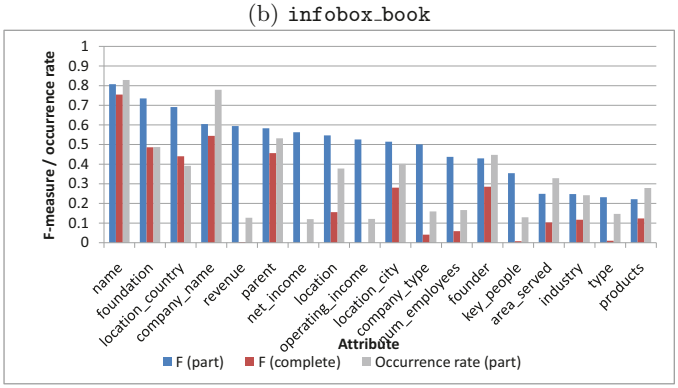
Discussion Figure 7 allows an analysis of differences between F_{part} and $F_{complete}$. F_{part} shows the extraction quality of the token value parts, while $F_{complete}$ indicates the quality of the attribute value construction from value parts. On average,



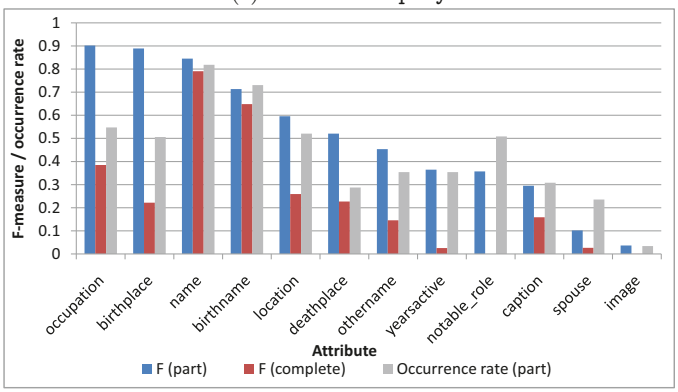
(a) `infobox_planet`



(b) `infobox_book`



(c) `infobox_company`



(d) `infobox_actor`

Fig. 7. Extraction results and attribute value part occurrence rates (as introduced in Sec. 3.2) for attributes in the analyzed infobox templates. Attributes are sorted by F_{part}.

Infobox template	#articles	Avg. F_{part}	Avg. $F_{complete}$
infobox_actor	12168	0.59	0.38
infobox_book	6256	0.62	0.41
infobox_company	7589	0.51	0.28
infobox_planet	5470	0.98	0.82

Table 2. Average extraction results per infobox template, calculated by weighting results of all infobox attributes by attribute frequency

$F_{complete}$ is about one third lower than F_{part}. This means that in two of three cases, complete attribute values could be constructed correctly from value parts using the attribute value construction algorithm (Sec. 3.4). In the other cases, at least one meaningful part of the value could be extracted.

For some attributes, such as `occupation` and `location` in `infobox_actor`, or `foundation` and `revenue` in `infobox_company`, F_{part} is considerably higher than $F_{complete}$. There are several reasons for that. First, if some parts of an attribute value could not be extracted, it is impossible to construct a complete attribute value. For example, many values for `revenue` in `infobox_company` contain a small icon indicating whether the revenue is a profit or loss. This icon is implemented as a template call, such as {{`profit`}} or {{`loss`}}, and thus never contained in article texts. This always yields a wrong complete value, even if other correct parts for the value were extracted correctly.

Another reason for incorrect complete results can be the order of value parts. Although the attribute value construction algorithm presented in Sec. 3.1 aligns the token value parts according to the extracted attribute value pattern, there are cases in which no alignment is possible. For example, the value for `occupation` in the infobox of the article on Dolores Hart is `[[Actress]], [[nun]]`. The extracted value is `[[nun]], [[actress]]`. Since the value structure for this attribute contains a list, the extracted attribute value parts are sorted by their order of appearance in the article text, which contradicts the order given in the article's infobox.

Moreover, the overall results in Table 2 show considerable differences between `infobox_planet` and all other infobox templates. A manual inspection of planet articles revealed that many are rather short and share a similar structure. This similar structure allows the attribute value extractors to easily identify the correct attribute values, if they are contained in the article at all. Presumably, only a relatively small community of authors regularly edits planet articles, while much more and more diverse authors edit articles about books or companies. Hence, generating extractors for these domains is more challenging.

An important factor for the extraction quality is the quality of the training data. For example, labeling a year is quite difficult, because numbers often appear in several sentences and different contexts. For selected attributes (five attributes per infobox template), we manually analyzed the quality by judging the context of the labeled attribute value occurrence. When comparing training data and

extraction quality, a correlation can be seen for some attributes. For example, for all `name` attributes, only few errors were made during training data construction, and good extraction results can be achieved (see Fig. 7 (a)–(d)). In contrast, for the attribute `isbn` in `infobox_book`, all labeled occurrences are in a wrong context; the extraction result for this attribute is unsurprisingly poor ($F_{part} = 0.32$). These findings indicate that further research on improving training data quality may also help improving extraction quality.

To further analyze reasons for good or poor extraction results, Fig. 7 also shows occurrence rates of the attribute values. For several attributes, such as `name` and `author` in `infobox_book`, or `name` and `birthname` in `infobox_actor`, there are high occurrence rates as well as good extraction results. For other attributes, e.g., `followed_by` and `image_caption` in `infobox_book`, or `image` and `spouse` in `infobox_actor`, values occur only rarely in article texts, and extraction results are rather poor. This indicates a relationship between occurrence rate and extraction performance. Note that for low occurrence rates, $F_{complete}$ is often rather low, while considerably better results could be achieved for F_{part}. This indicates that, even for rarely occurring attribute values, we can extract at least parts of a value.

4.3 Experiment 2: All Infobox Templates

Methodology 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 each template, 50% of all articles containing a call to this template have been selected for evaluation[1]. 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. To characterize this freedom, we calculated precision of extraction results for all created extractors. The results are shown in Fig. 8.

Discussion Figure 8 shows for how many attributes iPopulator achieves good extraction results. For example, we achieve precision $P_{part} \geq 0.8$ for 1521 attributes and $P_{part} \geq 0.9$ for 1127 attributes.

We also examined the sensitivity of our approach to training data size. In Fig. 9, each point represents (at least) one attribute value extractor. The graph shows the number of considered training articles as well as the extraction performance. We can see that there is no direct correlation between training data size and extraction performance: there are extractors created with only few training articles achieving good extraction results as well as extractors with much training data and poor results, and vice versa. iPopulator's performance does not seem

[1] For performance reasons, we chose a subset of all available articles.

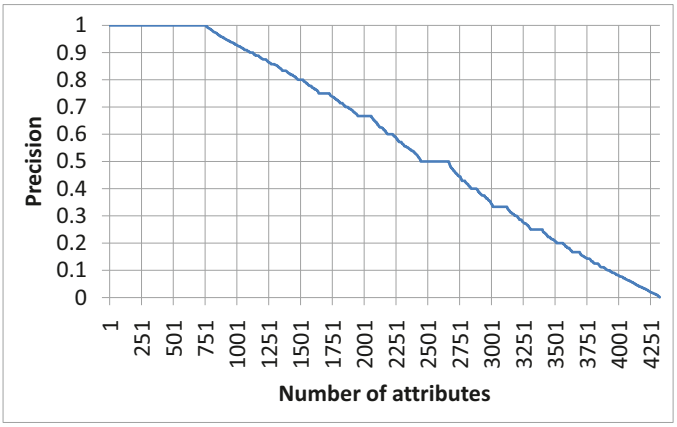


Fig. 8. Precision of attribute value extractors for all attributes with $P_{part} > 0$ of all infobox templates, sorted by precision

to be sensitive on training data size. Other attribute properties, as examined in the previous experiment, are more relevant for extraction performance.

For the following, we select only those 1,727 attributes with $P_{part} \geq 0.75$. The resulting average extraction results for all infobox templates are shown in Fig. 10. The overall average measures for all selected attributes are $F_{part} = 0.73$, $P_{part} = 0.91$, and $R_{part} = 0.66$.

4.4 Comparison with Related Work

The results of iPopulator and related systems presented in Sec. 2 are difficult to compare. Relevant differences between the evaluation methodologies of iPopulator and Kylin [21] as well as its successor [20], dubbed K2 in the following, are shown in Table 3.

Despite these differences in evaluation, we offer a comparison of extraction results for all domains for which results of Kylin/K2 are known (Table 4). 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.

Kylin/K2	**iPopulator**
Evaluation of 8 concepts	Evaluation of *all* infobox templates (≈ 800 concepts)
Tests on 20-50 randomly selected articles for each concept	Tests on 50% of all available articles for each concept (average for all concepts: 311 articles, average for Kylin/K2's selected concepts: 1471 articles), randomly selected
All attributes that appear in at least 15% of all infoboxes for the concept	All attributes that appear in at least 5% of all infoboxes for the concept and for which precision >= 0.75
Ground truth = manually extracted attribute values	Ground truth = existing infobox attribute values

Table 3. Differences between iPopulator and Kylin/K2 relevant for evaluation

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 4. Comparison of Kylin's, K2's, and iPopulator's extraction results and numbers of evaluated articles

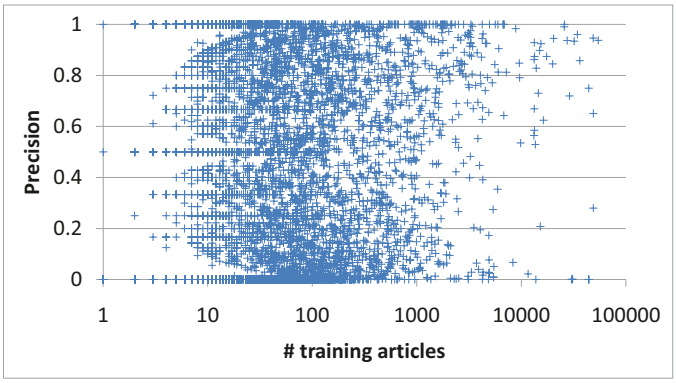


Fig. 9. Number of selected training articles and achieved extraction precision per attribute value extractor

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.

We leave for future work the application of our techniques to improve the structure of existing attribute values and to include external sources for further improvement of extraction results.

References

1. E. Adar, M. Skinner, and D. S. Weld. Information Arbitrage across Multi-lingual Wikipedia. In *Proc. of the 2nd Intl. Conf. on Web Search and Data Mining*, pages 94–103, 2009.
2. M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni. Open Information Extraction from the Web. In *Proc. of the 20th Intl. Joint Conf. on Artificial Intelligence*, pages 2670–2676, 2007.
3. G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. In *Proc. of the 17th Intl. Conf. on World Wide Web*, pages 825–834, 2008.
4. A. Brazma. Efficient Algorithm for Learning Simple Regular Expressions from Noisy Examples. In *Proc. of the 4th Intl. Workshop on Analogical and Inductive Inference*, pages 260–271, 1994.
5. R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM J. Sci. Comput.*, 16(5):1190–1208, 1995.

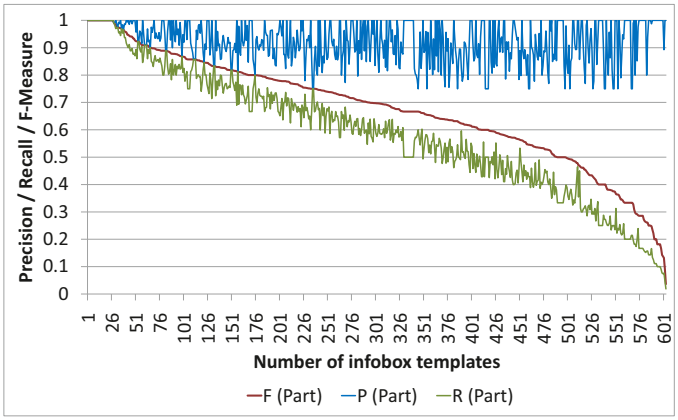


Fig. 10. Performance measures for all infobox templates with $F_{part} > 0$, sorted by F_{part}

6. W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of IJCAI-03 Workshop on Information Integration*, pages 73–78, August 2003.
7. O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised Named-entity Extraction from the Web: an Experimental Study. *Artificial Intelligence*, 165(1):91–134, 2005.
8. A. Herbelot and A. Copestake. Acquiring Ontological Relationships from Wikipedia Using RMRS. In *Proc. of the ISWC 2006 Workshop on Web Content Mining with Human Language Technologies*, 2006.
9. R. Hoffmann, S. Amershi, K. Patel, F. Wu, J. Fogarty, and D. S. Weld. Amplifying Community Content Creation with Mixed-Initiative Information Extraction. In *Proc. of the 27th Intl. Conf. on Human Factors in Computing Systems*, pages 1849–1858, 2009.
10. 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.
11. V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10, February 1966.
12. Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. V. Jagadish. Regular Expression Learning for Information Extraction. In *Proc. of the 2008 Conf. on Empirical Methods in NLP*, pages 21–30, 2008.
13. Q. Liu, K. Xu, L. Zhang, H. Wang, Y. Yu, and Y. Pan. Catriple: Extracting Triples from Wikipedia Categories. In *Proc. of the 3rd Asian Semantic Web Conf.*, pages 330–344, 2008.
14. J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel. Performance Measures For Information Extraction. In *Proc. of DARPA Broadcast News Workshop*, pages 249–252, 1999.
15. D. P. T. Nguyen, Y. Matsuo, and M. Ishizuka. Exploiting Syntactic and Semantic Information for Relation Extraction from Wikipedia. In *Proc. of the IJCAI 2007 Workshop on Text-Mining & Link-Analysis*, 2007.

16. N. Okazaki. CRFsuite: a fast implementation of Conditional Random Fields (CRFs), 2007. `http://www.chokkan.org/software/crfsuite/`.
17. M. Ruiz-Casado, E. Alfonseca, and P. Castells. Automatising the Learning of Lexical Patterns: An Application to the Enrichment of WordNet by Extracting Semantic Relationships from Wikipedia. *Data & Knowledge Engineering*, 61(3):484–499, 2007.
18. S. Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 1(3), 2008.
19. G. Wang, H. Zhang, H. Wang, and Y. Yu. Enhancing Relation Extraction by Eliciting Selectional Constraint Features from Wikipedia. In *Proc. of the 12th Intl. Conf. on Applications of Natural Language to Information Systems*, pages 329–340, 2007.
20. 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.
21. F. Wu and D. S. Weld. Autonomously Semantifying Wikipedia. In *Proc. of the 16th Conf. on Information and Knowledge Management*, pages 41–50, 2007.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
37	978-3-86956-078-6	**Toward Bridging the Gap Between Formal Semantics and Implementation of Triple Graph Grammars**	Holger Giese, Stephan Hildebrandt, Leen Lambers
36	978-3-86956-065-6	**Pattern Matching for an Object-oriented and Dynamically Typed Programming Language**	Felix Geller, Robert Hirschfeld, Gilad Bracha
35	978-3-86956-054-0	**Business Process Model Abstraction : Theory and Practice**	Sergey Smirnov, Hajo A. Reijers, Thijs Nugteren, Mathias Weske
34	978-3-86956-048-9	**Efficient and exact computation of inclusion dependencies for data integration**	Jana Bauckmann, Ulf Leser, Felix Naumann
33	978-3-86956-043-4	**Proceedings of the 9th Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS '10)**	Hrsg. von Bram Adams, Michael Haupt, Daniel Lohmann
32	978-3-86956-037-3	**STG Decomposition: Internal Communication for SI Implementability**	Dominic Wist, Mark Schaefer, Walter Vogler, Ralf Wollowski
31	978-3-86956-036-6	**Proceedings of the 4th Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
30	978-3-86956-009-0	**Action Patterns in Business Process Models**	Sergey Smirnov, Matthias Weidlich, Jan Mendling, Mathias Weske
29	978-3-940793-91-1	**Correct Dynamic Service-Oriented Architectures: Modeling and Compositional Verification with Dynamic Collaborations**	Basil Becker, Holger Giese, Stefan Neumann
28	978-3-940793-84-3	**Efficient Model Synchronization of Large-Scale Models**	Holger Giese, Stephan Hildebrandt
27	978-3-940793-81-2	**Proceedings of the 3rd Ph.D. Retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
26	978-3-940793-65-2	**The Triconnected Abstraction of Process Models**	Artem Polyvyanyy, Sergey Smirnov, Mathias Weske
25	978-3-940793-46-1	**Space and Time Scalability of Duplicate Detection in Graph Data**	Melanie Herschel, Felix Naumann
24	978-3-940793-45-4	**Erster Deutscher IPv6 Gipfel**	Christoph Meinel, Harald Sack, Justus Bross
23	978-3-940793-42-3	**Proceedings of the 2nd. Ph.D. retreat of the HPI Research School on Service-oriented Systems Engineering**	Hrsg. von den Professoren des HPI
22	978-3-940793-29-4	**Reducing the Complexity of Large EPCs**	Artem Polyvyanyy, Sergy Smirnov, Mathias Weske
21	978-3-940793-17-1	**"Proceedings of the 2nd International Workshop on e-learning and Virtual and Remote Laboratories"**	Bernhard Rabe, Andreas Rasche

ISBN 978-3-86956-081-6
ISSN 1613-5652