Abstract.
This deliverable introduces the refined LODIE prototype implementation of ontology-based information extraction, using Linked Open Data (LOD). LODIE incorporates DBpedia into the extraction process and performs disambiguation in order to unambiguously associate annotations with URIs. In this second version of LODIE, it has been extended to German, Hindi, and Bulgarian, as well as optimised for speed and memory efficiency. The disambiguation algorithm has also been extended beyond the closed world assumption that all all entities appearing in text must have a corresponding URI in the LOD resource. The latter is a significant shortcoming of many other approaches (including LODIE’s earlier version itself) since new named entities appear in social media practically daily and it can take months or even over a year for this to be reflected in the next release of DBpedia. This document additionally shows example of the population of TMO ontologies, described in details in D.2.1.2, and which are complementary to DBpedia for deployment in the use cases.

Keyword list: information extraction, ontologies
TrendMiner Consortium

This document is part of the TrendMiner research project (No. 287863), partially funded by the FP7-ICT Programme.

DFKI GmbH
Language Technology Lab
Stuhlsatzenhausweg 3
D-66123 Saarbrcken
Germany
Contact person: Thierry Declerck
E-mail: declerck@dfki.de

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Contact person: Kalina Bontcheva
E-mail: K.Bontcheva@dcs.shef.ac.uk

University of Southampton
Southampton SO17 1BJ
UK
Contact person: Mahensan Niranjan
E-mail: mn@ecs.soton.ac.uk

Ontotext AD
Polygraphia Office Center fl.4,
47A Tsarigradsko Shosse,
Sofia 1504, Bulgaria
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Internet Memory Research
45 ter rue de la Revolution
F-93100 Montreuil
France
Contact person: France Lafarges
E-mail: contact@internetmemory.org

Sora Ogris and Hofinger GmbH
Linke Wienzeile 246
A-1150 Wien
Austria
Contact person: Christoph Hofinger
E-mail: ch@sora.at

Eurokleis S.R.L.
Via Giorgio Baglivi, 3
Roma RM
00161 Italy
Contact person: Francesco Bellini
E-mail: info@eurokleis.com

Hardik Fintrade Pvt Ltd.
227, Shree Ram Cloth Market,
Opposite Manilal Mansion,
Revdi Bazar, Ahmedabad 380002
India
Contact person: Suresh Aswani
E-mail: m.aswani@hardikgroup.com
Executive Summary

This deliverable introduces the refined LODIE prototype implementation of ontology-based information extraction, using Linked Open Data (LOD). LODIE incorporates DBpedia into the extraction process and performs disambiguation in order to unambiguously associate annotations with URIs. In this second version of LODIE, it has been extended to German, Hindi, and Bulgarian, as well as optimised for speed and memory efficiency. The disambiguation algorithm has also been extended beyond the closed world assumption that all all entities appearing in text must have a corresponding URI in the LOD resource. The latter is a significant shortcoming of many other approaches (including LODIE’s earlier version itself) since new named entities appear in social media practically daily and it can take months or even over a year for this to be reflected in the next release of DBpedia.

This document additionally shows example of the population of the TrendMiner ontologies (TMO), described in details in D.2.1.2: Knowledge and Provenance modelling and Stream Reasoning v2. TMO are complementing the LOD-based knowledge sources (mainly DBpedia) used until now in TrendMiner. First experiments and user evaluation done in the context of WP7 have shown that it is very useful to have some ontological schemas customized to the needs of the use case partner. In the case of WP7, this is related to the political environment in Austria. In the second part of this deliverable, we just show some examples of a subset of the TM ontologies (political, biography, opinions and interface ontologies) populated with data extracted from a list of candidates to the national elections 2013 and from a German newspaper. Actual work is dedicated in cross-linking the TMO Ontologies with DBpedia. The schema of the TM Ontologies is available at http://www.dfki.de/lt/onto/tmo.owl.
Contents

1 Introduction .................................................. 2
  1.1 Relevance to TrendMiner ................................... 3
    1.1.1 Relevance to project objectives ......................... 3
    1.1.2 Relation to other workpackages .......................... 3

2 Ontology Based IE and Disambiguation ....................... 4
  2.1 Identifying Candidate LOD Instances ....................... 4
  2.2 Similarity Metrics ........................................ 4
    2.2.1 String Similarity ..................................... 5
    2.2.2 Semantic (Structural) Similarity ....................... 5
    2.2.3 Contextual Similarity .................................. 6
    2.2.4 Commonness ........................................... 8
  2.3 Disambiguation ............................................ 9
  2.4 Adapting LODIE to Other Languages ...................... 10
    2.4.1 Components of the Disambiguation Application ........... 11
    2.4.2 Adapting LODIE to German ............................. 12
    2.4.3 Adapting LODIE to Hindi ............................... 12
    2.4.4 Adapting LODIE to Bulgarian ......................... 14

3 Populating the TrendMiner Ontologies (TMO) ................ 15

4 Software Availability ....................................... 18
Chapter 1

Introduction

Semantic annotation is the process of tying semantic models and natural language together. It may be characterised as the dynamic creation of interrelationships between ontologies/knowledge bases and unstructured or semi-structured documents. From a technological perspective, semantic annotation is about annotating in texts all mentions of concepts from the ontology (i.e., classes, instances, properties, and relations), through metadata referring to their URIs in the ontology.

Ontology-based entity recognition is a kind of semantic annotation, typically broken down into two main phases: candidate selection (entity annotation) and entity linking (also called reference disambiguation or entity resolution) [RMD13]. Entity annotation is concerned with identifying in a given text all candidate mentions of instances from a knowledge base (e.g., DBpedia). The entity linking step then uses contextual information from the text, coupled with knowledge from the ontology, to choose the correct instance URI. If there is no such corresponding instance, then a NIL value needs to be returned (an open domain assumption). In particular, entity linking needs to handle name variations (entities are referred to in many different ways) and entity ambiguity (the same string can refer to more than one entity) [JG11, RMD13].

Linking Open Data (LOD) resources, in particular DBpedia and YAGO, have emerged as key sources of large-scale ontological knowledge, as well as being used as target entity knowledge bases for entity linking. These offer:

1. cross-referenced domain-independent hierarchies with thousands of classes and relations and millions of instances;
2. an inter-linked and complementary set of resources with synonymous lexicalisations;
3. grounding of their concepts and instances in Wikipedia entries and other external data.

The rich class hierarchies are used for fine-grained classification of named entities, while the knowledge about millions of instances and their links to Wikipedia entries are used as features in the entity linking algorithms. However, as noted by [GNP+09], the large-scale nature of LOD resources also makes entity linking particularly challenging, due to the ambiguity introduced by the presence of so many instances.

This deliverable introduces the refined LODIE prototype implementation of ontology-based information extraction, using Linked Open Data (LOD). LODIE incorporates DBpedia into the
extraction process and performs disambiguation in order to unambiguously associate annotations with URIs. The focus is on open-domain entity linking, similar to that carried out by DBPedia Spotlight, as opposed to domain-specific approaches (e.g. [GNP+09])

In this second version of LODIE, it has been extended to German, Hindi, and Bulgarian, as well as optimised for speed and memory efficiency. The disambiguation algorithm has also been extended beyond the closed world assumption that all all entities appearing in text must have a corresponding URI in the LOD resource. The latter is a significant shortcoming of many other approaches (including LODIE’s earlier version itself) since new named entities appear in social media practically daily and it can take months or even over a year for this to be reflected in the next release of DBpedia.

1.1 Relevance to TrendMiner

TrendMiner aims to work over large volumes of streaming media and end users require the text to be categorized in order to help them quickly understand events. Polarity classification is one way in which streaming media can be classified, and the information extraction and disambiguation approaches reported can serve as clues to polarity.

The majority of the work reported in this deliverable is related to Tasks 2.1 (Multilingual knowledge and lexical acquisition for customizing Ontology Schema) and 2.3 (Multilingual, ontology-based IE from stream media: entities, events, sentiment, and trends). As this is a prototype deliverable and does not focus on evaluation only a small amount of work related to Task 2.4 is included where we thought it appropriate.

1.1.1 Relevance to project objectives

The work reported in this deliverable provides the software processing required for both information extraction and disambiguation over streaming media.

1.1.2 Relation to other workpackages

Information extraction and disambiguation are requirements of the two use cases (WP6 and WP7) as well as being preparatory steps required for the summarization work being carried out in WP4.
Chapter 2

Ontology Based IE and Disambiguation

2.1 Identifying Candidate LOD Instances

The first step is to identify all candidate instance URIs from DBpedia, which are mentioned in the given document. This candidate generation step uses the lexical information associated with LOD instances, in order to build very large gazetteer lists. These are then used to perform lookups of n-grams derived from the document text. The DBpedia lexicalisation properties used in our experiments are `rdfs:label`, `db:name`, `foaf:nick`, `db:nickname`, `db:official_name`.

The DBpedia-based candidate selection was implemented using the open-source GATE Large Knowledge Gazetteer (LKB)\(^1\) [CMB\(^+\)11]. LKB performs fast string lookup and assigns URIs to words/phrases in the text.

For DBpedia (which has mappings to Wikipedia pages), additional lexicalisations for each URI are acquired from link anchor texts, disambiguation pages, and redirect pages from Wikipedia. This has been shown [JG11, RMD13] to help improve the recall of the candidate selection phase. For instance, many abbreviations and acronyms are acquired in this way.

We have also used the open-source ANNIE Information Extraction system [CMB\(^+\)11] to assign entity types for entity candidates. ANNIE also resolves within-document coreference, so that mentions of the same entity within a document are linked together. For example, European Environmental Agency and EEA would be marked as referring to the same entity. This co-reference information is used to restrict the textual context considered in the subsequent entity disambiguation stage.

2.2 Similarity Metrics

The entity disambiguation algorithm uses the textual context in which a given candidate entity appears to calculate a number of similarity metrics. These are then used by the disambiguation algorithm (see Section 2.3) to determine the best matching candidate URI.

The four classes of metrics are:

\(^1\)http://gate.ac.uk/userguide/sec/gazetteers/lkb-gazetteer
String similarity: edit distance between the text string (such as Paris), and the lexicalisations of the entity URIs (e.g. Paris and Paris, Texas).

Semantic (structural) similarity: calculated based on the ontology and instance property values in the LOD resource.

Contextual similarity: the probability that two words have a similar meaning, based on random indexing.

Commonness: a normalised frequency metric, calculated against Wikipedia and the LOD resource.

2.2.1 String Similarity

For each candidate URI, two string similarity metrics are calculated. The first compares the text of the candidate with the resource name of the URI in order to provide a measure of how well they fit; for example, where “Paris” has a candidate with the URI “http://dbpedia.org/resource/Paris(Texas)”, “Paris” is compared to “Paris,(Texas)”. The second is calculated using a context of 30 tokens on both sides of the candidate, including all sentences from any co-reference chain, rather than simply the string; this longer context string is compared to the resource name of the URI. For efficiency reasons, only named entities are used.

After some experiments with different string similarity metrics (Levenstein, Jaccard, and MongeElkan [JBGG09]), the Levenstein (or string edit distance metric) was chosen. In a nutshell, the Levenstein score of two strings is equivalent to the number of substitutions and deletions needed to transform one string into the other. More formally, let $s$ be the source string and let $t$ be the target string. The distance is the number of deletions, insertions, or substitutions required to transform $s$ into $t$. If $s$ and $t$ are identical, then $LD(s,t) = 0$, because no transformations are needed. If $s$ is “matrics” and $t$ is “metrics”, then $LD(s,t) = 1$, because one substitution is sufficient to transform $s$ into $t$. The greater the Levenshtein distance, the more different the strings are.

This approach to determining string similarity is applied to both the metrics described above; that is to say, one using the string itself and the other using a context window. A third metric averages the two. Future work will involve empirically determining the optimum size for the context window.

2.2.2 Semantic (Structural) Similarity

Semantic (structural) similarity is calculated based on whether the ambiguous candidate NE has a relation with any other NE from the same sentence or document. For example, if the document mentions both Paris and France, then semantic similarity assigns the highest score to db:Paris, as the two are connected directly via the db:country property. On the other hand, if Paris appears in the context of USA, the semantic similarity metric will assign higher score for db:Paris, _Texas. The latter is derived by combining the DBpedia knowledge that Paris, Texas is part of Lamar County and the latter has country United States.

Semantic similarity scores are first computed against any unambiguous URI instances, found in the context. If this fails to produce results, other NE candidates from the context are prioritised,
based on how close they are to the ambiguous entity (measured in number of intermediate tokens) and on which side of the entity they are (left context vs right context).

For efficiency reasons, context size is limited to 60 tokens around the candidate (30 to the left and 30 to the right), as well as all sentences including co-referent mentions of this candidate (as determined by ANNIE). Nevertheless, on larger document sets this could lead to a large number of SPARQL queries needing to be fired. This was optimised through caching.

2.2.3 Contextual Similarity

Contextual similarity methods aim to evaluate candidates on the basis of how well a context window around the candidate matches representative text from the target URI of the candidate. Three approaches to contextual similarity are implemented. Firstly, a vector-space approach compares context to candidate using a cosine measure of similarity. Secondly, we use Random Indexing to select a set of representative words from the candidate target and compare this with the context of the text to be disambiguated. Thirdly, a Lucene search retrieves candidate URIs based on a query constructed from the context window, and the score and rank of each candidate is then used to provide further metrics. Each approach is described in more detail below.

Vector-Space Approach

A baseline vector-space approach to contextual similarity is taken whereby a context window from the named entity is compared against the content of the abstract for each candidate using the cosine measure of similarity.

A semantic space has been created using TAC 2010 data [JGD+10], providing background data on the types of words that tend to co-occur with each other. By using a corpus of task-appropriate text to create a semantic space, background information about which words are likely to co-occur can be used, which allows for a greater sensitivity in determining the similarity between the two strings to be evaluated at runtime. For example, in our context window, we may have Paris and Europe, but no mention of France, and there may be no mention of Europe in the target article describing Paris, France. However, the background corpus indicates that France and Europe are highly related. So by using a semantic space to map our vectors into a richer representation, we can make use of this information. Dimensionality reduction techniques are often employed at this stage; however, no dimensionality reduction is performed in this simple, baseline approach to contextual similarity. Such approaches may be tried in the future.

The words in the context of the named entity are then mapped into this semantic space, where they can be compared to the words from the abstract of the candidate. The size of the context window is empirically determined, and currently rests at 300 characters to either side, or the boundaries of the document in the case that they are closer. Experimentation indicates that increasing the size of the context window improves the result; however, the improvement beyond 300 characters is negligible. Currently, the entire of the string is used unaltered. Future work will involve evaluating the impact of selecting particular types of words rather than using the whole string.

The entire of the target abstract is used, and is retrieved at runtime using a SPARQL query,
before being similarly mapped into the semantic space and compared with the context vector using the cosine, which then constitutes the metric.

This approach produces a good result, and forms a valuable part of the overall system. Future work would involve increasing the size of the semantic space and extending the approach to languages other than English.

**Random Indexing**

For calculating a further measure of contextual similarity, we use Random Indexing (RI) [Sah05], to select salient words from the target abstract that can then be compared with the entity context. Random Indexing and other similar methods rely on the Harris Distributional Hypothesis, i.e. words occurring in similar contexts tend to have similar senses. For efficiency reasons, we indexed only DBpedia abstracts as context for each URI which refers to either dbpedia:Person, dbpedia:Organisation or dbpedia:Place. This means that our initial term × document matrix looks as follows:

\[
\begin{array}{ccc}
\text{term1} & \text{term2} & \ldots & \text{termN} \\
dbpedia:Paris & 5 & 4 & \ldots & \ldots \\
dbpedia:London & 3 & 12 & \ldots & \ldots \\
\ldots
\end{array}
\]

Generating and searching through the semantic space is computationally costly and in order to make it more efficient we pre-processed the abstracts and included only proper and common nouns in the term × document matrix. The corpus contained 3 million terms and 3.5 million documents. We used dimensionality of 150, seed length of 4 and minimum term frequency of 3, which reduced the semantic space to around 1.2 million terms. The selection of these parameters is based on our earlier experiments with DBpedia [DSL12].

Once the semantic space was computed, we used it to find terms related to the specific documents (URIs). When calculating the contextual similarity score, for a given candidate URI, we first retrieve the top \( n \) related terms, and then calculate string similarity with the context of the ambiguous NE. Our approach to calculating string similarity is described above in section 2.2.1. The success of the approach has shown itself largely insensitive to the choice of number of salient words retrieved; future work will involve determining how far this number can be reduced without impacting on performance. The number currently rests at 20.

**RDF-Based Contextual Similarity**

There are three main types of triples in DBpedia:

1. Triples with type/class information (e.g. Barack Obama is a Person, United States is a Country).

2. Triples that give us metadata information about a URI, i.e. annotation properties (e.g. Barack Obama is also known as Mr President, United States is often written as USA or US).
3. Triples that describe relations between individual entities, i.e. object-type properties (e.g. Barack Obama is a president of United States).

Here, the focus of the RDF-Based Contextual Similarity is on the second and third types of triples. The idea is to look up in the text surrounding the entity to be disambiguated and see if there is any similarity with the metadata information of the candidate URIs.

For every candidate URI, a document is built containing its labels, values of annotation properties, comments, aliases and the lexicalisations of all URIs which are directly connected via an object property. Such documents are then indexed with Lucene.

Given an entity to be disambiguated (for which a number of candidate URIs have been obtained), a query vector (OR query) containing all nouns and proper nouns from the context (currently 100 characters either side) is built. The query is executed in Lucene to find out the top $n$ (currently 500) URIs that match the query vector. Lucene returns a ranked list with the first candidate having the highest similarity and the last candidate having the least similarity amongst the returned results.

We use the following equation to calculate a score based on the rank of a candidate URI in the Lucene index.

$$\text{rdfScore}(aURI) = 1 - \frac{\text{rank}(aURI)}{n} + 0.0001$$

Here,

- $\text{rank}(aURI) =$ position of the URI in the list of candidate URIs found in the result returned by Lucene;
- $n =$ total number of candidate URIs found in the result returned by Lucene.

For example, if Lucene returns 500 URIs in total, but only 20 candidates appear in the list and if position of the current candidate URI in the list of found candidates is 5, the score would be calculated as follow:

$$1 - \frac{5}{20} + 0.0001 = 0.7501$$

In case a candidate URI is not present in the list of URIs (returned by Lucene), it is assigned a score of 0. The small constant (0.0001) is added to the score of every URI found in the candidate list to avoid assignment of 0 to the candidates appearing last in list of results.

### 2.2.4 Commonness

Given an ambiguous named entity and a set of real-world candidates, one can think of commonness being defined as the first identity that comes to one’s mind. For example, there might be several people named “Obama” but at present, the most likely association is with the President of United States (unless there is additional contextual evidence suggesting a different interpretation).

In this section, we discuss two different types of commonness similarity metrics. First, given a set of candidate URIs, which of the candidate URIs is the most common in Wikipedia. Second, which amongst the given candidate URIs is the most common in terms of how many other entities are linking to it.
CHAPTER 2. ONTOLOGY BASED IE AND DISAMBIGUATION

Wikipedia-based Commonness

The commonness metric reflects the assumption that if a named entity is mentioned frequently in Wikipedia, then it will be also more common within other corpora. Due to the one-to-one mapping between English Wikipedia URLs and DBpedia URIs, the commonness score for a candidate URI is assigned using the commonness metric defined by [MW08] for Wikipedia pages. This has also been referred to as popularity [RMD13, ACJ+09]. However, unlike [RMD13], for efficiency reasons we do not use Google queries as additional evidence.

RDF-based Commonness

URIs have direct relations with other URIs. For example, db:Paris is directly related to db:France. Similarly, db:Paris, _Texas is related Lamar_County, _Texas.

RDF-based commonness is a calculation of how many instances are directly connected to the current URI in comparison to number of instances connected to other candidates for the same span.

If Paris (France) has more direct relations with other URIs in DBPedia than the Paris (Texas), popularity of the former in DBPedia would be higher.

2.3 Disambiguation

As described above, a variety of similarity metrics are generated, each providing different information about the fit of each candidate URI to the target entity. Metrics may be present or absent, and differ relative to each other in their reliability and discriminatory ability. The process of deciding how to combine these metrics to select the best candidate is therefore non-trivial.

The approach taken here involves using a maximum entropy implementation to decide, based on a training set, how best to combine the features to select the best candidate. The choice of maximum entropy was made empirically–maximum entropy was found to consistently produce a good standard of result, and proved to be a good fit to the task. The Mallet implementation was used. The training set comprised years 2010 through 2012 of TAC KBP data 2, along with the AIDA training set [HYB+11]. AIDA data is chosen because a large number of annotated examples are provided, and the division into three sets, one for training, one for tuning and one for evaluation, allow us to compare our system performance directly with others. TAC KBP data consists of training data provided for the knowledge base population section of the Text Analysis Conference shared task. Addition of this data allows us to increase the amount of training data available.

This step also involves the detection of nils; that is to say, entities for which there is no URI available. The machine learning step is trained on a dataset that consists of a selection of candidates, each with their set of metrics and a class of true or false, derived from the training data and indicating whether the candidate in question was the correct candidate. In this way, the machine learning step learns to identify with varying degrees of confidence whether this is a good candi-

2http://www.nist.gov/tac/2013/KBP/
date or not. Selecting the final choice at application time then becomes a matter of choosing the “true” with the highest confidence. A confidence threshold is applied such that in some cases, no candidate is “true” with a sufficiently high confidence, and in that case, the final result is a nil. In fact, the algorithm tends to be over-strict, so a “negative” confidence threshold may be applied that allows a final choice that is “false” with a low confidence. The final choice of confidence threshold was determined based on the best result achieved on the AIDA tuning set [HYB+11].

Criticisms of the approach include that the training set may contain good candidates that happen not to be the best candidate (for example, the United States of America and the US Government may be equally valid in certain contexts). This means that the system is seeing examples of good fits that are labelled false. In fact, this might explain a tendency on the part of the system to label many good items false (we see a tendency toward high precision and low recall), that we then compensate for through the selection of a permissive confidence threshold. (Currently, we accept the best candidate if it is either labelled true or labelled false with a confidence score of less than 0.8. Otherwise, we consider nil to be the more likely conclusion.) Improvement might involve labelling training items as being an acceptable or unacceptable fit rather than simply the best or not the best; however this would be time consuming, since existing gold standard datasets could not be used.

We also developed a simple baseline overall score which involves adding the following metrics together:

- Commonness Similarity, as an average of RDF-based Commonness and Wikipedia-based Commonness
- String Similarity, as an average of Resource Name String Similarity and Context String Similarity
- The average of RDF Contextual Similarity and Random Indexing Similarity
- Structural Similarity

In case of tie-breaks, i.e. candidate URIs for the same textual mention and with the same overall score, the baseline system produces multiple annotations.

2.4 Adapting LODIE to Other Languages

Since English accounts now for less than 50% of social media content, TrendMiner aims to deliver a multilingual disambiguation system. In particular, we are currently testing on German (as the WP7 use case language) and Hindi and Bulgarian (two less resourced languages).

The first version of the LODIE entity disambiguation algorithms only worked for English. Nevertheless, it was designed to be easily adaptable to new languages by keeping components as generic as possible. In this section, we briefly discuss the various LODIE components and how they were adapted to support the three new languages.
2.4.1 Components of the Disambiguation Application

The Disambiguation application consists of three types of components:

- linguistic pre-processing
- candidate generation
- disambiguation

The pre-processing components are by their very nature language specific, since they carry out the necessary low-level linguistic analysis. They are the tools used for recognising:

- word and sentence boundaries,
- part-of-speech categories for individual words,
- named entities, and
- English transliterations for the entities written in languages other than English.

Given an entity or a span of text, the candidate generation components are used for obtaining a list of candidate URIs (from DBpedia). As specified in the Section 2.1, we use DBpedia resources to prepare a gazetteer with labels, names and aliases (including acronyms) of various entities. Execution of such a gazetteer creates Lookups highlighting candidate entities that should be disambiguated in the text.

Since our goal is to disambiguate entities, it is important to mention here that unless the script is different, it is very unlikely that the entities (proper names) would have been written in any different lexical form than they are in English. For example, a sentence which mentions "Microsoft" may have words other than proper names translated, but the word "Microsoft" would normally appear unchanged.

Thus, for languages with the same script as English (e.g. German, French), we still use the resources prepared for English and fetch candidate URIs. However, where the script is different, it is difficult to use the same English language resources. Comparing the size and amount of DBpedia resources available for different languages with the resources available for English, it is clear that the resources available for the other languages are too few or too small.

To address this issue and thus be able to utilise the English resources for other languages, we use a transliteration component (see Section 2.4.3) to obtain one or more English transliterations for each entity in the text. Each transliteration is then looked up in the English DBpedia resources and respective candidates are obtained.

Once the pre-processing is achieved and list of candidates are produced, a set of disambiguation components are executed. In particular, the following steps are taken:

1. Candidates produced in the first step are filtered out if they do not have at least one proper noun under the annotation span. This is to make sure that the spans being disambiguated are referring to entities only.
2. Wikipedia redirects are applied to the candidate URIs and any candidate URIs referring to pages with disambiguation URIs are excluded [JG11, RMD13].

3. Each candidate URI is then assigned scores by the similarity metrics described above.

4. Finally, the disambiguation algorithm uses all similarity scores to choose the best scoring candidate URI. This step is completely language independent, so it is reused for all language versions of LODIE.

In the language specific sections below, we describe how the pre-processing and transliteration components have been adapted to the different languages.

2.4.2 Adapting LODIE to German

Since the script used for writing German is the same as the script used for writing English, a German transliteration component was not required. Thus only the linguistic pre-processing components had to be adapted.

Preprocessing Tools

1. In order to recognise word and sentence boundaries we use the GATE Unicode tokeniser\(^3\) and the English sentence splitter\(^4\). The Unicode tokeniser uses white spaces (i.e. space, tab, new line characters) as word delimiters. It also takes into account special usage of apostrophies, dots, hyphens etc.

2. TreeTagger\(^5\) is used for recognising the part-of-speech category for each word.

3. We do not perform any named entity recognition for German. Instead, all spans which contain at least one proper noun and have assigned one or more candidate URIs, are considered for disambiguation.

2.4.3 Adapting LODIE to Hindi

Hindi is an Indo-European language, mostly written in the Devanagari script.

In order to process Hindi text, we make the use of GATE’s Hindi resources\(^6\).

Linguistic Pre-processing Tools

1. In order to recognise word and sentence boundaries, we use the Hindi tokeniser and Hindi sentence splitter from GATE.

\(^3\)http://gate.ac.uk/sale/tao/splitch21.html#x26-51400021.3
\(^4\)http://gate.ac.uk/userguide/sec:annie:splitter
\(^5\)http://www.cis.uni-muenchen.de/schmid/tools/TreeTagger/
\(^6\)http://gate.ac.uk/userguide/sec:misc-creole:language-plugins:hindi
2. GATE also has a part-of-speech tagger for Hindi which helps in assigning part-of-speech tags to individual words. These tags are used later in the process to filter out some of the candidates (i.e. non proper nouns).

3. GATE also has a Named Entity Recognition system for Hindi, which recognises mentions of Persons, Locations and Organizations.

4. Specifically for LODIE, we developed a Hindi-English transliteration algorithm that given a Hindi string returns one or more English transliterations. We describe this next.

**Hindi-English Transliteration System**

When translating a sentence from a source language into a target language, named entities such as person names, names of places, organizations, etc. are not translated but transcribed into the writing system of the target language. Transliteration is defined as the task of transcribing a word or text from one writing system into the another writing system such that the pronunciation of the word remains the same and a person reading the transcribed word can read it in his language.

In order to transliterate words written in Devanagari into their respective English transliteration, we use a system developed by [AG10]. In their system the authors have presented a bi-directional mapping between one or more characters in the Devanagari script and one or more characters in the Roman script (pronounced as in English). They also present an algorithm for computing a similarity measure which also takes into account the constraints needed to match English-Hindi transliterated words. They use this system as part of a English-Hindi word alignment system [AG09] where they transliterate named entities written in Hindi to align them with their respective occurrence of English entities in the English language. They report accuracy of 0.95 when used for identifying transliteration pairs.

In our disambiguation system, after having identified the named entities in the text, each of these entities is sent to the transliteration system. The transliteration system, using the mapping it has for English and Devanagari characters, produces one or more transliterations. In its simplest form, once the transliterations are obtained, one can compare these transliterations with the labels available in DBpedia and select those transliterations which match best. We explain the procedure of selecting transliterations below:

- Prepare a list of unique words appearing in labels from DBpedia;
- Given an entity for which transliterations need be obtained, transliterate one word at a time and obtain all possible transliterations which exist in the list of unique words obtained in step 1. If there is no match found, a fuzzy match approach is used to obtain the nearest DBPedia label. If even after that there is no match found, the shortest 15 transliterations are preserved. Thus, there is always one transliteration available for every word being transliterated.
- Having obtained transliterations for every word, a list all possible combinations is produced. For example, if a phrase has three words and for the first word there are 2 transliterations, for the second word there are 3 transliterations and for the third word there are 2 transliterations, it gives us $2 \times 3 \times 2 = 12$ combination of transliterations.
• A gazetteer with labels of entities from DBpedia is executed on this list to choose strings matching the DBpedia labels. All the transliterations that match with DBpedia labels are selected. If there is no match found, a fuzzy match approach is used to obtain the nearest DBpedia labels.

Once the transliterations are obtained, LODIE uses these transliterations to obtain candidates from the English DBpedia. These candidates are then scored with the similarity metrics defined above. As mentioned in the Section 2.2, most of the similarity metrics make use of contextual information. If the words in context are written in Devenagari, they cannot help much. In order to solve this problem, we use a Hindi-English dictionary that given a Hindi word, obtains all its possible translations from the dictionary. These words are then used by the various similarity metrics when referring to the contextual information.

2.4.4 Adapting LODIE to Bulgarian

Adaption of LODIE to Bulgarian is not very different from the process specified for Hindi (see Section 2.4.3), except that Bulgarian uses Cyrillic script.

Preprocessing Tools

1. In order to recognise word and sentence boundaries we use the Unicode tokeniser and Sentence splitter from GATE.
2. We use TreeTagger to find out part-of-speech tags for the Bulgarin words. These tags are used later in the process to filtering out candidates with no proper nouns.
3. We use a sequence of proper nouns as possible spans to be disambiguated.
4. Finally, we have developed a Bulgarian-English transliteration system that given a Bulgarian string returns one or more English transliterations for the string. The transliteration system is the same as the one described for Hindi, except for the mapping file. In this case we use a character mapping file dervied from a wikipedia page on Romanization of Bulgarian text.

Similar to the Hindi system, once the transliterations are obtained, the system uses these transliterations to obtain candidates from the English DBpedia. These candidates are then scored with various similarity metrics. We use a standard Bulgarian-English dictionary that given a Bulgarian word, obtains all its possible translations from the dictionary. These words are then used by the various similarity metrics when referring to the contextual information.

7http://www.shabdkosh.com/archives/content/shabdanjali_-english_hindi_dictionary/
Chapter 3

Populating the TrendMiner Ontologies (TMO)

As stated in the executive summary of this deliverable, we are starting to implement the population step of the TrendMiner Ontologies (TMO). The schema of TMO has been described in D.2.1.2: Knowledge and Provenance modelling and Stream Reasoning v2, and is available at www.dfki.de/lt/onto/tmo.owl. Our first experiment for this task has been pursued in the context of WP7 and is dealing with political personalities and institutions in Austria. As structured input we got from the partner SORA a full list of candidates to the National Austrian elections (September 2013) and we also performed a limited analysis of some on-line media. Our goal was first to get a high precision population that can be checked by the use case partner SORA for the soundness of this population step. A quantitative evaluation will be performed in the next period of the project, when the population of TMO is done on a purely automated fashion. Taking now as an example the case of the actual Chancellor of Austria, Werner Faymann, we get the following types of information (or facts) included in the A-Box of TMO.

Basic (biographical) facts about the person and the politician Faymann (extracted from structured data):

<!-- http://www.dfki.de/lt/political.owl#werner_faymann -->

<owl:NamedIndividual rdf:about="&pol;werner_faymann">
  <rdf:type rdf:resource="&biography;Man"/>
  <rdf:type rdf:resource="&pol;Chancellor"/>
  <rdf:type rdf:resource="&pol;Politician"/>
  <rdfs:label xml:lang="de">Werner_Faymann</rdfs:label>
  <listrank rdf:datatype="&xsd;integer">1</listrank>
  <biography:dateOfBirth rdf:datatype="&xsd;date">1960-05-04</biography:dateOfBirth>
  <occupation xml:lang="de">Bundeskanzler</occupation>
  <OWLDataProperty_4bcda9a0_b1e1_4ffd_9dda_05ea5ddc1090 xml:lang="de">Bundeskanzleramt (BKA)</OWLDataProperty_4bcda9a0_b1e1_4ffd_9dda_05ea5ddc1090>
  <last&gt;LastName xml:lang="de">Faymann"</last>
CHAPTER 3. POPULATING THE TRENDMINER ONTOLOGIES (TMO)

Mitglied des Bundesparteipräsidiums
Parteivorsitzender
Werner
Werner Faymann
männlich
candidatesFor rdfs:resource="&pol;nr2013"/
worksFor rdfs:resource="&pol;regierung_faymann"
worksFor rdfs:resource="&pol;sp"
biography:bornIn rdfs:resource="&pol;vienna"/
</owl:NamedIndividual>

Opinion that can be derived from a statement about the person Faymann:

<!-- http://www.dfki.de/lt/political.owl#Opinion_1_Faymann -->

<owl:NamedIndividual rdf:about="&pol;Opinion_1_Faymann">
<rdf:type rdf:resource="&opinion;Opinion"/>
<opinion:polarityValue rdf:datatype="&xsd;double">0,2</opinion:polarityValue>
<opinion:holdersTrust rdf:datatype="&xsd;double">9,0</opinion:holdersTrust>
<opinion:opinionText xml:lang="de">Denkzettel für Faymann und Spindelegger</opinion:opinionText>
<opinion:hasHolder rdfs:resource="&pol;n-tv"/>
<opinion:describesObject rdfs:resource="&pol;werner_faymann"/>
</owl:NamedIndividual>

The name Faymann is also used to name the actual government (a bi-partite coalition):

<!-- http://www.dfki.de/lt/political.owl#regierung_faymann -->

<owl:NamedIndividual rdf:about="&pol;regierung_faymann">
<rdf:type rdf:resource="&pol;Government"/>
<rdfs:label xml:lang="de">Regierung Faymann</rdfs:label>
<formedBy rdfs:resource="&pol;sp"/>
<formedBy rdfs:resource="&pol;vp"/>
</owl:NamedIndividual>

We also have information about the party Faymann is belonging to:

<!-- http://www.dfki.de/lt/political.owl#sp -->

<owl:NamedIndividual rdf:about="&pol;sp">
<rdfs:label xml:lang="de">SP</rdfs:label>
<rdfs:label xml:lang="en">Social Democratic Party of Austria</rdfs:label>
</owl:NamedIndividual>
Etc. The important point being that biographical information is linked (or better interfaced) with political information and also opinion/polarity detection. Inferences are also possible in this field (see D2.1.2 for more information): if Faymann, as the main candidate of his party, gets a negative polarity, we can assume that this is also the case for the party.
Chapter 4

Software Availability

The LODIE disambiguation pipeline is available via a web service described at http://demos.gate.ac.uk/trendminer/lodie/

This service uses the software described in this deliverable to process text to produce Mention annotations linked to DBpedia. To use the service POST an XML based request (the Content-Type parameter must be set to application/xml), such as the following, to http://demos.gate.ac.uk/trendminer/lodie/service/annotate

<request>
  <text>President Obama had flown back to United States after visiting Iran’s president.</text>
  <lang>en</lang>
</request>

The request should contain plain text within the XML `<text>` element. The language of the text should be provided within the `<lang>` element and can be one of `en` (English), `de` (German), `hi` (Hindi), `bg` (Bulgarian), `tweet-en` (English tweets), or `tweet-de` (German tweets). The values `tweet-en` and `tweet-de` indicate that tweets are to be processed and a different set of preprocessing modules adapted to tweets will be used.

A successful response to this request would be

<?xml version="1.0" encoding="UTF-8" standalone="yes">
  <message>
    <msg></msg>
    <status>SUCCESS</status>
    <text><![CDATA[&lt;Mention inst="http://dbpedia.org/resource/Barack_Obama"&gt;President Obama&lt;/Mention&gt; had flown back to &lt;Mention inst="http://dbpedia.org/resource/United_States"&gt;United States&lt;/Mention&gt; after visiting &lt;Mention inst="http://dbpedia.org/resource/Iran"&gt;Iran&lt;/Mention&gt;'s president.]]>
  </text>
</message>

The annotated text is returned in the `<text>` element of the response message as CDATA-encoded XML. Each annotated span of text is enclosed in a `<Mention>` element with an attribute `inst` that contains the disambiguated DBpedia URI.

If an error occurs then the `<status>` element would read ERROR and the underlying exception would be available within the `<msg>` element of the response. Note that for performance reasons the service currently restricts the text to process to no more than 2000 characters.

A web based demo of the service is also available at http://demos.gate.ac.uk/trendminer/obie/
Bibliography


