Improving Efficiency and Accuracy in Multilingual Entity Extraction

Joachim Daiber  
Computational Linguistics  
Rijksuniversiteit Groningen  
Groningen, The Netherlands  
daiber.joachim@gmail.com

Max Jakob  
Neofonie GmbH  
Robert-Koch-Platz 4  
10115 Berlin, Germany  
max.jakob@neofonie.de

Chris Hokamp  
Lang. and Inf. Technologies  
University of North Texas  
Denton, TX, USA  
chris.hokamp@gmail.com

Pablo N. Mendes  
Kno.e.sis Center, CSE Dept.  
Wright State University  
Dayton, OH, USA  
pablo@knoesis.org

ABSTRACT
There has recently been an increased interest in named entity recognition and disambiguation systems at major conferences such as WWW, SIGIR, ACL, KDD, etc. However, most work has focused on algorithms and evaluations, leaving little space for implementation details. In this paper, we discuss some implementation and data processing challenges we encountered while developing a new multilingual version of DBpedia Spotlight that is faster, more accurate and easier to configure. We compare our solution to the previous system, considering time performance, space requirements and accuracy in the context of the Dutch and English languages. Additionally, we report results for 7 additional languages among the largest Wikipedias. Finally, we present challenges and experiences to foment the discussion with other developers interested in recognition and disambiguation of entities in natural language text.

Categories and Subject Descriptors
I.2.7 [Artificial Intelligence]: Natural Language Processing—language models, text analysis; H.3.1 [Information Storage and Retrieval]: Content Analysis—linguistic processing

General Terms
Algorithms, Performance

Keywords
Named Entity Recognition, Entity Linking, Information Extraction

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
I-SEMANTICS 2013, 9th Int. Conf. on Semantic Systems, Sept. 4-6, 2013, Graz, Austria
Copyright 2012 ACM 978-1-4503-1972-0 ...$10.00.

1. INTRODUCTION
DBpedia Spotlight [3] is an open source project developing a system for automatic annotation of DBpedia entities in natural language text. It provides programmable interfaces for phrase spotting (recognition of phrases to be annotated) and disambiguation (entity linking) as well as various output formats (XML, JSON, RDF, etc.) in a REST-based web service. The standard disambiguation algorithm is based upon cosine similarities and a modification of TF-IDF weights (using Apache Lucene[1]). The main phrase spotting algorithm is exact string matching, which uses LingPipe’s[2] Aho-Corasick implementation.

The project has focused initially on the English language. However, since DBpedia Spotlight’s models are learned from Wikipedia, it should be possible to adapt the system to any other language that has a Wikipedia edition. In addition, although fairly intensively used by researchers[4] and others[5], the current implementation can be improved in certain aspects.

In this paper, we describe how we enhanced the performance and accuracy of our entity recognition and disambiguation components and discuss challenges encountered while adapting DBpedia Spotlight to work with other languages. For this experiment, we focused on the Dutch language. As a result, we present a new version of the system, which is faster, more accurate and improves the ease of internationalization. Demonstrations of internationalized versions are provided in English and Dutch and models for 7 additional languages are made available via the supporting material for this submission.

2. IMPLEMENTATION

2.1 Phrase Spotting
Phrase spotting is the task of finding phrases in a text that later should be linked to DBpedia entities. The stan-

[3]As of the time of writing, Google Scholar reports 108 citations to the main DBpedia Spotlight article.
http://git.io/qB3m6g
dard phrase spotting algorithm used by DBpedia Spotlight is very light on its NLP demands, only requiring one language-dependent step: tokenization. Recognition of spot candidates is then performed by substring matching using a kind of prefix tree and the Aho-Corasick algorithm.

Our goal is to have the best possible phrase spotting quality and time performance with little memory overhead. Previous work [2] showed increased precision when using NLP-informed phrase spotting methods. Therefore, we developed a Spotter that proceeds in two steps.

First step: In the first step, candidates for possible annotations are generated. There are two implementations of this step. In the language-independent implementation, the candidates are generated by traversing a finite state automaton encoding all possible sequences of tokens that form known spot candidates.

In the language-dependent implementation, candidates are generated using three methods: 1. identifying all sequences of capitalized tokens, 2. identifying all noun phrases, prepositional phrases and multi word units, 3. identifying all named entities. Methods 2 and 3 are performed using Apache OpenNLP [5] models for phrase chunking and Named Entity Recognition.

For English and German, all required OpenNLP models are readily available. For Dutch, we created a chunker model from the Lassy Small corpus, a corpus of Dutch language texts with automatic syntactic annotations which were manually corrected [6]. The resulting phrase chunker detects Noun Phrase chunks (NP), Prepositional Phrase chunks (PP) and Multi Word Units (MWUs). We use this model in addition to the publicly available Dutch Named Entity Recognition models.

Second step: As our goal is to achieve the best possible precision, the second step selects the best candidates from the set of phrases generated in step 1. First, overlaps in the candidates are resolved based on a score and a preference-based choice for the method that generated them. Second, all candidates that fall below a specified score threshold are removed. We compute the score for each spot candidate as a linear combination of features, since this gives us flexibility in treating certain kinds of phrases differently. One of our features is the annotation probability P(\text{annotation}\mid s). Given a set of mentions of article links e with anchor text s (spot), we estimate the annotation probability as: 
\[ P(\text{annotation}\mid s) = \sum_c \frac{\text{count}(e, s)}{\text{count}(s)} \]

We identified several cases of phrases where this annotation probability is consistently lower than the general case. Acronyms are one example of this phenomenon (e.g. ‘AIG’, ‘IBM’). Hence, we add binary features that are triggered for these kinds of cases to produce the overall score. We determine the weights for the components of this score from held-out data via linear regression. This method provides the additional advantage of automatically estimating the optimal cut-off threshold from the data.

2.2 Disambiguation

Disambiguation in our build is performed using the generative probabilistic model from [6]. The score for an entity e given the phrase s and context c is calculated as a combination of \( P(e) \), \( P(s|e) \) and \( P(e|c) \). The combination can be either a linear regression mixture or, in accordance with the generative model, the product of the individual values.

Given the Wikipedia data set \( M \) consisting of article links with their anchor texts and textual context, we estimate the required probability distributions from the raw counts using the maximum likelihood approach presented in [6]. \( P_{\text{LM}}(t) \) is the smoothed unigram language model estimated over all tokens in the data set and weighed by the parameter \( \lambda \) whose value we adopted from [6].

\[
\begin{align*}
P(e) &= \frac{\text{count}(e)}{|M|} \\
\quad \end{align*}
\]
\[
\begin{align*}
P(s|e) &= \frac{\text{count}(e, s)}{\text{count}(e)} \\
\quad \end{align*}
\]
\[
\begin{align*}
P(e|c) &= \frac{\text{count}(e, s)}{\text{count}(s)} \\
\quad \end{align*}
\]
\[
\begin{align*}
P_{\text{ML}}(t) &= \lambda P_{\text{LM}}(t) + (1 - \lambda) P_{\text{ML}}(t) \\
\quad \end{align*}
\]
\[
\begin{align*}
P_{\text{ML}}(t) &= \frac{\text{count}(t)}{\sum \text{count}(t)} \\
\quad \end{align*}
\]

In this generative model, probabilities are negligibly small and are represented in logarithmic space to avoid underflows. To produce a more useful score for each entity, we normalize the score via the softmax function and obtain a final disambiguation score between 0.0 and 1.0.

For each phrase and its surrounding context tokens, we further generate a NIL entity, which represents the hypothesis that the context and phrase were not generated by any known entity. We use the formulas for \( P(\text{NIL}) \), \( P(s|\text{NIL}) \) and \( P(e|\text{NIL}) \) from [6] to calculate the score for the NIL entity. All entity candidates with a lower score than the NIL entity are removed.

\[
\begin{align*}
P(\text{NIL}) &= \frac{1}{|M|} \\
P(s|\text{NIL}) &= \prod_{t \in C} P_{\text{LM}}(t) \\
P(e|\text{NIL}) &= \prod_{t \in C} P_{\text{LM}}(t) \\
\quad \end{align*}
\]

As above, \( P_{\text{LM}} \) is the smoothed general language model probability of a token that we estimate over all tokens imported to the system as context of an entity mention.

2.3 Indexing

The indexing process of our model consists of two phases: First, we collect the raw counts necessary for our models from Wikipedia. Some language-dependent pre-processing is performed, including stemming and the removal of stopwords. We use PigNLProc [7] a collection of Pig Latin scripts and utilities focused on Wikipedia and DBpedia that we adapted for this work. Apache Pig is part of the Hadoop ecosystem, providing a high-level abstraction of MapReduce using an SQL-like syntax. In the second step, the raw counts are serialized into efficient data structures that the system can de-serialize at runtime. This step can be performed on a commodity PC and takes a few minutes for Dutch and 1-2 hours for the English version.

For the annotation probability, we must find \( \text{count}(s) \), the total number of string occurrences of a phrase in the corpus. Since performing a string search for each possible phrase in the entire corpus is not feasible, we collect all n-grams (where

[2] In the order PER> ORG> LOC> MISC> NP> MWU> PP> FSA lookup>
Capitalized Sequence.

2.4 Data Storage, Models and Configuration

When selecting our preferred data storage implementation, we considered the following criteria: compatibility with the Apache 2.0 license, optimization for read operations, type of data serialization, ease of integration and scalability. While some full-fledged DBMS, such as Apache Cassandra, met most of our criteria, we decided to use an in-memory model efficiently serialized using Kryo, since this allows maximum retrieval performance, high flexibility, smaller external dependencies and deliberate optimizations based on our knowledge of the data. Optionally, JDBM3 can be used for disk-based access on low memory systems. We share our performance evaluations of suitable data storages in the supporting material for this submission. To reduce the complexity of the configuration, we introduced a self-contained model directory structure that is produced by the indexing module and can be run with no further configuration using the DBpedia Spotlight server.

3. EVALUATION

Our aim was to produce a system providing easier and faster indexing, faster and more accurate runtime performance and simple internationalization. The evaluation was carried out on our build as well as the current build of DBpedia Spotlight for Dutch and English. We evaluated indexing and runtime performance, phrase spotting and disambiguation. Additional to English and Dutch, where we had previous systems to compare against, we provide these values for 7 additional languages.

3.1 Performance: Runtime and Footprint

Annotation time. To measure time performance (Table 1), we annotated a small corpus of around 500 randomly selected articles from the Reuters news corpus using both systems with their default settings. In Table 1, we report the disk and memory footprint for our build and for two versions of the current build: in Current (1), the candidate index is kept in memory and the disambiguation index is on disk while in Current (2) both indexes are loaded to memory.

<table>
<thead>
<tr>
<th>Language</th>
<th>Model</th>
<th>Articles</th>
<th>Disk</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>Our build</td>
<td>1.20s</td>
<td>601.39s</td>
<td>1.9GB</td>
</tr>
<tr>
<td></td>
<td>Current</td>
<td>9.52s</td>
<td>4758.31s</td>
<td>2.1GB</td>
</tr>
<tr>
<td>English</td>
<td>Our build</td>
<td>1.28s</td>
<td>640.2s</td>
<td>5.2GB</td>
</tr>
<tr>
<td></td>
<td>Current</td>
<td>5.72s</td>
<td>2803.22s</td>
<td>11.7GB</td>
</tr>
</tbody>
</table>

Table 1: Performance evaluations

Disk and Memory Footprint. In Table 2, we report the disk and memory footprint for our build and for two versions of the current build: in Current (1), the candidate index is kept in memory and the disambiguation index is on disk while in Current (2) both indexes are loaded to memory.

<table>
<thead>
<tr>
<th>Language</th>
<th>Model</th>
<th>Articles</th>
<th>Disk</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>Our build</td>
<td>480MB</td>
<td>1.9GB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Current (1)</td>
<td>2.1GB</td>
<td>2.4GB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Current (2)</td>
<td>2.1GB</td>
<td>14.6GB</td>
<td></td>
</tr>
<tr>
<td>English</td>
<td>Our build</td>
<td>5.2GB</td>
<td>11.7GB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Current (1)</td>
<td>18.2GB</td>
<td>5.9GB</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Space requirements

3.2 Phrase Spotting

We evaluated the phrase spotting performance automatically on the first 10,000 paragraphs of a Dutch held-out data set from Wikipedia. Table 3 shows the results for the default spotting algorithm of the current DBpedia Spotlight build, as well as the language-independent FSA-based implementation and language-dependent OpenNLP-based implementation of our build.

<table>
<thead>
<tr>
<th>Language</th>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>Lang. dep.</td>
<td>49.45</td>
<td>55.53</td>
<td>52.32</td>
</tr>
<tr>
<td></td>
<td>Lang. indep.</td>
<td>48.26</td>
<td>55.01</td>
<td>51.42</td>
</tr>
<tr>
<td></td>
<td>Current</td>
<td>8.26</td>
<td>77.17</td>
<td>14.92</td>
</tr>
</tbody>
</table>

Table 3: Phrase spotting evaluation

3.3 Disambiguation

We automatically evaluated the disambiguation performance of both builds for Dutch on 28,475 randomly selected paragraphs from the full held-out data set with only ambiguous annotations. Results are shown in Table 4. MRR (mean reciprocal rank) indicates n-best performance and No URI is the percentage of annotations in which the correct URI was not among the first 20 candidates. Note that the current build for Dutch is trained on the full Wikipedia data. Our build is trained only on the training section of the Wikipedia corpus, excluding the held-out sections. The English results are measured on an independent test corpus.

<table>
<thead>
<tr>
<th>Language</th>
<th>Model</th>
<th>Accuracy</th>
<th>MRR</th>
<th>No URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dutch</td>
<td>Our build</td>
<td>0.841</td>
<td>0.622</td>
<td>0.076</td>
</tr>
<tr>
<td></td>
<td>Current</td>
<td>0.581</td>
<td>0.432</td>
<td>0.367</td>
</tr>
<tr>
<td>English</td>
<td>Our build</td>
<td>0.851</td>
<td>0.797</td>
<td>0.074</td>
</tr>
<tr>
<td></td>
<td>Current</td>
<td>0.716</td>
<td>0.688</td>
<td>0.166</td>
</tr>
</tbody>
</table>

Table 4: Disambiguation evaluation

3.4 Internationalized Models

We created models for a number of languages among the largest Wikipedias. Table 5 presents performance values as well as the size of the respective input corpus for each of these languages. As above, we show accuracy, mean reciprocal rank and the percentage of annotations for which the correct URI was not found for the disambiguation sub-task. Additionally, we include the accuracy Acc, of a strong baseline for each language. The baseline is to always pick the...
most common sense $\hat{e} = \arg\max_{e} \text{count}(e, s)$ for each phrase using the same candidate mapping as the other algorithms.

For the phrase spotting task, the table shows precision, recall and $F_1$ score on heldout data, as well as the type of spotting algorithm used in the system:

**Tokenization:** T indicates that the system uses a supervised OpenNLP tokenization model and I indicates a semi-supervised tokenizer based on the java.text package.

**Spotting algorithm:** FSA uses a token-based finite state automaton to retrieve exact matches, C and NER indicate the use of OpenNLP models for phrase chunking and named entity recognition.

The performance values for each model are estimated on a heldout data set consisting of 6,000 paragraphs randomly selected from the input corpus. When examining the randomly selected paragraphs, we observed that they are generally short. It can be expected that a bigger improvement over the baseline will be achieved when more context is available. For evaluating disambiguation, we only consider ambiguous phrases.

For evaluating disambiguation, we only consider ambiguous phrases.

5. ACKNOWLEDGMENTS

Parts of this work were funded by Google Summer of Code 2012 and by the FP7 grant Dicode (GA no. 257184). The authors would like to thank David de Boer and Gossé Bouma for help with the Dutch model. We also thank Faveeo and Globo.com for support and computing resources. Special thanks to MTA Sztek (through Mihály Hédér) for providing the necessary computing resources for creating the internationalized models.

6. REFERENCES