A modular framework to learn seed ontologies from text

Davide Eynard
Politecnico di Milano, Italy
Matteo Matteucci
Politecnico di Milano, Italy
Fabio Marfia

ABSTRACT

Ontologies are the basic block of modern knowledge-based systems; however the effort and expertise required to develop them are often preventing their widespread adoption. In this chapter we present a tool for the automatic discovery of basic ontologies –we call them seed ontologies– starting from a corpus of documents related to a specific domain of knowledge. These seed ontologies are not meant for direct use, but they can be used to bootstrap the knowledge acquisition process by providing a selection of relevant terms and fundamental relationships. The tool is modular and it allows the integration of different methods/strategies in the indexing of the corpus, selection of relevant terms, discovery of hierarchies and other relationships among terms. Like any induction process, also ontology learning from text is prone to errors, so we do not expect from our tool a 100% correct ontology; according to our evaluation the result is more close to 80%, but this should be enough for a domain expert to complete the work with limited effort and in a short time.

INTRODUCTION

In the last years there has been a considerable increase in research on knowledge-based systems, especially in the context of the Semantic Web. However these systems, as they aim at something more than just supplying trivial functionalities, suffer in their development process of the so-called *knowledge acquisition bottleneck*: creating large, usable, expandable and valid representations of semantics about a specific domain of interest, i.e., ontologies, represents the most time-consuming task of the whole project.

One of the main reasons for this knowledge acquisition bottleneck is that these formal representations, representing semantics previously unknown to machines, need to be manually annotated by domain experts. In this ontology building process, two different kinds of expertise are usually required: the knowledge of the domain that has to be described, and the ability to encode the ontology in a machine interpretable, i.e., computational, format. Unfortunately, satisfying both of these requirements is, in most cases, not trivial (if not impossible). On the one hand, domain knowledge might be very specific and known only to a small community of practice usually not in the realm of ontology writers; on the other hand, the encoding process does not only depend on standards and tools, but also on the specific context in which this knowledge has to be used.

Easing this task means either making semantic technologies more accessible to domain experts or providing ontology experts with structured information about a domain. The work presented in this chapter is mainly focused on the second approach and it aims at providing an alternative to the manual generation of ontologies through the automatic extraction of candidate concepts and relationships from a set of documents. The generated seed ontology is just an initial and approximate representation of the domain knowledge, and it obviously needs to be further modified and expanded, but it considerably reduces the time required for the overall formalization of the domain knowledge from scratch.

There is already plenty of information available on the Internet (and in other more reliable digital libraries) which could be used to teach a machine about virtually any domain of knowledge. This information is stored as collections of Web pages, large document corpora, databases, and so on. These repositories, however, cannot be directly consumed by a machine as they contain no structured information according to any standard model for knowledge representation. In this scenario, the main challenge we are interested in is the so-called *Ontology Learning from Text*. Free text is a valuable source of unstructured knowledge, and this requires researchers to adopt original heuristics in order to extract structured semantics from it. These heuristics often return inaccurate results, and have to be modified and validated by experts of the domain, but they can be exploited to bootstrap the whole ontology learning process.

Our work aims at providing a modular, semi-automatic framework that allows its users to apply different heuristics, (possibly) combine them, and finally measure their accuracy. The framework splits the process of ontology learning from text in well-defined steps and relies on different techniques at each stage. This provides some additional benefits to the whole process; for instance, it allows the use of the system with different combinations of algorithms, or to access any useful information generated at intermediate stages. The outcome of this work is a tool called Extraction, which allows the identification of main concepts and their basic relationships from a corpus of free text documents about a specific domain of knowledge.

This chapter has two purposes. On the one hand, we want to describe our modular architecture for the semi-automatic extraction of ontologies from text. This part should be especially useful for anyone trying to develop a similar system, as we provide insights into the building process and show the main issues we had to face, together with the choices we made to solve them. On the other hand, we review existing approaches for the extraction of relationships between terms, we evaluate them in the context of our framework, and we introduce a novel approach aimed at identifying new types of relationships, Action and Affection, which are distinct from classical subsumption.

The chapter is organized as follows: in the Background section we introduce the main concepts that characterize our work and describe the current state of the art about ontology learning from text. In the following section we suggest our design for a modular framework for ontology learning. This framework splits the whole process in steps, allowing users to customize them by applying their own metrics and to inspect the partial results returned by the system. Then the implementation of the framework is described, together with the description of the algorithms we decided to implement. In the following section we show the results of some experiments we ran by using Extraction, and finally we draw our conclusions about this project and suggest ideas for possible future work.

BACKGROUND

Ontology learning from text, in its most general meaning, is that branch of Information Retrieval that tries to generate formal and structured repositories of semantic information using, as its source, a corpus of unstructured text (typically dealing with a specific domain of knowledge). This process can be supervised by a human expert or use any sort of structured data as an additional source. In the former case, we talk about *assisted* or *semi-automatic learning*; in the latter, we refer to *oracle guided learning*; if the algorithm makes no use of structured sources or human help, it is considered an *automatic learner*. As an orthogonal definition, when the objective of the algorithm is the expansion of a pre-built ontology we usually talk about *bootstrapping* instead of *learning*.

Ontology learning is typically composed of different steps. Roughly, they can be grouped into two main families: acquisition of concepts (i.e., relevant terms extraction, name filtering, synonym identification, and concept identification) and acquisition of relations (i.e., hierarchies of concepts, other types of relations, and hierarchies of relations). As our experiments are mainly focused on different approaches for the acquisition of relations between concepts, we introduce these techniques in the following subsections.

Machine Readable Dictionaries and Hearst Patterns

Early work on extracting taxonomies from machine readable dictionaries (MRDs) goes back to the early '80s (Amsler, 1981; Calzolari, 1984). The main idea is to exploit the regularities of dictionary entries to find a suitable hypernym for a given word. An important advantage of using dictionary definitions to build a taxonomy is that dictionaries separate different senses of words, allowing a machine to learn taxonomic relations between *concepts*, rather than between *terms*. In many cases, the head of the first noun appearing in the dictionary definition is actually a hypernym. We can consider, for example, the following definitions quoted by Dolan et al. (1993):

- *spring*: "the season between winter and summer and in which leaves and flowers appear";
- *nectar*: "the sweet liquid collected by bees from flowers";
- *aster*: "a garden flower with a bright yellow center".

Of course, there are also some exceptions to the above rule. For instance, the hypernym can be preceded by an expression such as "a kind of", "a sort of" or "a type of". Some examples taken from Alshawi (1987) follow:

- *hornbeam*: "a type of tree with a hard wood, sometimes used in hedges";
- *roller coaster*: "a kind of small railway with sharp slopes and curves".

The above problem is easily solved by keeping an exception list with words such as "kind", "sort", "type" and taking the head of the noun following the preposition "of" as the searched term.

In general, approaches deriving taxonomic relations from MRDs are quite accurate. Dolan, for instance, mentions that 87% of the hypernym relations they extract are correct. Calzolari cites a precision of more than 90%, while Alshawi mentions a precision of 77%. These methods are quite accurate due to the fact that dictionary entries show a regular structure. We see, however, two main drawbacks in using a dictionary-based approach in ontology learning: the former is related to the fact that the acquired knowledge heavily depends on the stylistic (however regular) behaviour of the authors in writing dictionary entries. The latter is that in ontology learning we are mostly interested in acquiring domain-specific knowledge, while dictionaries are usually domain independent resources.

A different approach, that tries to exploit regularities in natural text and can therefore be employed to extract knowledge from domain-specific corpora, is the one suggested by Marti A. Hearst (1992) in her seminal work titled *Automatic Acquisition of Hyponyms from a Large Text Corpora*. The simple ideas contained in this work have heavily influenced many of the following approaches to ontology learning. The so-called *Hearst patterns* are a pre-defined collection of patterns indicating hyponymy relations in a text for a specific language. An example of such a pattern is:

"such $\mathbf{NP_0}$ as $\mathbf{NP_1}$, ..., $\mathbf{NP_{n-1}}$ (and|or) other $\mathbf{NP_n}$",

where ***NP*** stands for an English noun phrase. If this pattern is matched in a text, according to Hearst we can derive that, for every *i* from *1* to *n*, $\boldsymbol{NP_i}$ is a hyponym of $\boldsymbol{NP_0}$. The other patterns suggested by Hearst are the following ones:

- **NP** such as **NP**$^+$ (and|or) **NP**
- such **NP** as **NP**$^+$ (and|or) **NP**
- **NP**$^+$ or other **NP**
- **NP**$^+$ and other **NP**
- **NP** including **NP**$^+$ (and|or) **NP**
- **NP** especially **NP**$^+$ (and|or) **NP**

Here **NP**$^+$ refers to a comma-separated list of NPs, possibly composed of a single element. As mentioned by Hearst, the value of such lexico-syntactic patterns is that they are quite accurate and can be easily identified. Working on a set of textual documents from the New York Times, Hearst showed that, out of 106 extracted relations whose hyponym and hypernym appeared in WordNet, 61 were correct with respect to WordNet (i.e., 57.55% accuracy). The drawback of the patterns is however that they rarely appear and most of the words related through an *is-a* relation do not appear in Hearst-style patterns. Thus, one needs to process large corpora to find enough of these patterns. For this reason, more recently several researchers have attempted to look for these patterns using online search engines, i.e. considering the Web as a huge document repository (Markert, 2003; Pasca, 2004; Etzioni, 2004).

Distributional Similarity

A large number of methods for ontology learning from text are based on the same conceptual approach, called Distributional Similarity. This approach is based on the *distributional hypothesis* (Firth, 1957; Harris, 1968), according to which words found in similar contexts tend to be semantically similar. Different empirical investigations corroborate the validity of this hypothesis. Miller and Charles (Miller, 1991), for example, showed with several experiments that humans determine the semantic similarity of words on the basis of the similarity of the contexts they are used in. Grefenstette (Grefenstette, 1994) further showed that similarity in vector space correlates well with semantic relatedness of words. From a technological point of view, context similarity is analyzed by calculating the co-occurrence of words in the same sentence, paragraph, document, or other types of context. The more the distribution of two words in these contexts is similar, the more the words themselves are expected to be semantically similar.

In this family of approaches, concepts are extracted from a collection of documents and then organized using some representation that is based on their distributional similarity. Then, different methods can be used to identify relationships between neighbors. ASIUM (Faure, 1998), for example, is a software for the generation of concept hierarchies that uses as a context both the verb of the sentence where a concept appears and the syntactical function (i.e., subject, object, or other complements) of the concept itself. This tool leaves relation discovery to the user, by showing her similar words and allowing her to specify their hierarchical organization.

Caraballo (1999) presents an approach to build a hierarchy from a set of concepts extracted from a corpus of articles from the Wall Street Journal, with the parser described in (Caraballo and Charniak, 1998). The described model uses as a context the paragraph in which the terms appear and looks for Hearst patterns in the text to generate the hierarchy.

A different approach is the one called *Learning by Googling*: Hearst patterns can not only be found within document corpora, but they can also be searched on the Web. PANKOW (Cimiano, 2004), for instance, is a software that looks for Hearst patterns on Google and, according to the number of results returned by the engine, decides whether a subsumption relation between two concepts can be confirmed or not. A peculiarity of PANKOW is that it employs a variation of the classical Hearst patterns, using the plural form for classes of concepts (i.e. <CONCEPT>s such as <INSTANCE>). This choice is corroborated by

Ritter (2009), who found that different patterns are characterized by a high precision only when their class name is identified as a plural by a POS tagger.

An alternative model is the one presented by Fionn Murtagh (Murtagh, 2005; Murtagh, 2007). This is a Distributional Similarity approach that relies on Correspondence Analysis (a multivariate statistical technique developed by J.-P. Benzécri in the ’60s to calculate the semantic similarity between concepts (Benzécri, 1976). The generation of the hierarchy starts from the assumption that terms appearing in more documents are more general than others, thus the algorithm places them in a higher position within the hierarchy. Murtagh also applied Correspondence Analysis to study the distribution of concepts within Aristotle’s “Categories”, following the semantic evolution of Aristotle’s work in its 15 different parts; for instance, he found that parts 6, 8, 9 deal with some common concepts such as “category”, “quality” and “degree”. Parts 7, 10, 11 and 14, instead, appeared, in their distribution, very far from the others. The main limit of Murtagh’s approach is that, in the end, the inferred information is about how the text is organized while no real semantics is extracted, i.e., we have represented semantics about documents (or parts of them), and not about concepts expressed by the corpus itself. This is also true for other studies on linguistics presented by Murtagh in (Murtagh, 2005), because the distribution of terms is always studied across the documents.

For what concerns *bootstrapping* approaches, many methods have been developed which are based on distributional similarity (Hearst, 1993; Schutze, 1993; Alfonseca, 2002; Maedche, 2003). Their common approach is to evaluate the similarity between a word appearing in a corpus of documents and other words (concepts) in an ontology. Then, the word becomes a sub-concept of the concept whose children are more similar to it.

Formal Concept Analysis

Formal Concept Analysis, as described by Philipp Cimiano (2006), is an approach to extract concept hierarchies from text that departs from distributional similarity. It is based on different assumptions and largely relies on Natural Language Processing (NLP) algorithms. One of the main advantages of distributional similarity is that, in most cases, its algorithms can be largely considered to be language-independent, while this is not the case of FCA approaches (actually, the algorithms we previously described relying on Hearst patterns are language dependent; however changing patterns with their equivalents in other languages is a trivial task).

The main idea in FCA is to identify the actions that a concept can do or undergo. Words are organized in groups according to the actions they share; then they are ordered in a hierarchy according to these actions (i.e., the group of entities that can run and eat and the group of entities that can fly and eat are put together under the more general group of entities that can eat). Finally, the user is asked to label every node of the hierarchy (or other automatic methods can be used to perform this operation), then the final hierarchy is ready. Cimiano’s paper also describes an approach to generate hierarchies from text by using, as a sort of prompter, a pre-constructed ontology. This algorithm does not obtain an extension of the pre-existent ontology (as it happens with bootstrapping methods), but a new and independent one, not necessarily containing all the entities and relationships of the original one.

Ontology Learning Frameworks

In this section we describe other frameworks which, like ours, aim at modelling the whole ontology learning process, embedding different algorithms and possibly allowing researchers to easily add new ones.
The most relevant work in this field is the one performed at the University of Karlsruhe, Germany. Maedche & Staab (2001) introduce both a theoretical model for the ontology learning process (which is organized as a cycle) and a framework, called Text-To-Onto, organized in different modules whose

purposes range from system management to the extraction of knowledge with an NLP system and a tool for ontology editing. This work is later refined in Maedche & Staab’s KAON Text-To-Onto (2004), organized in four main modules: the *ontology management* component, the *resource processing* component, the *algorithm library* component, and the *coordination* component. A further evolution of this framework can be seen in Text2Onto (Cimiano & Völker, 2005), where learned ontological structures are represented as modelling primitives, making them independent from any specific knowledge representation language. Moreover, *Probabilistic Ontology Models* (POM) are applied to attach probability values to system’s results (a very useful feedback for ontology managers) and *data-driven change discovery* is introduced to detect changes in the corpus, calculating POM differences with respect to the changes, without the need of recalculating them for the whole collection of documents.

Other works, even if not aimed at providing a framework as general and complete as the ones in the Text-To-Onto family, present modules that might be interesting to be employed in such a kind of system. El Sayed et al. (2008), for instance, introduce a *relevance feedback* module into their tool. Keyword-based queries are expanded with their related terms by means of the synonymy and hypernymy relationships. Then, user interactions with the system are taken into account to update the taxonomy by means of a relevance feedback mechanism.

A MODULAR FRAMEWORK TO LEARN SEED ONTOLOGIES FROM TEXT

Heuristics for the automatic extraction of ontologies from text often return inaccurate results that need to be validated by users. Actually, no single solution is always better or worse than the others, rather the performance of each heuristic can be influenced by different factors: for instance, the number of documents taken into account, their length, or their language. As a result, there is no “one size fits all” solution that could be efficiently applied in this field. Learning which solution might be better for a specific problem is an interesting research question per se, and having a system which allows to test different heuristics on the same data might prove useful both for the comparison of results and for the generation of the final ontology.

Another point in favour of a modular approach is the fact that the ontology extraction process can be usually split into different steps which can be individually tweaked to obtain different results. And, since making two people agree on an ontology defined by only one of them is already a difficult task, we cannot expect a machine to get the right solution out of the box, without allowing an ontology manager to check its partial results and possibly refine them. Possible reasons for user intervention are, for instance, *simplification* (i.e., the number of extracted terms/concepts needs to be reduced), *customization* (i.e., some concepts are missing and need to be added, or some equivalences between concepts need to be defined), and *integration* (i.e., the ontology manager might need to work on an existing ontology, so she has to provide one as an input to the system).

As a direct consequence of these considerations, the objective of this work was to provide a modular framework for ontology extraction from text. The main requirements for this framework were the following two: (1) allowing users to easily plug their own metrics in the system and test them; (2) organizing the ontology extraction process as a sequence of steps, each one with well-defined inputs and outputs, and allow the process to be interrupted and its results to be inspected at each different stage. To accomplish this goal we designed our system to support a specific multi-step process, having at each step a well-defined set of inputs and outputs (see Figure 1). All the different stages of the process are accessible as methods of a container “Extraction” object, which can be easily embedded within another application. The description of each stage follows:

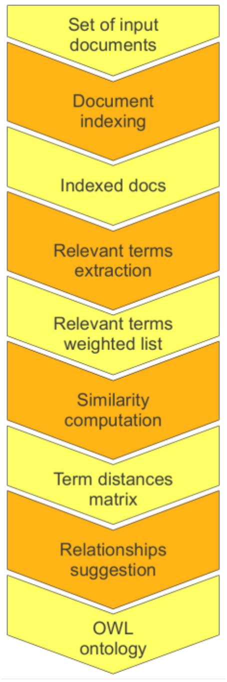


Figure 1: The four stages of the Extraction process, and the data they exchange with each other.

1. **Document indexing**. This step transforms a generic set of documents in a quickly accessible document index. The only constraint we put was that the documents had to be located in the same place, for instance inside a directory in the file system. As most of available document indexers (and in particular Lucene, the one we chose for our implementation) require input documents to be in plain text format, the document indexing module should also take care of the detection and conversion of different file types to text. The output produced at this step is a document index. To allow ontology managers to verify the results of the indexing process (i.e., if words in documents are correctly stemmed, or if terms that should be added to the stopwords list appear in the index instead), the index should be easy to open and inspect even using third-party applications. Moreover, the indexing process tends to be very expensive in terms of time. For this reason, this module should not only be able to make the index available to the following steps in the pipeline, but also to save it to disk in a standard (or, at least, widely known) format.
2. **Relevant terms extraction**. The expected output for this step is a weighted list of terms, appearing in the corpus of documents and considered relevant according to a chosen algorithm. Different implementations of these algorithms could be plugged in, provided they all receive as their input the document index generated in the previous step and return, as output, the weighted list of terms. While term relevance calculation can be considered as the main operation performed at this stage, the management of its output is nevertheless very important: as we want to allow users to inspect the tool's results at each step, it is essential to provide this information so that it can be used in different ways. In our case (see next section for details), we chose to return relevant terms as simple text for direct inspection, as RDF/OWL to be opened with an ontology editor, and as JSON to be easily consumed by external applications.
3. **Similarity computation**. Most of the approaches we have studied for the automatic detection of subsumption relationships rely on some measures of similarity (or, conversely, distance) between terms. We have designed a specific module for this, whose inputs are both the document index and the list of relevant terms. The reason for this double input is that, on the one hand, term similarities are related to the number of their occurrences in one or more documents, while on the other hand we are interested in analyzing only the subset of terms which have been chosen as the

most relevant ones. Again, different approaches to the calculation of similarities can be implemented and added to the system. The expected output is a matrix of distances between terms.

4. **Relationships suggestion**. In this last step the different algorithms for the detection of relationships are implemented. Even if most of these algorithms are aimed at detecting subsumption relationships, we preferred to talk here about generic ones, leaving the possibility for other types of algorithms (such as the "action/affection" we will introduce in the next section) to be plugged in. The main input is the matrix of distances between terms returned by the previous step. However, as we wanted the system not only to create new ontologies but also to enrich the already existing ones, it is also possible to provide an ontology as an additional input. The output of this step is a new or enriched ontology, containing as concepts the top relevant terms extracted in step 2 (and possibly the ones already present in the input ontology), and as relationships between concepts the ones identified in this very step. Input and output ontologies should follow the OWL (Web Ontology Language, see http://www.w3.org/TR/owl-ref/) format, making it possible to open them with an ontology editor such as Protégé (http://protege.stanford.edu/).

According to this design, at the end of each step the user can choose to interrupt the process and inspect the generated data. However, it is important to note that this interruption, though useful, should not be compulsory. In the implementation phase, it is important to allow this process (or some parts of it) to be also completed automatically. According to the task the user has to perform (i.e., create a new ontology or enrich an existing one) and the quality/time constraints she has, it will then be possible to choose which parts will be executed automatically and which ones will instead be supervised by a human.

Finally, we are aware that this process might not be common to all ontology learning algorithms, so it should be possible to either skip some steps or add new ones. For this reason, we tried to keep the framework as open as possible, relying on open standards and tools at the different stages. For instance, the indexing process can be performed separately, provided that the final index is saved in a format which is compatible with Lucene. As another example, different modules for relationship suggestion have already been developed, and not all of them require the similarity computation step to be performed.

THE *EXTRACTION* TOOL

In this section we describe the implementation of our framework. The general architecture of our system, called *Extraction*, closely follows its design (see Figure 2).

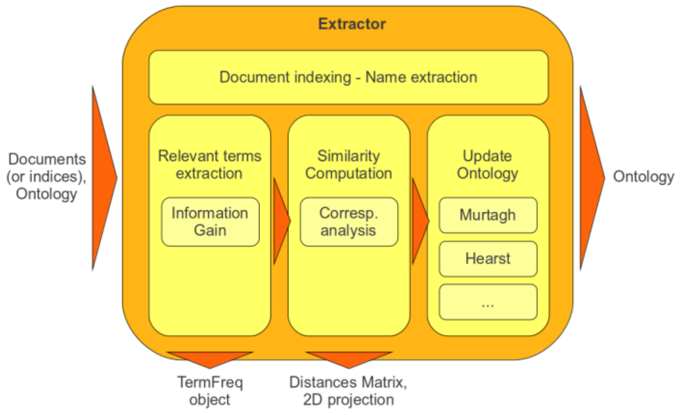

Extractor
Document indexing - Name extraction
Documents (or indices), Ontology
Relevant terms extraction
Information Gain
Similarity Computation
Corresp. analysis
Update Ontology
Murtagh
Hearst
...
Ontology
TermFreq object
Distances Matrix, 2D projection

Figure 2: Extraction*'s architecture.*

Extraction has been developed as a Java application and its code is available online at http://davide.eynard.it/SAOD/. The whole system is accessible through one container class, called *Extractor*. This class manages input data (document corpora, document indexes, and, optionally, ontologies to be enriched) and makes them available to the other modules. As the other modules require documents to be indexed, the Extractor class also takes care of document indexing. At the end of the process, an ontology containing the most relevant terms extracted from the document collection and the relationships found between them is returned.

The following subsections describe in detail the choices we made during the development of the different system components: indexing and data filtering, relevant term identification, similarity computation, and relationship suggestion (divided into suggestion of subsumption relationships and action/affection identification). In each subsection, when a specific algorithm is used, we also describe how it works and how we implemented it.

Indexing and data filtering

The indexing and data filtering process is taken care of by the main Extractor class. Documents are indexed using Apache Lucene (http://lucene.apache.org), which takes care of tokenization, indexing, and stopwords filtering. Different applications are then available to visually inspect or automatically analyze the indexes generated by Lucene (i.e. the ones available at http://wiki.apache.org/lucene-java/CommunityContributions).

The process taking place in this phase can be divided into different sub-steps: first of all, documents which have not been saved as plain text are converted into this format; then, Lucene performs the document analysis, splitting texts into tokens and applying different transformations on them; finally, the index is built and optionally saved to disk.

Document conversion has been inspired by the architecture described in Gospodnetic et Al. (2005): different document handlers have been developed, each one for a specific format; a file indexer class checks a file extension and then chooses the appropriate document handler; the mapping between extensions and handlers classes is done in a plain text file, therefore at any time it is possible to create a handler for a new type of document and upgrade the system just by updating the text file. Currently we have handlers for:

- **plain text** files;
- **HTML** files, which are parsed using CyberNeko (http://nekohtml.sourceforge.net/);
- **Word** documents, converted with Apache POI (http://poi.apache.org/), the Java API for Microsoft documents;
- **PDF** documents, converted with Apache PDFBox (with http://pdfbox.apache.org/);
- **RTF** files, using the Swing class RTFEditorKit.

During the analysis phase, the text is split into tokens and Lucene performs a number of optional operations on them. Some examples are the elimination of stopwords, lowercasing, and stemming. As stemming is language-specific, different analyzers exist according to the language that needs to be parsed: some examples are the GermanAnalyzer, the RussianAnalyzer, and the SnowballAnalyzer, which implements a whole family of stemmers and is able to support many European languages. For simplicity, we decided to restrict our system to the English language in this phase, and we just used Lucene’s default StandardAnalyzer to perform document analysis.

At this stage, additional custom filters can be applied to the single tokens: for instance, as concepts are typically names, we are interested in applying a name detector to automatically discard non-name terms before indexing. Extraction provides two such filters: the former is based on Wordnet, using the JWNL library (http://jwordnet.sf.net), while the latter uses the OpenNLP natural language processing library (http://opennlp.sf.net) to detect terms. Another filter we developed discards all the terms which are composed by less than *n* letters (usually, very short terms rarely represent relevant concepts, even if there are some exceptions such as acronyms). Applying these filters is optional, however it improves the results of the relevant term identification step by removing in advance all those terms which are likely to represent not relevant, if not invalid, concepts.

After the analysis has been performed, the converted terms which survive filtering are saved inside an inverted index, which relates words to all the documents they appear in. The indexing process is usually very time-consuming, so users are allowed to save the generated indexes and reuse them through different executions.

Relevant term identification

To extract relevant terms from text, we chose to calculate the Information Gain (IG hereafter) of indexed terms with respect to the common knowledge represented by a training set. Even if the calculation of IG is not the most common approach to detect relevant terms from a set of documents, it has an advantage over other techniques (i.e. TF-IDF) which only rely on term frequencies: the training set is used as a reference corpus and, to be considered relevant, a term needs to have, at the same time, many occurrences in the test set and few in the reference set. Thus, an accurate choice of both can provide better results with respect to classic approaches based on a single corpus. The training set might contain generic documents to automatically detect terms in the test set that are domain-specific (i.e. a collection of news articles vs. documents related to Semantic Web). Dually, the training set might be built with domain-specific documents to only extract those terms which are related to another domain (i.e. a set of Wikipedia-related pages can be used to automatically discard words belonging to this domain –for an application of this example see the Evaluation section). For every term appearing in the test corpus the score of Information Gain is computed. At the end of the process, terms are ordered by their IG and the top *n* ones are taken as potentially relevant concepts.

Information Gain measure is based on the concept of entropy. In our case, we want to calculate how the total entropy changes for the two training and test corpora by knowing that a specific term is more or less common within these sets. We can identify the *p(i)* function as the probability for a document to appear in a corpus *i*, computed as the number of the documents in *i* divided by the total number of documents:

$$p(i) = \frac{documentsIn(i)}{totalDocuments}$$

Here *i* can be either the training or the test set. Entropy for a group of documents D_g distributed in the two corpora is calculated as:

$$H_{D_g} = -\sum_i p(i) \cdot log(p(i))$$

We now identify three different groups of documents:

- the group of all documents, D_{total};
- documents presenting the term *t* in them, D_t;
- and documents not presenting the term *t* in them, $D_{\neg t}$.

The Information Gain score of term *t* is then calculated as:

$$IG(t) = H_{D_{total}} - \frac{|D_t|}{|D_{total}|} H_{D_t} - \frac{|D_{\neg t}|}{|D_{total}|} H_{D_{\neg t}}$$

Information Gain has low values for terms that are very common in both training and test corpora. Terms that, instead, are very common in just one of the two corpora have a high Information Gain. In this context, the training corpus has the role of a reference for the test corpus: terms that are frequent in test corpus can both be characterizing terms of the corpus, or very common and uninteresting terms. If a term *t* is also very frequent in the training corpus, which refers to a topic different from the test corpus, then it is supposed to be a useless term, and in fact it will receive a lower IG value. Vice versa, if *t* is very frequent in the test corpus, but not in the training one, it is supposed to be a characterizing term of the test corpus, thus it will receive a higher IG value.

At the end of this process, a set of *n* (with *n* specified by the user) relevant terms is collected, as the top *n* terms ranked by their IG. The weighted list of terms is contained within a *TermFreq* object, which is basically an extension of a HashMap object that can be serialized in different formats. The formats we implemented are:

- **simple text**, for direct inspection during debugging;
- **JSON**, for an easy access within Web-based frontends;
- and a customizable **OWL/RDF ontology**, based on a template whose T-Box defines the vocabulary and whose A-Box is created at serialization time.

Similarity computation

This step can be seen as propaedeutical to the extraction of relationships between words. In our case, the approach of Correspondence Analysis (Murtagh, 2005) is used to project the relevant terms in a *k*-dimensional euclidean space according to their distributional behaviour over the documents. The closer two terms are in this space (in Euclidean sense), the more their distributional behaviour over the documents of the corpus is similar. The more the distributional behaviour is similar, the more we are allowed to infer a semantic similarity between the terms.

Correspondence Analysis starts from a *(test documents)*×*(relevant terms)* matrix, where the cell $n_{i,j}$ holds the number of occurrences of the *j*-th term in the *i*-th document, as, for example:

	Caligula	**city**	**group**
D1	60	12	60
D2	20	54	5
D3	32	3	2
D4	1	2	5

Table 1: Observations of terms across documents.

In order to create the Euclidean space, Correspondence Analysis applies different matrix transformations. The first step is to calculate the grand total of the individual observations and divide the value of every cell by this grand total. This is done in order to have a matrix M_{prob} expressing, in each cell $n_{i,j}$, the co-occurrence probability of the two (*i, j*) modalities (i.e., documents and terms). In this simple example, the grand total is 376, and the probability matrix becomes like the one shown in Table 2. Here one more column and one more row appear: they represent the *marginal distributions* of the two modalities, which are calculated in the following step. For every row *i*, the marginal distribution F_i is the sum of all the values appearing in that row; for every column *j*, the marginal distribution F_j is the sum of all the values appearing in that column.

M_{prob}	**caligula**	**city**	**group**	F_i
D1	0.159	0.031	0.159	35%
D2	0.053	0.143	0.013	21%
D3	0.085	0.007	0.005	10%
D4	0.002	0.005	0.332	34%
F_j	30%	19%	51%	

Table 2: The probability matrix with marginal distributions F_i *and* F_j.

Being $\sum_{n=0}^{i} F_i = 1$ and $\sum_{n=0}^{j} F_j = 1$, we prefer to show F_i and F_j as percentages. Every percentage represents the contribution of the *i*-th or *j*-th modality to the total of the occurrences. Now, let us consider the columns of our M_{prob} matrix: the algorithm divides every $n_{i,j}$ probability by the F_j value. What we obtain is the matrix whose columns are called *column profiles*:

$M_{colprof}$	**caligula**	**city**	**group**	F_i
D1	53%	17%	31%	35%
D2	18%	76%	3%	21%
D3	28%	4%	1%	10%
D4	1%	3%	65%	34%

Table 3: The column profile matrix.

Column profiles are an important element of Correspondence Analysis because they represent the pure distributional behaviour of column modalities (the terms, in our case), independently from the original amount of occurrences we started with.

The last column F_i represents the average behaviour of the different column profiles. The divergences of the single column profiles from this average profile can be evaluated with the χ^2 test of independence, and the χ^2 distance between two column profiles *l* and *k* can be computed as

$$\chi^2(l,k) = \sqrt{\sum_j \frac{(n_{l,j} - n_{k,j})^2}{F_j}}$$

The sum of all the χ^2 tests applied to all column profiles with respect to the average profile represents the total inertia of the matrix with respect to its columns. Inertia represents the total amount of divergence of the column profiles from the assumption of independence. The higher this number, the higher the probability of an interdependence between rows and columns. There is also something more to say about the χ^2 distance between two different column profiles. The obtained value represents how much two different rows differ in their distributional behaviour: the more similar this behaviour, the more there should be a similarity between the entities represented by the two columns (in our case these entities are the terms appearing in the documents). What Correspondence Analysis does, starting from the column profiles matrix, is to provide a compact representation of the similarities between the column modalities. In order to do so we project them into an Euclidean space of *k* dimensions, where *k < min(i-1, j-1).* This space has the following properties:

- the Euclidean distance between the column modalities in this space of representation is exactly equal to the χ^2 distance between their column profiles. Calling the α-th dimension of the j-th column $F_\alpha(j)$ we can state:

$$\chi^2(l,k) = \sqrt{\sum_i \frac{(n_{i,l} - n_{i,k})^2}{F_i}} = \sqrt{\sum_\alpha (F_\alpha(l) - F_\alpha(k))^2}$$

- the origin of the axes is placed in the barycentre of the different column profiles with respect to the χ^2 distance measure, that is, as we said, the average column profile;
- axes are selected to have along them, in decreasing order from the first to the k-th, the maximum possible variance of the projected elements.

The creation of the space is done with the application of different operations over the column profiles matrix. The most important between them is the eigenvalue decomposition applied to translate the cloud of modalities into a different coordinate system, whose axes follow the directions of maximum variance of the points. Knowing that the axes are ordered according to the variance of the elements, we can reduce the complexity of the representation by discarding as many dimensions as we want, starting from the last one. In this way, we know that we always keep the dimensions with the highest variance, that is the dimensions carrying the higher amount of distributional information. As an example, the plot in Figure 3 was obtained just keeping the two main dimensions in the k-dimensional space.

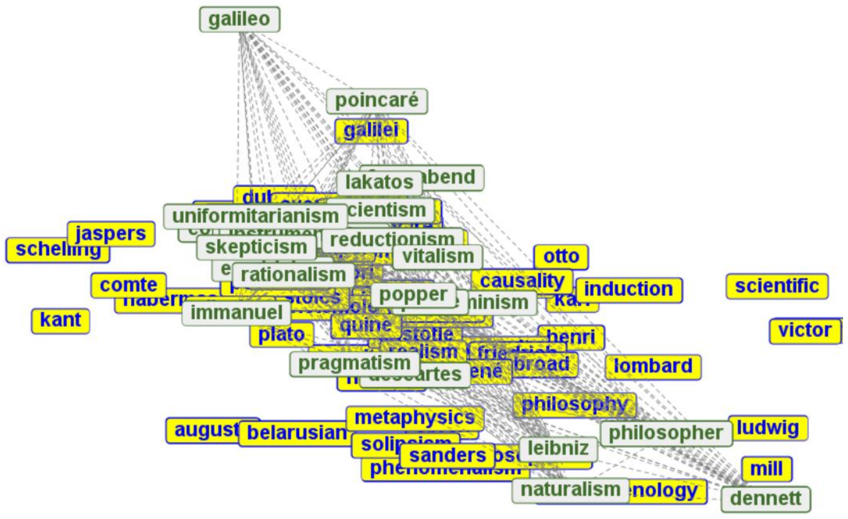


Figure 3: Philosophers and philosophical movements plotted in a 2D Euclidean space.

Subsumption Relationships

Subsumption relationships between the concepts identified in the previous steps are built using three different algorithms: hierarchy generation from hierarchical clustering representation (as in Murtagh's approach), search for Hearst Patterns on the Web, and bootstrapping.

Murtagh's Algorithm

This approach applies the technique presented by Fionn Murtagh (Murtagh, 2007) for the generation of a hierarchy of concepts from a hierarchical clustering representation. Having used the Correspondence Analysis technique, adopted in many works by Murtagh, it was natural for us to experiment with his solution for the generation of hierarchies of concepts, although we obtained better results with other techniques.

A Hierarchical Clustering tree (or *dendrogram)* is created according to term proximity in the Euclidean space. Murtagh's algorithm starts from this representation to build the concept hierarchy. As a first step, it transforms the dendrogram into its *canonical representation*, where later agglomerations are always represented by right child nodes (see Figure 4). Starting from the bottom-left cluster (i.e. the pair of nearest terms), then, the right sibling in the tree is always considered as a hypernym of the left one. In our example, for instance, *emperor* would be considered as a hypernym of *caligula* and *pompey*, and *troop* as a hypernym of *soldier* and *chevalry*. Finally, the cluster dominated by *troop*, being the right sibling of the other one, would also be its dominator. This final representation is returned as the searched hierarchy.

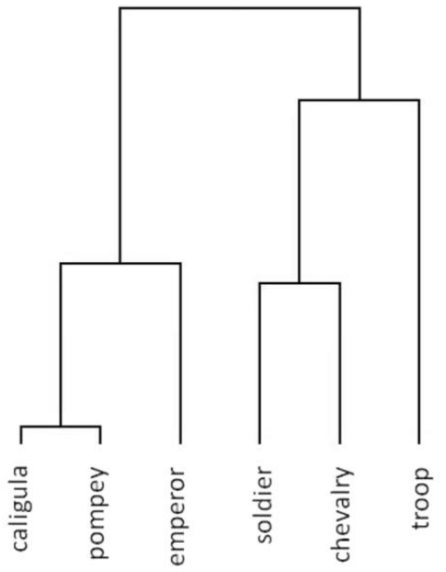


Figure 4: Result of hierarchical clustering over a set of terms from a corpus about Roman empire.

Hearst Patterns on Web

The Hearst Patterns on Web approach can be applied both for the creation of a hierarchy from scratch or to expand an existing one. For the implementation of the algorithm we took inspiration from PANKOW and applied the general principles of Learning by Googling, pluralizing Hearst patterns' classes as in (Cimiano, 2004). The algorithm starts either with a hierarchy to be expanded (bootstrapping), or with an empty one. In the latter case, the first term t_1 to be added is simply put under the root element. Then, for every new term t_n to be added, its likelihood to be a hyponym of any term t_i already in the hierarchy is checked as follows:

1. five different pre-defined strings based on Hearst patterns are built:
    - hp_1: pluralize(t_i) such as t_n
    - hp_2: pluralize(t_i) including t_n
    - hp_3: pluralize(t_i) especially t_n
    - hp_4: pluralize(t_i) like t_n
    - hp_5: t_n is a/an t_i
2. six Google queries are executed (the five Hearst patterns plus the single term t_n), obtaining the number of Google hits for each query;
3. the score of every string is defined as the ratio between the number of its Google hits and the number of Google hits of the hyponym when searched by itself:
    - $score_{hpi} = googleHits(hp_i)/googleHits(t_n)$

4. the total score is obtained as a sum of the five different scores.

Once the hypernym scores have been calculated for every t_i in the hierarchy, t_n is saved as a hyponym of the highest scoring t_i, provided that it exceeds a pre-defined threshold level. If no t_i hypernym scores higher than the threshold, the user can manually suggest one, discard the term or add it in the hierarchy as a hyponym of the root element. The algorithm can also be executed in an automatic way: in this case, it directly puts the t_n term as a hyponym of the root element. After t_n is placed, all its siblings are checked in the same way for a hyponymy relation with it; if the score is above the threshold, the sibling element is saved as a hyponym of t_n.

Maedche and Staab's Bootstrapping
Maedche and Staab's Bootstrapping Process is an implementation of the model defined in (Maedche et Al., 2003). In this model, a concept hierarchy is expanded adding a new t_n term according to the hypernyms of its *m* nearest neighbors in the *k*-dimensional space, where *m* is a parameter of the algorithm. The score for every *f* candidate hypernym is calculated as follows.
The Least Common Superconcept between two concepts *a* and *b* in a hierarchy is defined as:

$$lcs(a,b) = \underset{c}{\operatorname{argmin}} \quad \delta(a,c) + \delta(b,c) + \delta(root,c)$$

where *δ(a, b)* is the distance between *a* and *b* in terms of the number of edges which need to be traversed. Then the taxonomic similarity *σ* between two concepts in a hierarchy is defined as:

$$\sigma(a,b) = \frac{\delta(root,c)+1}{\delta(root,c)+\delta(a,c)+\delta(b,c)+1}$$

where *c = lcs(a, b)*. The score *W(f)* for a given candidate hypernym *f* is finally computed as:

$$W(f) = \sum_{h \in H(f)} sim(t_n, h) \cdot \sigma(n,h)$$

where t_n is the term to be classified and *H(f)* is the set of hyponyms of candidate hypernym *f* that are also nearest neighbors of t_n in the *k*-dimensional space. The *sim()* function is the similarity between two concepts as obtained from the *k*-dimensional space. If no hypernyms are found, the user can manually suggest one, discard the term or add it in the hierarchy as a hyponym of the root element. The algorithm can also be executed in an automatic way: in this case, it directly puts the t_n term as a hyponym of the root element.

Combination of Hearst Patterns on Web and Bootstrapping algorithms
This approach works by combining the two algorithms we previously described. The pre-existing hierarchy could be huge, and adding a new term t_n with the Hearst patterns algorithm would generate too many connections to the search engine servers. Depending on the Internet connection speed, the execution of a very large number of HTTP requests could be very time consuming. In this case, we look for the *n* nearest neighbors in the *k*-dimensional space of t_n in the pre-existing hierarchy, and collect all their ancestors. These ancestors are considered as the candidate hypernyms for t_n and they are evaluated according to the Hearst Patterns on Web algorithm.
As with the previous algorithms, if no hypernyms are found, the user can manually suggest one, discard the term or add it in the hierarchy as a hyponym of the root element. The algorithm can also be executed in an automatic way: in this case, it directly puts the t_n term as a hyponym of the root element.

Action/Affection identification

In 340 BC Aristotle wrote his “Categories”, a treatise whose purpose was to classify all the possible propositions about a being. This subject is still referential today for many works on logics. In the context of our work we are interested in learning from natural text knowledge about two of these categories of attributes:

- an *action* attribute for a concept, defined as the action expressed by the verb of the sentence where the concept appears as subject. Roughly said, it specifies what the subject can do;
- an *affection* attribute for a concept, defined as the action expressed by the verb of the sentence where the concept appears as object. Roughly said, it specifies what can be done with (or to) the object.

In this case we cannot only rely on distributional similarity, but we also use NLP algorithms over a corpus of documents in order to build a table of occurrences of nouns that do or undergo given actions. Correspondence Analysis can be used to project all the nouns and verbs in a Euclidean space of two or more dimensions (the more the dimensions, the more the precision of the information encoded by that representation). What we expect is that entities (nouns or verbs) that are more likely to be related are also close in this space. As an example, let us suppose that, as a result of document indexing, we create the matrix *M* shown in Figure 5 (top left). Following the steps of Correspondence Analysis algorithm, we can then calculate the matrix M_{prob} (Figure 5, top right), containing the probabilities for the different events, and its row and column profiles $M_{rowprof}$ and $M_{colprof}$ (Figure 5, bottom).

M	**eat**	**play**	**Shoot**
man	60	12	60
cat	20	54	5
chicken	32	3	2
gun	1	2	125

M_{prob}	**eat**	**play**	**Shoot**	F_i
man	0.159	0.031	0.159	35%
cat	0.053	0.143	0.013	21%
chicken	0.085	0.007	0.005	10%
gun	0.002	0.005	0.332	34%
F_j	30%	19%	51%	

$M_{rowprof}$	**eat**	**play**	**Shoot**
man	45%	10%	45%
cat	25%	68%	7%
chicken	85%	7%	5%
gun	1%	1%	98%
F_j	30%	19%	51%

$M_{colprof}$	**eat**	**play**	**Shoot**	F_i
man	53%	17%	31%	35%
cat	18%	76%	3%	21%
chicken	28%	4%	1%	10%
gun	1%	3%	65%	34%

Figure 5: The different steps of the CA algorithm using nouns and verbs related to them.

Figure 6 shows an example of 2-dimensional representation for the analyzed data. This representation leads to three different kind of interpretations:

- *semantic similarity between nouns*: the analysis of the distribution of nouns with respect to done or undergone actions can provide information about their semantic similarity. According to Aristotle’s Categories, the more two beings share the same attributes (in this case, action and affection are analyzed), the more they can be classified as similar. So, the more two nouns are seen as near in the plot, the more we expect them to be semantically similar;
- *semantic similarity between verbs*: as for the nouns, we expect that the more two verbs share a similar distribution with respect to the nouns, the more they can be considered semantically similar;

- *cross-distance between a noun and a verb*: in this representation the distance between a noun and a verb is significant in the sense that the more an attribute (action or affection) is seen to be referred to a noun in the corpus of documents, the more the noun is expected to be near that attribute in the Euclidean space.

The last point represents the most interesting and innovative kind of information that Correspondence Analysis is able to provide in this context: from a huge corpus of documents we can obtain an *n*-dimensional map of concepts, and of actions done and undergone by those concepts. This map can then be used to have a compact (and, if 2 or 3 dimensions are used, also graphical) representation of the likelihood of a concept to be involved in a given action. As an example, Figure 6 shows a 2-dimensional plot of the previously described entities. Looking at the picture we can conclude that a gun is much more likely to be involved in shooting than in eating, while a man can be equally involved in both; a chicken is often either subject or object of the “eat” verb, while a cat mostly plays (at least according to the data we provided).

As a last note, in our work we chose to extract only action and affection attributes, however other techniques could be applied to extract different types of attributes, with the possibility to build more complete knowledge repositories.

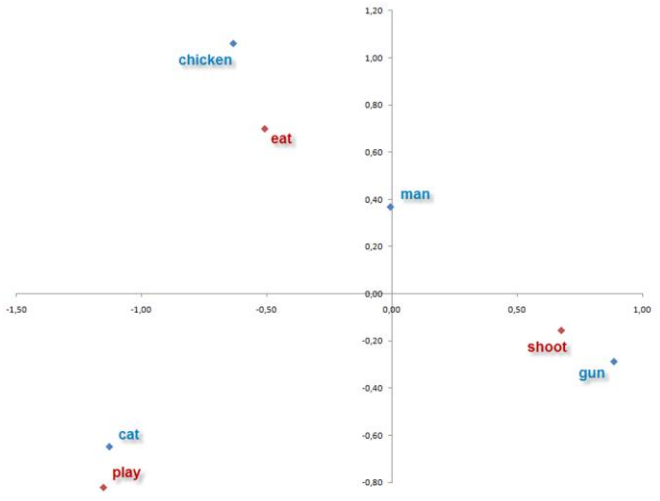


Figure 6: A 2-dimensional representation of the data shown in Figure 5.

QUANTITATIVE EVALUATION OF THE APPROACHES

Ontology evaluation, in the software developing process, is part of the quality assurance sub-process. Dellschaft and Staab (Dellschaft et al., 2008) suggest to evaluate ontologies in two different dimensions: a functional and a structural one. The functional dimension (according to which an ontology is generally required to be (1) consistent, (2) complete, (3) concise, and (4) expandable) tries to evaluate the quality and usability of an ontology *with respect to the software environment* in which the ontology is employed. The main limit of this dimension is that it does not give us a general and unambiguous method to evaluate ontologies, but is rather dependent on the specific use the ontology has been designed for.

The structural dimension fulfills this requirement, evaluating the ontology logical structure. This is the approach we took for this work, as we wanted to evaluate ontologies generated by different algorithms, without actually being interested in how they could be used in a given software environment. Structural evaluation of an ontology usually foresees two different measures of evaluation, *precision* and *recall*.

Precision is computed as the number of correct relationships found divided by the total number of the relationships in the ontology. Recall is the number of correct relationships found divided by the total number of correct relationships. While precision calculation can be performed without a reference ontology (i.e. it is possible to manually evaluate the correctness of the single subsumptions suggested by the tool), recall calculation requires one.

There are two main methods to decide whether a relationship is correct or not. The Manual Evaluation Method relies on a human domain expert who can personally judge whether a relationship is correct or not (precision), or if a relationship is missing in the ontology (recall). The second method is the "gold standard" based approach: a corpus of documents and a manually built ontology that is considered to be the correct representation of the concepts and the logical relations of the corpus are given. The corpus is provided as an input to the algorithm, and the generated ontology is compared to the original one.

It is easy to understand that the gold standard test provides us a general and unique measure which every algorithm can be evaluated against. However, as a limited number of gold standard ontologies exist, the testing of an ontology extraction algorithm with this method is constrained to a limited set of domains. As the algorithm should be able to work with any domain, manual evaluation is recommended too: for this reason, we chose to adopt the manual approach, at least for the evaluation of the computed hierarchies. For the experiments concerning relations different from subsumption, we have chosen other evaluation approaches that are described in detail in the related section.

Datasets details

For our evaluation three different document corpora have been selected:

- a set of 847 Wikipedia articles about Artificial Intelligence and related topics;
- a set of 1464 Wikipedia articles about Roman Empire and related historical articles;
- a set of 1364 Wikipedia articles about Biology.

As a referential Training corpus we have always used the same collection of 1414 Wikipedia articles about Wikipedia itself. This choice has been purposely made to exploit the IG measure for term relevance we previously described: using Wikipedia itself as a training corpus, some terms such as "Wikipedia", "Wikimedia", or "article", that have a high frequency in all the four document corpora, received a lower Information Gain score, becoming less relevant than other terms specific of each test corpus.

Subsumption relationships

We applied the Correspondence Analysis algorithm in order to generate a 2-dimensional representation of distributional similarity of the relevant terms, then we ran the previously described ontology learning algorithms and asked a human judge (with knowledge about ontologies and, not being an expert in all the three domains, allowed to search the Web when in doubt about a subsumption) to evaluate the generated hierarchies. As our document corpora contain hundreds of documents, the manual creation of reference ontologies for the three different domains we took into consideration was too expensive to be performed. Thus, while precision has been calculated for all the corpora, we performed the recall calculation only for one domain (the Roman Empire). We have built the reference ontology using the top 200 terms from the corpus and manually adding subsumption relationships. Then we have compared the generated ontology with the one we built, obtaining a low recall value (3.5%). This is somehow expected, as Hearst Patterns are characterized by having a high precision and low recall. In Table 4 the average precision measures of the ontologies are summarized. While Murtagh's algorithm does not seem to perform well, the research for Hearst patterns on the Web seems to be a good option for the generation of concept hierarchies. Its semi-automatic version provides nearly ready-to-go ontologies, producing hierarchies such as the one shown in Figure 7. Maedche' and Staab's Bootstrapping algorithm and its combination with Hearst Patterns on the Web were designed to expand large concept hierarchies, and, as we did not have one available, no valid attempts have been done to test them. Early tests with small ontologies showed that the combination of Maedche and Staab's algorithm with Hearst Patterns on the Web improves the precision of Maedche and Staab's algorithm by itself of about 10%. These results, however, should be considered only as preliminary, so their details are not shown here.

Algorithm	**AI**	**Rome**	**Biology**	**Average**
Murtagh	6.6%	5.22%	1.87%	**4.56%**
Hearst Patterns on Web	60.00%	75.00%	37.50%	**57.50%**
Hearst Patterns on Web (assisted)	90.00%	94.52%	85.07%	**89.86%**

Table 4: Precision measures obtained from the evaluation of the different ontology learning algorithms.

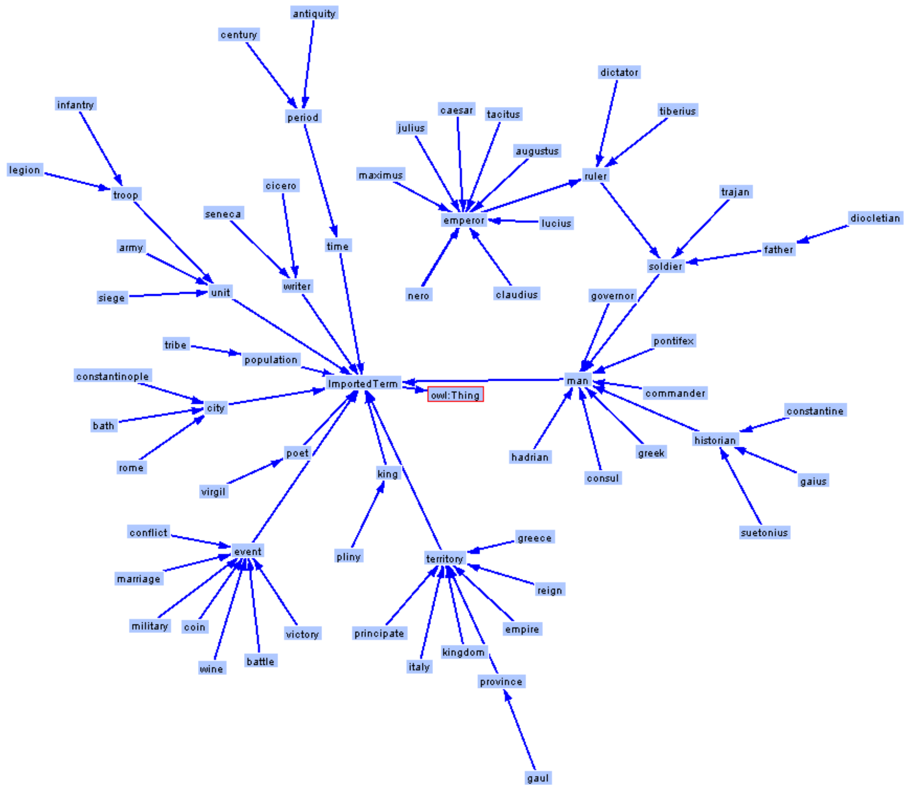


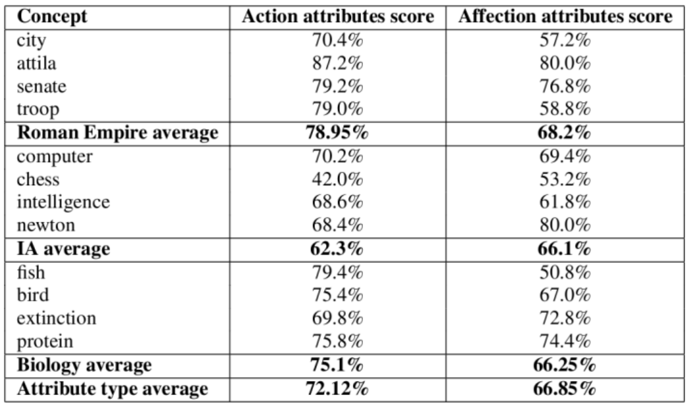
Figure 7: A hierarchy generated by the semi-automatic version of Hearst Patterns on the Web, with terms extracted from a corpus about the Roman Empire.

Action/Affection Identification

The evaluation of the action/affection extraction module has been performed with two different tests: evaluation of the 20 nearest attributes, and evaluation of the best concept extracted for a group of attributes.

20 nearest attributes

In this group of tests a set of concepts has been chosen from the ones extracted by the algorithm, and the 10 nearest action attributes and 10 nearest affection attributes have been identified for each one. The concepts for the test have been selected as follows: terms were extracted at random from the set of concepts, then a human expert evaluated whether they were actually important concepts for that domain of knowledge or not. If a term did not pass the test, another random one was selected and evaluated. Three independent human judges not involved in research evaluated whether the attributes found with our algorithm could represent typical actions that the term can do (action) or undergo (affection). The tests have been performed with a single-blind control procedure, with the human judge unaware of the origin of the data and of the purposes of the test: the judge had been only informed about the domain of knowledge of origin of names and attributes (necessary to better understand the semantics of the propositions). The judges graded each attribute with an integer value between 0 (impossible to match the attribute with the concept) and 5 (attribute typically matches the concept well). The ratio between the average of the three votes obtained and the maximum vote obtainable (5) has been considered as a partial score for a concept. Then, the final score for each concept has been computed as the average of its partial scores. The final scores obtained by the chosen concepts are summarized in Table 5. The total average score is 69.485%, which appears as a positive result: for affection attributes the scores are slightly lower, however we can state that the knowledge representation generated by the tool is able to recognize action and affection attributes of concepts with a confidence of about 2/3. Anyway, having no other referential works, we cannot state whether these results are good with respect to other approaches or not.



Concept	Action attributes score	Affection attributes score
city	70.4%	57.2%
attila	87.2%	80.0%
senate	79.2%	76.8%
troop	79.0%	58.8%
Roman Empire average	**78.95%**	**68.2%**
computer	70.2%	69.4%
chess	42.0%	53.2%
intelligence	68.6%	61.8%
newton	68.4%	80.0%
IA average	**62.3%**	**66.1%**
fish	79.4%	50.8%
bird	75.4%	67.0%
extinction	69.8%	72.8%
protein	75.8%	74.4%
Biology average	**75.1%**	**66.25%**
Attribute type average	**72.12%**	**66.85%**

Table 5: Summary of the average scores obtained from tests on extracted terms and their nearest attributes.

Best Concept from Attributes

This test has been designed as a sort of game: the user takes the role of an entity, and provides the tool some information about what this entity can do (action), or what can be done with it (affection). The tool, just based on what it extracted from the corpus of documents, tries to guess the entity. This is done by looking in the Euclidean space for the concept that minimizes the sum of the squares of the distances between it and all the attributes given as input. Thus, every term t_i is given a score calculated as:

$$score(t_i) = \frac{1}{\sum_n distance(t_i, a_n)^2}$$

The term with the highest score is then returned, together with the five top concepts and their scores. Some examples follow:

Corpus about Roman Empire:

```
User: I can persecute, I can burn, I can conquer, I can command, I can lead, I can
decide, I can kill, who am I?
Tool: You are pompey (5 top concepts found: pompey, score: 5.097; vespasian, score:
2.548; valentinian, score: 2.430; caesar, score: 2.115; sassanid, score: 1.839)
```

(Note: Pompey is the Roman general, do not mistake him for the Italian city. The user had Nero in mind, but all those attributes related to war influenced the answers to be more directed towards commanders).

Corpus about Artificial Intelligence:

```
User: I can think, I can evolve, I can play, I can grow, I can infer, who am I?
Tool: You are a neuron (5 top concepts found: neuron, score: 9.070; intelligence,
score: 2.731; predicate, score: 1.909; mechanic, score: 1.470; process, score: 1.469)
```

(Note: the user did not have a specific concept in mind, but just wanted to find the one which minimized the distance with all the others).

Corpus about Biology:

```
User: I can be killed, I can be observed, I can be fighted, I can be born, I can be
fed, who am I?
Tool: You are a sponge (5 top concepts found: sponge, score: 8.238; tree, score:
2.898; fish, score: 1.874; soil, score: 1.742; clade, score: 1.461)
```

(Note: here the user thought about a human, but, despite being informed about the domain, she did not choose the most proper attributes to describe a human in biological terms).

As the examples show, the evaluation returned both good and bad results. The tests have been performed with different numbers of action and affection attributes. These attributes have not been chosen at random, but we employed a human judge aware of the fact that the corpus would have dealt with concepts related with them.
In 30 conversations, providing 4-5-6 concepts, the algorithm showed about 35% of the times a valid concept for the given attributes as the first word, and about 60% of the times at least one valid concept in the top 5. Providing fewer attributes led to worse results, but it is understandable, because fewer attributes are not enough to characterize a specific entity, increasing the number of wrong answers.
The results are encouraging: this last interaction example with the user, in particular, is not a trivial game but it represents a well-founded rough draft of a Natural Language Understanding application: the tool here simulates an intelligent agent that in a short time can read thousands of documents and proves to be able to express learned semantics. It is thus understandable that this model could represent a good source of knowledge for complex semantic applications.

CONCLUSIONS AND FUTURE WORK

In this chapter we have described the Extraction system, a modular framework for the semi-automatic extraction of ontologies from natural text. The system design is based on a model of the ontology extraction process, with the purpose of allowing its users (i.e. ontology experts and programmers) to plug-in their own metrics and algorithms and to interrupt the process at any time to inspect the results of the tool analyses. The steps that form this process are document indexing, relevant term extraction, similarity computation and relationship suggestion. The system architecture strictly follows this design, and choices have been made in the implementation phase to (1) rely on open source solutions for the framework components which were not developed from scratch, and (2) provide intermediate results in standard formats which could easily be opened and inspected within the system or with third-party applications. Having a predetermined model of the ontology extraction process is of course a limit and a constraint, as not everyone follows the same steps to perform this activity. Thus, our effort has been directed towards easing the integration of our tool with others by using open standards when available and leaving users/developers the possibility of adding, skipping, or merging steps inside the process.

Algorithms for the extraction of the most relevant terms and their relationships from a document corpus have been developed and integrated within the system. The framework is versatile enough to allow us to compare the results of different algorithms for the creation of the hierarchy, both in an automatic and in a supervised execution. The final results showed us that the best-performing algorithm for the generation of a hierarchy of concepts (assisted Hearst Patterns on the Web) obtained a precision higher than 80%, which is quite satisfying if we consider it as a seed ontology which will then be polished by an ontology manager. As a limit, the tool still requires the document corpus to be in English. However, switching to a different language is possible as far as it is supported by Lucene (see http://lucene.apache.org/java/3_3_0/api/contrib-analyzers/index.html), OpenNLP (see http://opennlp.sourceforge.net/models/) and WordNet. The switch can be done in few steps: substitution of the current Lucene analyzer, modification of the language parameter passed to the OpenNLP library, and choice of the desired WordNet vocabulary. Finally, Hearst Patterns should also be updated to reflect the chosen language, but this is a really simple operation.

Extraction also provided us some useful and interesting byproducts. First of all, the possibility of plotting extracted terms in the 2D Euclidean space where they are projected (space which is also the same for documents), allowing users to actually see the semantic similarity between words in a document corpus. Another interesting example is the application of the same framework to identify relations between words different from subsumption, such as the Action/Affection relationships described earlier and others that could be inferred by the straightforward application of NLP techniques. We think these byproducts deserve more investigation and represent a good starting point for the (near) future work. More long-term plans, instead, include the evolution of the user interface (which is already present in a graphical format but is still not mature) and the integration of other algorithms, plus a new set of experiments to make it more easily comparable to other systems of its kind. Finally, as the tool grows in features, simple ontology extraction might easily move towards ontology evolution, thus requiring the integration of newly extracted pieces of knowledge within the existing knowledge base. For this reason, our architecture should be extended with an integration module, with features such as the ones provided by SOFIE (Suchanek, 2009).

REFERENCES

Alfonseca, E., & Manandhar, S. (2002). Extending a lexical ontology by a combination of distributional semantics signatures. In *13th International Conference on Knowledge Engineering and Knowledge Management. Lecture Notes in Computer Science, vol 2473* (pp. 1—7). Springer.

Alshawi, H. (1987). Processing dictionary definitions with phrasal pattern hierarchies. *Computational linguistics*, vol. 13 (pp. 195—202). MIT Press.

Amsler, R. (1981). A taxonomy for English nouns and verbs. *19th Annual Meeting of the Association for Computational Linguistics* (pp. 133—138). Stanford, CA.

Benzécri, J.-P. (1976). *L'Analyse des Donnes*. Paris, France: Dounod.

Benzécri, J. P. (1990). Programs for linguistic statistics based on merge sort of text files. In *Les Cahiers de l'Analyse des Données*, vol. XIV.

Calzolari, N. (1984). Detecting patterns in a lexical database. *10th International Conference on Computational Linguistics* (pp. 170—173). Stroudsburg, PA, USA: Association for Computational Linguistics.

Caraballo, S., & Charniak, E. (1998). New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics*, vol. 24 (pp. 275—298).

Caraballo, S. (1999). Automatic construction of a hypernym-labeled noun hierarchy from text. *37th Annual Meeting of the Association for Computational Linguistics* (pp. 120—126). Morristown, NJ, USA: Association for Computational Linguistics.

Cimiano, P. (2006). *Ontology Learning and Population from Text*: *Algorithms, Evaluations, and Applications*. New York: Springer.

Cimiano, P., Handschuh, S., & Staab, S. (2004). Towards the self-annotating web. *13th World Wide Web Conference* (pp. 462—471). New York, NY, USA: ACM.

Cimiano, P., & Völker, J. (2005). Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery. *10th International Conference on Applications of Natural Language to Information Systems (NLDB)* (pp. 227—238). Springer.

Dellschaft K., & Staab, S. (2008). Strategies for the evaluation of ontology learning. Bridging the Gap between Text and Knowledge. Amstedam, IOS Press.

Dolan W., Vanderwende L., R. S. (1993). Automatically deriving structured knowledge bases from online dictionaries. In Proceedings of the Pacific Association for Computational Linguistics.

El Sayed, A., Hacid, H., & Zighed, D. (2008). A new framework for taxonomy discovery from text. *12th Pacific-Asia conference on Advances in knowledge discovery and data mining* (pp. 985—991). Berlin, Heidelberg. Springer-Verlag.

Etzioni, O., Cafarella, M., Downey, D., Kok, S., Popescu, A., Shaked, T., Soderland, S., Weld, D., & Yates, A. (2004). Web-scale information extraction in knowitall: (preliminary results). 13th international conference on World Wide Web (pp. 100—110). New York, NY, USA: ACM.

Faure, D., Nédellec, C. (1998). A corpus-based conceptual clustering method for verb frames and ontology. *LREC Workshop on Adapting lexical and corpus resources to sublanguages and applications* (5—12). Granada, Spain.

Firth, J. R. (1957). A synopsis of linguistic theory 1930-55. *Studies in Linguistic Analysis* (special volume of the Philological Society), 1952-59 (pp. 1—32).

Gospodnetic, O., & Hatcher, E. (2005). *Lucene in Action*. Greenwich, US. Manning.

Grefenstette, G. (1994). *Explorations in Automatic Thesaurus Construction*. Kluwer.

Harris, Z. (1968). *Mathematical Structures of Language*. New York: Wiley.

Hearst, M. (1992). Automatic acquisition of hyponyms from large text corpora. *14th International Conference on Computational Linguistics* (pp. 539—545).

Hearst M., Schütze, H. (1996). Customizing a lexicon to better suit a computational task. *ACL SIGLEX Workshop on Lexical Acquisition*, Columbus, OH.

Maedche, A., Pekar, V., & Staab, S. (2003). *On discovering taxonomic relations from the web*. Technical report, Institute AIFB - University of Karlsruhe, Germany.

Maedche, A., & Staab, S. (2001). Ontology Learning for the Semantic Web. *IEEE Intelligent Systems* 16(2) (pp. 72—79).

Maedche, A., & Staab, S. (2004). Ontology Learning. *Handbook on Ontologies* (pp. 173—189). Springer.

Markert, K., Modjeska, N., & Nissim, M. (2003). Using the web for nominal anaphora resolution. EACL workshop on the Computational treatment of Anaphora (pp. 39—46).

Miller, G. (1995). Wordnet: A lexical database for English. *Communications of the ACM*, vol. 38 (pp. 39—41).

Murtagh, F. (2005). *Correspondence Analysis and Data Coding with Java and R*. Chapman & Hall.

Murtagh, F. (2007). *Ontology from hierarchical structure in text.* Technical report, University of London Egham.

Pasca, M. (2004). Acquisition of categorized named entities for web search. *13th ACM international conference on Information and knowledge management* (pp. 137—145). New York, NY, USA: ACM.

Ritter, A., Soderl, S., & Etzioni, O. (2009). What is this, anyway: Automatic hypernym discovery. *AAAI-09 Spring Symposium on Learning* (pp. 88—93).

Schutze, H. (1993). Word space. *Advances in Neural Information Processing Systems 5* (pp. 895—902). Morgan Kaufmann.

Suchanek, F. (2009). *Automated Construction and Growth of a Large Ontology*. Unpublished doctoral dissertation, Saarland University, Saarbrücken, Germany.