The Journal of Systems and Software 86 (2013) 1426–1452

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A novel semantic information retrieval system based on a three-level domain model

Licia Sbattella[a], Roberto Tedesco[b,*]

[a] Politecnico di Milano, Dipartimento di Elettronica e Informazione, Via Ponzio 34/5, 20133 Milan, Italy
[b] Politecnico di Milano, MultiChancePoliTeam, P.zza Leonardo da Vinci 32, 20133 Milan, Italy

ARTICLE INFO

Article history:
Received 12 July 2012
Received in revised form
30 November 2012
Accepted 11 January 2013
Available online 20 January 2013

Keywords:
HMM
MaxEnt
Ontology
WordNet
Semantic information retrieval
Word sense disambiguation

ABSTRACT

This paper presents a methodology and a prototype for extracting and indexing knowledge from natural language documents. The underlying domain model relies on a conceptual level (described by means of a domain ontology), which represents the domain knowledge, and a lexical level (based on WordNet), which represents the domain vocabulary. A stochastic model (the ME-2L-HMM2, which mixes – in a novel way – HMM and maximum entropy models) stores the mapping between such levels, taking into account the linguistic context of words. Not only does such a context contain the surrounding words; it also contains morphologic and syntactic information extracted using natural language processing tools. The stochastic model is then used, during the document indexing phase, to disambiguate word meanings. The semantic information retrieval engine we developed supports simple keyword-based queries, as well as natural language-based queries. The engine is also able to extend the domain knowledge, discovering new and relevant concepts to add to the domain model. The validation tests indicate that the system is able to disambiguate and extract concepts with good accuracy. A comparison between our prototype and a classic search engine shows that the proposed approach is effective in providing better accuracy.

1. Introduction

In the field of information extraction (IE), semantic information retrieval (SIR) systems aim to extend the classic information retrieval (IR) approach. They argue that pure frequency-based techniques cannot extract the true meaning of words and, thus, are not able to reconstruct the information being conveyed by a set of documents. This is due to several problems that arise from the intrinsic ambiguity of natural language constructs.

The first well-known problem is that words are often polysemic, that is the meaning they carry, in a given sentence, can be influenced by their syntactic roles, and by the relationships they hold with the other words in that sentence. For example, in the following three sentences: "photographers seemed to dog her every step", "a dog fox", and "the domestic dog", three different meanings of the word *dog* are used. Even the domain of interest can affect a word's meaning; for example, the word *pilot* will assume completely different meanings in the field of television programs and in the field of aviation.

Another well-known problem is that words can be synonyms, that is multiple words can carry the same meaning. Perfect synonyms are actually quite rare (think of the two sentences "that's my big sister" and "that's my large sister" wherein *big* and *large* are not synonyms), but in specific syntactic contexts and/or domains the difference in meaning could be negligible and the terms should be treated as equivalent (e.g., "Canada is quite a large/big country").

The process of solving the aforementioned issues, that is finding the right meaning for words in a sentence, is referred to as word sense disambiguation (WSD). Many methodologies exist, based on statistical or domain-modeling approaches (see Navigli, 2009 for a survey on WSD techniques).

The three-level-model we propose in this paper combines the best of both approaches. A domain model is used to describe the relevant concepts, while a stochastic model – that leverages linguistic information extracted through specialized tools – is used to disambiguate words and map them to the right concepts. The a priori knowledge provided by the domain model complements the stochastic model, allowing us to gather as much more information from the text. Moreover, our approach provides a natural way to disambiguate and augment user queries; for example, exploiting the whole/part relationship provided by the domain model, the query "mammalian body" can be used to find a document containing a description of a "dog's legs". Finally, our approach supports inferences on the domain model on the basis of new evidence found in the text, allowing us to add new knowledge to the model itself.

The main contribution of this work is twofold. One the one hand, we have defined a novel WSD approach, based on a hybrid

* Corresponding author. Tel.: +39 02 2399 3477; fax: +39 02 2399 3574.
E-mail addresses: licia.sbattella@polimi.it (L. Sbattella), roberto.tedesco@polimi.it (R. Tedesco).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.01.029

ontological-linguistic-stochastic model; on the other hand, we have designed a SIR engine, based on our model, that can extract and index relevant information from natural language documents, extend the domain model with new information that has been found in the documents, and support both keyword- and natural language-based queries to documents and to the domain model.

Since our main industrial partner was a wine company, we evaluated our approach in the context of a winery case study. We started by defining our own domain-specific ontology, and by preparing a dataset to train the stochastic part of our model; then, we tested the model's ability to map each word to its correct meaning, and obtained an accuracy of 0.83; finally, we compared our SIR engine to a pure TF-IDF engine, and discovered that our approach provides better rankings.

Both the methodology and the prototype were developed in the context of the ARTDECO [1] (Adaptive infRasTructures for DECentralized Organizations) project (Anastasi et al., 2012), as a test bed for investigating advanced knowledge extraction techniques. In the ARTDECO project, a group of partner companies shared knowledge and computational resources to act as a global entity (the so-called "networked enterprise"). In this scenario, the partners were in charge of feeding a common repository with textual documents gathered from heterogeneous sources. The repository provided advanced search functionalities (Montedoro et al., 2012), and was able to select the most relevant documents,.

The paper is structured as follows. Section 2 introduces relevant related approaches to the IE problem; Section 3 provides an overview of the domain model and of the SIR prototype; Section 4 details the domain model, which contains both the domain knowledge and the domain vocabulary; Section 5 illustrates the linguistic information we can extract from text; Section 6 introduces the stochastic model used to create a mapping between the domain knowledge and the domain vocabulary by taking into account linguistic information, and Section 7 explains how to train such a model; Section 8 describes the indexing process, which disambiguates word meanings by mapping them to the most probable concept in the domain knowledge; Section 9 details how the engine handles queries; Section 10 presents the domain knowledge extension mechanism; Section 11 illustrates the data that we gathered from the prototype, and the performance indexes that we obtained; finally, Section 12 draws a conclusion and presents future work.

2. Literature review

There are several ways to extract information from texts. The common goal of such methodologies is to automatically extract structured information from natural language documents. In general, IE implies three steps: the application of a WSD procedure; information extraction according to some predefined structure; and the application of some inference mechanism to extract information that is not explicitly asserted but is derivable (usually, exploiting some a priori domain information). Among these approaches, named entity recognition (NER) systems and ontology-based annotators are often used.

2.1. Stochastic approaches: NERs

NERs (Ratinov and Roth, 2009; Nadeau and Sekine, 2007) start from a collection of labels and, usually, a stochastic model that describes how these labels are related to words. At run-time, these systems are able to assign the right labels to the words, resolving ambiguities. As an example, the well-known Stanford NER (Finkel et al., 2005) is based on a factored model: a conditional random field defines the local-structure language model, while a second model takes into account long-distance language dependencies, usually present in natural language constructs; the two models are combined and an approximate decoding technique is used to obtain the most probable sequence of named-entity labels. NERs can also be used to augment queries; as an example, the system presented in Guo et al. (2009) focuses on applying NER techniques to queries (named entity recognition in query – NERQ); a weakly supervised latent dirichlet allocation (WS-LDA) stochastic model is used to assign the most probable named entity and *class* to the query (where a named entity can belong to many classes).

In both these systems, and in general in NERs, labels are not connected by any structure (i.e., they do not represent an ontology) and cannot be used for further elaborations such as reasoning or query extension. In contrast, in our approach each label attached to words is actually a concept that is part of an ontology; this way, reasoning and query extension are fully supported.

2.2. Domain-model approaches

Ontology-based annotators (Wimalasuriya and Dou, 2010) try to address NERs' shortcomings by leveraging a domain model provided by an ontology. Some systems leverage lexical resources as domain model descriptions; for example, the system described in Che-Yu and Hua-Yi (2010) suggests using WordNet – a lexical database – as a "vocabulary ontology", and exploits its structure to simplify the WSD procedure. This way, however, the approach does not allow for the definition of proper domain models, since WordNet is not an ontology, and the definitions it provides are general and not tailored to any specific domain. Other systems, like the one described in Bratus et al. (2009), present algorithms for ontology-guided entity disambiguation. In particular, the approach described in the paper makes use of an ontology in which nodes are tagged with names and descriptions (descriptions are composed of lexical units: words and stems); for each node of the ontology tree, the system defines a collection of features, derived from the lexical contents of the textual description associated with the node, its ancestors, its siblings, and additional lexical units attached to the nodes of the ontology tree. Sometimes, a linguistic model is paired to the domain model, as described in Artequakt (Alani et al., 2003). In Artequakt, however, the ontology and the linguistic models are handled by two different modules. Thus, the ontology can be used for inference but cannot help during the recognition phase.

In contrast to the aforementioned systems, our approach describes the domain *both* from the ontological *and* the lexical points of view, mapping them by means of a stochastic language model; this way, both descriptions are leveraged during the recognition phase. Moreover, we avoid mixing lexical and ontological levels, and instead emphasize the importance of separating these two information levels (see Section 4).

2.3. Query expansion

Several IR methodologies exist for searching extracted information; in particular, *query expansion* is often used (Bhogal et al., 2007). In Rinaldi (2009), queries consist of a list of terms to retrieve (so-called *subject keywords*) and of a domain of interest (the so-called *domain keyword*). Subject keywords are expanded by means of the related synsets found in WordNet, resulting in a semantic network. Similarity is based on a hybrid approach that considers both term frequency and semantic distance (calculated as path lengths in the semantic network). Meaning disambiguation is based on the so-called *centrality* of a term, which is related to its polysemy, yet it does not consider the actual lemma probability distributions (a *lemma* is the base-form of a word). In Aufaure et al. (2007),

[1] See http://www.artdeco.elet.polimi.it.

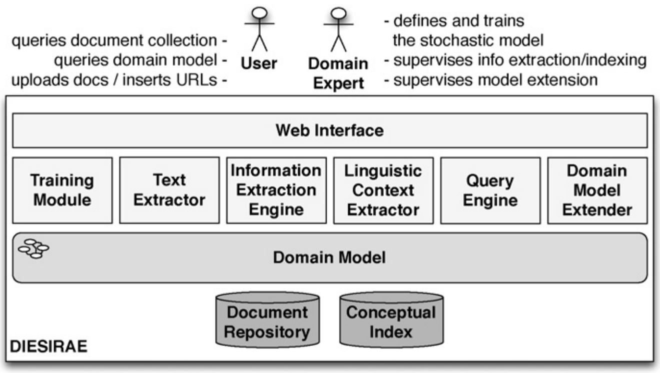

queries document collection -
queries domain model -
uploads docs / inserts URLs -
User
Domain Expert
- defines and trains the stochastic model
- supervises info extraction/indexing
- supervises model extension
Web Interface
Training Module
Text Extractor
Information Extraction Engine
Linguistic Context Extractor
Query Engine
Domain Model Extender
Domain Model
Document Repository
Conceptual Index
DIESIRAE

Fig. 1. The DIESIRAE main modules.

a lemmatization procedure is performed on the text, in order to obtain the part-of-speech of words; then, queries and documents are converted to vectors of concepts using a domain ontology and WordNet (to find the synonym, hypernym, hyponym having the same part-of-speech). Similarity is calculated using the well-known term frequency-inverse document frequency (TF-IDF) approach, but concepts are used instead of words. In this work, however, if an ambiguity arises, user intervention is needed to select the appropriate meaning.

A pure statistical approach to semantic similarity is given by latent semantic indexing (LSI). LSI intuition is that terms that tend to co-occur should be semantically correlated. LSI approximates the original term-document matrix using a limited number of orthogonal factors. These factors represent a set of abstract concepts, each conveying some idea common to a subset of the input collection. LSI is known to improve search recall; however, inferring semantic similarity from co-occurrence of terms – a pure statistical information – can be misleading, and tends to have a negative impact on precision (Kontostathis and Pottenger, 2006).

In contrast to the aforementioned systems, our approach applies the same WSD process used for the document set to the queries, solving the ambiguities. In particular, since we avoid LSI, the query expansion process does not cause the addition of spurious information to the query.

3. Approach and architecture: an overview

Fig. 1 describes the main modules of DIESIRAE (document-indexing engine for the semantic information-retrieval ARTDECO environment), the SIR prototype we developed (Sbattella and Tedesco, 2012). DIESIRAE interacts with two main human actors: the domain expert and the user. The domain expert is in charge of defining the domain's ontological and linguistic representation, of training the stochastic part of the domain model, and of supervising the information extraction, indexing, and model extension processes. The user inserts documents into the collection (by either uploading files or providing web page URLs to download), and issues queries that search both the document collection and the domain model.

DIESIRAE's structure is composed of two data modules, the domain module, several process modules, and the web interface module. Among the data modules, the document repository stores the text that is extracted from the documents, along with morphologic, lexical and syntactic information, while the conceptual index stores relevant information that are found within the documents (i.e., mappings between words that appear in the text and concepts that belong to the domain model).

The domain model describes the domain of interest at the lexical and conceptual levels of abstraction. Mappings between words and concepts are – in general – many-to-many, as words are often polysemic (i.e., they can carry several meanings), and since any concept can, in general, be lexicalized by means of different words. For this reason, a stochastic model – trained using the training module – defines how words map to concepts. The domain model is exploited by the process modules, to extract relevant information from the text.

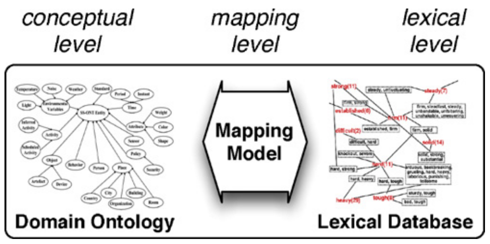


Fig. 2. The domain model internal structure.

The text extractor is a process module that is able to read several document formats, and to extract and clean plain text. The linguistic context extractor transforms each document from a sequence of words into a sequence of so-called *information items* [2]: $\langle \boldsymbol{I_0}, \boldsymbol{I_1}, \ldots, \rangle$, where each information item carries several pieces of data about a word. Information items allow us to take into account the *linguistic context* of the words, that is the information one can extract by reading the text. A word's simplest linguistic context, in a phrase, is made up of its surrounding words, but much more can be done by means of natural language processing (NLP) tools, which are able to extract morphologic, lexical and syntactic information.

The information extraction engine uses the domain model to extract relevant information from documents, perform the WSD procedure, and build the conceptual index. During the indexing process, a specific algorithm determines whether a given word carries a useful meaning, by taking into account its linguistic context. If it does, the algorithm selects the right concept to associate with such a word. Words that do not carry any useful meaning are discarded. At the end of this phase, each retained lemma L_i is associated with a concept C_i, and with a collection of synonym lemmas S_i. Moreover, the documents are represented by a list $\langle (L_0, S_0, C_0), (L_1, S_1, C_1), \ldots \rangle$. Such concepts and synonyms represent the actual information that is stored in the Conceptual Index, following the TF-IDF schema. This approach allows us to enhance search precision, since: (a) documents containing irrelevant words are likely to get a low rank; and (b) the words' meanings are disambiguated and the polysemy problem is solved. The query engine, which manages several different query typologies, allows the user to search both the document repository and the domain model. Finally, the domain model extender is able to discover and add new concepts to the domain model.

Sections 4–6 introduce the domain module's internal structure, explain how documents are represented, and define the stochastic model used to map the domain model to the lexical one.

4. Defining the domain model

Our model addresses the issues discussed in Section 2, providing a unified approach that combines a domain ontology and a stochastic model; the domain knowledge stored in the ontology will help the recognition phase, while the stochastic model will solve the linguistic ambiguities. We call this model the domain model (see Fig. 2).

[2] In this paper, symbols in bold contain multiple values, such as tensors, matrices, vectors, sets, or n-tuples.

The domain model is the foundation of our knowledge management strategy – it represents the specification of the domain being considered. The model represents the domain at two distinct levels of abstraction: the *conceptual level* and the *lexical level*.

The conceptual level is modeled by means of the domain ontology, which contains all the domain knowledge we need to represent. This level is language independent, since the domain ontology contains the *definition* of concepts, but does not consider any specific linguistic representation.

The lexical level provides the vocabulary – that is the words the system is able to recognize. In particular, we adopted a Lexical Database as a lexical-level model. The Lexical Database connects words using specific linguistic relationships, such as synonymy, antonymy, hyponymy, etc. These relationships allow us to navigate the vocabulary, and discover word similarities and meanings. This level allows us to *lexicalize* the concepts – by translating abstract definitions into concrete words. Changing the language supported by the system is a matter of providing the related lexical databases. The adoption of a multilingual lexical database (like MultiWordNet or EuroWordNet) could allow us to search multilingual text collections using the user's preferred language (a multilingual extension is planned for future versions of DIESIRAE).

A mapping level is needed to connect the conceptual and lexical levels, since words are usually polysemic and, depending on the linguistic context, can carry different meanings. Therefore it is, in general, impossible to map a given word to one and only one concept. Instead, the relationship between concepts and words is, in general, many-to-many – that is, a word can lexicalize many concepts, and a concept can be lexicalized by many words.

Thus, a *mapping level* should provide a model to represent such many-to-many mappings. Moreover, when indexing documents, a disambiguation procedure should select the right concept-word association, depending on the linguistic context of the sentence being analyzed.

Following the aforementioned ideas, the domain model is composed of three parts: the domain ontology, the lexical database, and the mapping model. We will now explain these components in more detail.

4.1. The domain ontology

An ontology is "an explicit specification of a shared conceptualization" (Gruber, 1993), a knowledge base that formally represents that concepts of a given domain. The domain ontology model we adopted is based on description logics (DL) (Baader et al., 2003), which subdivides concepts into three parts: TBox, which contains *classes*; ABox, which contains *individuals*; and RBox, which contains *relationships* among classes and *relationship instances* among individuals. In general, we refer to classes, individuals and relationships as *concepts*.

In the rest of this paper we will adopt a simple graphical representation of the domain ontology, where classes are depicted as rectangles, individuals are depicted as ovals, and relationships are represented by means of labeled solid arcs.

The upper part of Fig. 3 shows a small fragment of our domain ontology and specifies that a wine has a production region (which is part of a country), a color, and an appellation, while an appellation has a typical color; moreover, the wine named "Donnafugata Nero d'Avola" is produced in Sicily (which is an Italian region), is red, and belongs to Nero d'Avola wines, which are red wines.

Authoring ontologies is a challenging task, as methodologies and tools for ontology definition are still under active research (Kashyap et al., 2008). In addition, a good ontological representation of the domain is fundamental to the system's performance. Thus, this is probably the hardest step in building the domain model.

For our prototype we developed a wine domain ontology (in accordance with the testbed of the ARTDECO project). We used the Protégé (Knublauch et al., 2004) editor and the OWL (Smith et al., 2004) DL language.

4.2. The lexical database

Lexical databases define a graph structure, wherein words are connected by lexical relationships. WordNet (Fellbaum, 1998) is a lexical database for the English language. WordNet organizes words into four sections: nouns, adjectives, adverbs, and verbs.

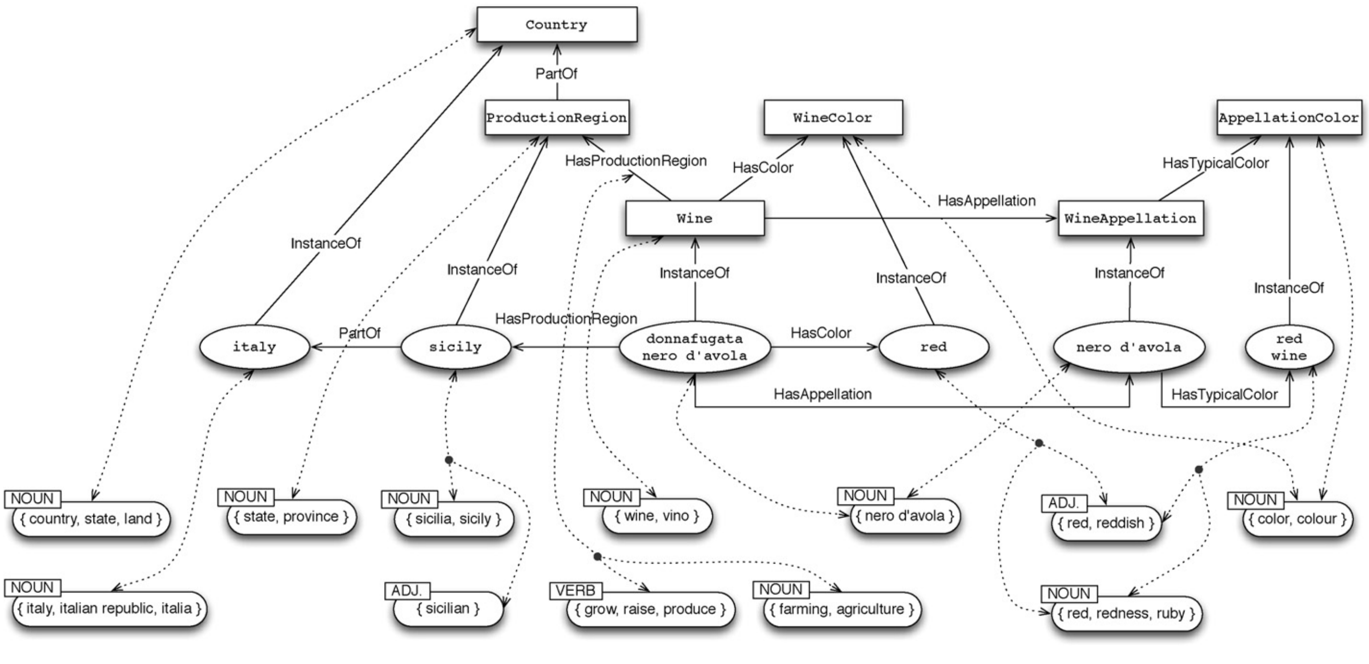


Fig. 3. Mappings between domain ontology and lexical database (WordNet relationships, and probability values we added, are omitted).

WordNet's building block is the *synset*. A synset is a set of synonym lemmas that carries a certain "meaning"; if a lemma is polysemic it will appear in many synsets. Synsets and words are connected to each other by means of lexical relationships, such as a *hypernymy* (i.e., the relationship between a given noun synset and another noun synset carrying a more general meaning; e.g., "red" and "color"), a *meronymy* (i.e., the whole/part relationship between two synsets; e.g., "italy" and "sicily"), a *derivationally related word* (i.e., the relationship between an adjective or a verb, and the noun it derives from; e.g., "sicilian" and "sicily"), etc. We adopted a simple graphical representation for WordNet, where synsets are represented by rectangles with rounded borders, and are tagged with the section that the synset belongs to. Each rectangle also contains the synonym lemmas (for simplicity we do not represent lexical relationships); see the lower part of Fig. 3.

Since the "wine" domain is highly specific, WordNet did not provide all the lemmas we needed (for example, the wine names); thus, we added a vocabulary extension, containing all the unknown words.

Notice that synsets are *not* assimilable to concepts, as they are language-dependent and do not carry domain-specific information. For example, `red wine`, which is an individual of the class `AppellationColor`, represents the concept "red as a wine color", and cannot be found in WordNet; instead, it could be *mapped* to one or more synsets whose lemmas carry a related meaning: the noun synset {red, redness}[*red color or pigment; the chromatic color resembling the hue of blood*], and the adjective synset {red, reddish, ruddy, blood-red, carmine, cerise, cherry, cherry-red, crimson, ruby, ruby-red, scarlet}[*of a color at the end of the color spectrum (next to orange); resembling the color of blood or cherries or tomatoes or rubies*]. This example emphasizes the fact that lexical databases, like WordNet, cannot be used to describe domain knowledge, and that a domain ontology is needed. The ontological representation and the lexical representations are then brought together by means of the mapping model.

4.3. The mapping model

At this point we have defined the domain of interest (i.e., the domain ontology TBox, RBox, and ABox), as well as the language we want to recognize (done by choosing WordNet as the lexical database). The third component we need is a mapping schema between the domain ontology and the lexical database. As previously stated, such a schema must take into account that words are in general polysemic, that synsets can lexicalize one or more concepts, and that concepts can be lexicalized by one or more synsets.

Fig. 3 depicts a possible mapping (the dotted arcs) between a small fragment of our wine domain ontology and the related lexical database synsets. Classes and individuals are mapped to synsets that carry a similar meaning; usually class and individuals map to nouns, adjectives, and adverbs. Class relationships are often associated with verbs (e.g., the relationship `HasProductionRegion` between `Wine` and `ProductionRegion` can be mapped to verbs like "to grow"), but also with nouns (the noun "farming"); notice that some relationships cannot be mapped, since the related synset would carry a meaning that is too general (e.g., the relationship `HasColor` could be mapped to the verb "to have"). Finally, notice that instance relationships (e.g., the `ProductionRegion` between `donnafugata nero d'avola` and `sicily`) "inherit" the mappings of their class relationships (for the sake of simplicity, the figure does not show such "inherited" mappings).

Fig. 3 shows the ambiguities one faces when defining the mapping schema. The first form of ambiguity arises from the presence of polysemic lemmas; for example, the lemma "state" carries the meanings of "country" and "region of a country". The second form of ambiguity is due to the many-to-many nature of the concepts-synsets mapping; for example, both the appellation `nero d'avola` and the wine `donnafugata nero d'avola` are lexicalized by the same synset (meaning we have a many-to-one relationship), while the region `sicily` is lexicalized by two synsets (meaning we have a one-to-many relationship). Finally, a many-to-many relationship is also present – the wine color `red` and the typical appellation color `red wine` are both mapped to the same two synsets.

For these reasons, the domain ontology-lexical database mapping is stochastic – we associate a probability value to each link connecting a concept to a synset; moreover, we do the same for each lemma-synset pairing. The resulting probability distribution is used by the mapping model, as Section 6.3 will explain.

The WSD procedure, during the indexing phase, will leverage the stochastic model to solve the ambiguities. The WSD procedure, performed by the information extraction engine, considers each word and tries to determine whether the word is meaningful, given the linguistic context and the concepts defined by the domain ontology. If the word is not useful, it is not indexed; if it is, the engine selects the right concept (class, individual, or relationship) among the possible candidates found in the mapping schema, connects the concept to the word being indexed, and calculates a probability value that represents a reliability measure for the meaning being associated to the word being indexed.

The mapping model will be presented in detail in Section 6.

5. Adding linguistic context information to the domain model

The WSD procedure tries to determine the words right meanings, given their linguistic contexts. To enrich the linguistic context, the linguistic context extractor module associates each word with information extracted by NLP tools. In particular, the text undergoes morphologic analysis, part-of-speech (POS) tagging, dependency parsing, and anaphora resolution (see Fig. 4).

Since words usually appear in inflected forms (plural forms, verb conjugations, etc.), morphologic analysis attempts to associate each word with all its possible base forms (the lemmas) and lexical categories (the POS's). For example, the word "drunk" can be lemmatized as "drink", if used as a verb, or "drunk", if used an a noun. Then, POS tagging analysis examines a word's linguistic context, selects the right part of speech and, as a consequence, the right lemma.

We adopted the Freeling morphologic analyzer (Atserias et al., 2006) as our morphologic & POS tagger analyzer. It returns all the possible base forms of a given word, with the related probabilities. Then, we exploited the output of the Stanford POS tagger (Toutanova et al., 2003) to select the right base form, providing the related POS tag (see Francis and Kucera, 1979 for a description of the Brown Corpus POS tags).

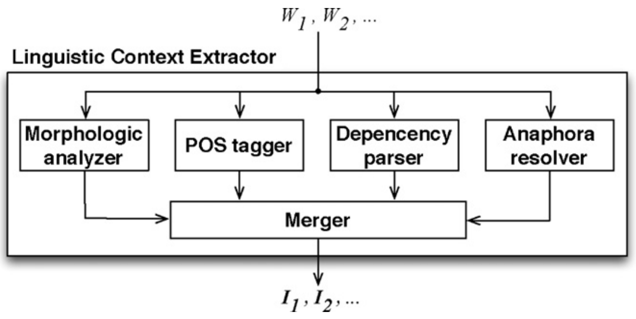


Fig. 4. Generation of lemma information items.

Inside a proposition, words take part in so-called *syntactic relationships*; for example, looking at the sentence "They elected John president", word "they" is the *subject* of the action "elected", while "John" is the *direct object* of the same action. In other words, one can define two relationships, such as `subj(they, elected)` and `dir-obj(John, elected)`. A dependency parser analyzes a sentence and extracts the list of syntactic relationships among words. We relied on the Stanford parser (Klein and Manning, 2003; de Marneffe and Manning, 2008) as our dependency parser.

Notice that, in the above example, the subject is a pronoun, which refers to the actual subject (which is actually found in a preceding sentence). This kind of reference, called *pronominal anaphora*, can be addressed by means of specific anaphora resolution tools. Such tools are able to link pronouns with the nouns they refer to. JavaRAP (Long Qiu and Chua, 2004) is the tool we adopted for this task.

Finally, each word is associated with the list of synsets it belongs to, the relationships that are connected to these synsets, the list of concepts it could be mapped on to, and the relationships connected to such concepts.

Thus, the plain text, which is composed of a sequence of words, is transformed into a sequence of *information items* $\boldsymbol{I}_\mathbf{t}$. An information item aggregates a word, its lemma, and all the knowledge extracted using the linguistic tools, lexical database, and domain ontology:

$$\boldsymbol{I}_\mathbf{t} \overset{\text{def}}{=} (W_t, L_t, POS_t, \boldsymbol{V}_\mathbf{t}^\mathbf{nlp}, \boldsymbol{S}_\mathbf{t}^\mathbf{eff}, \boldsymbol{V}_\mathbf{t}^\mathbf{syn}, \boldsymbol{C}_\mathbf{t}^\mathbf{eff}, \boldsymbol{V}_\mathbf{t}^\mathbf{onto}) \quad (1)$$

where W_t is the tth word extracted from the text, lemma L_t is its base-form, POS_t is its part of speech, $\boldsymbol{V}_\mathbf{t}^\mathbf{nlp}$ is a vector of syntactic relationships involving the lemma L_t, $\boldsymbol{S}_\mathbf{t}^\mathbf{eff}$ is the vector of synsets the lemma L_t belongs to, $\boldsymbol{V}_\mathbf{t}^\mathbf{syn}$ is the vector of lexical relationships involving synsets in $\boldsymbol{S}_\mathbf{t}^\mathbf{eff}$, $\boldsymbol{C}_\mathbf{t}^\mathbf{eff}$ is the vector of concepts the lemma L_t could be mapped to, $\boldsymbol{V}_\mathbf{t}^\mathbf{onto}$ is the vector of ontological relationships involving concepts in $\boldsymbol{C}_\mathbf{t}^\mathbf{eff}$.

Finally, the *linguistic context information* is the sequence $\boldsymbol{I} \overset{\text{def}}{=} \langle \boldsymbol{I}_\mathbf{t} : \mathbf{t} = 0, 1, \ldots \rangle$ that encodes a whole document.

6. Defining the ME-2L-HMM2 mapping model

After the linguistic context extraction phase, lemma information items $\boldsymbol{I}_\mathbf{t}$ represent information that the system can directly *observe* in the text. Concepts stored in the domain ontology, however, are *not observable*, since the information they represent is implicit – they express the meaning that the words intend to express, in their contexts.

Concepts do not appear in random order – some sequences of concepts are more likely to appear than others. The outcome of the decision process, regarding a given word, depends on the decisions that were already made for the preceding words.

To summarize, it seems that a viable model to represent word-concept mappings could be a hidden Markov model (HMM) (Rabiner and Juang, 1986), since it allows us to deal with observable and non-observable variables, and to implement the WSD procedure required by the indexing phase.

A HMM describes the probability distribution of an observable variable, given a hidden variable; this is called the *emission probability distribution* $p(observable|hidden)$. The model also describes how the past values of the hidden variable affect its current value; this is called the *transition probability distribution* $p(hidden_{current\_step}|hidden_{last\_step})$.

Our model is slightly more complex, as we have to map concepts to synsets, and synsets to lemmas (see Fig. 3); lemmas represent the observable variables, while synsets and concepts are both hidden. Our model considers the past last-two steps (i.e., it is a second-order HMM), instead of the usual single step, common in regular HMMs. This choice complicates the model but allows us to increase the accuracy of the prediction on the current hidden variable, since the model will take into account more information about the past (see Appendix F for a comparison of HMMs and second-order HMMs). Another peculiarity of our model is that HMM transition probabilities are described in terms of maximum entropy models. This allows us to take into account not only preceding words, but also *subsequent* words; moreover, such a model is also able to take into account the whole linguistic context information of the sentence being analyzed. As we will show in the following sections, all these extensions do not affect the good qualities of the HMMs, that is their efficiency and compactness.

A *decoding* algorithm (see Section 8.1) enables us, given a sequence of lemmas, to calculate the most probable sequence of concepts and synsets. Such an algorithm is exactly the WSD procedure we need, since each word will be mapped to the most probable concept (i.e., the most probable *meaning*), given the lexical context.

We will now present our model, define transition probability distributions (and show how they allow us to take into account the linguistic context information), define emission probability distributions, and discuss the training and decoding procedures.

6.1. The ME-2L-HMM2

Our model, called maximum entropy, two-level, second-order HMM (ME-2L-HMM2), is a second-order HMM with hidden transient nodes, whose output depends on both hidden levels; the model is defined as:

$$\mathcal{M} \overset{\text{def}}{=} (\Omega_C, \Omega_S, \Omega_L, \boldsymbol{A}^\mathbf{1}(\boldsymbol{I}), \boldsymbol{A}(\boldsymbol{I}), \boldsymbol{B}, \boldsymbol{D}, \boldsymbol{\Pi}) \quad (2)$$

where the finite sets $\boldsymbol{\Omega}$ represent:

$$\begin{aligned} \Omega_C &\overset{\text{def}}{=} \{C_i : i = 1, \ldots, N_c\} : \text{possiblehiddenconcepts} \\ \Omega_S &\overset{\text{def}}{=} \{S_i : i = 1, \ldots, N_s\} : \text{possiblehiddensynsets} \end{aligned} \quad (3)$$

$$\Omega_L \overset{\text{def}}{=} \{L_i : i = 1, \ldots, N_l\} : \text{possibleobservedlemmas} \quad (3)$$

The two-dimensional tensor field $\boldsymbol{A}^\mathbf{1}(\boldsymbol{I})$, built over $\boldsymbol{I}$, represents the transition probability distributions at $t = 1$; the three-dimensional tensor field $\boldsymbol{A}(\boldsymbol{I})$, built over $\boldsymbol{I}$, represents the transition probability distributions for $t > 1$; the matrix $\boldsymbol{B}$ represents the synset emission probability distribution; the three-dimensional tensor $\boldsymbol{D}$ represents the lemma emission probability distribution; and, finally, the vector $\boldsymbol{\Pi}$ represents the concept prior probability distribution:

$$\begin{aligned} \boldsymbol{A}^\mathbf{1}(\boldsymbol{I}) &\overset{\text{def}}{=} [a^1_{i,j}(\boldsymbol{I}) : a^1_{i,j}(\boldsymbol{I}) = p^\diamond(C_1 = c_j|C_0 = c_i, \boldsymbol{I})] \\ \boldsymbol{A}(\boldsymbol{I}) &\overset{\text{def}}{=} [a_{k,i,j}(\boldsymbol{I}) : a_{k,i,j}(\boldsymbol{I}) = p^\diamond(C_t = c_j|C_{t-1} = c_i, C_{t-2} = c_k, \boldsymbol{I})] \\ \boldsymbol{B} &\overset{\text{def}}{=} [b_{i,k} : b_{i,k} = p(S_t = s_k|C_t = c_i), \forall t \geq 0] \\ \boldsymbol{D} &\overset{\text{def}}{=} [d_{i,j,k} : d_{i,j,k} = p(L_t = l_k|C_t = c_i, S_t = s_j), \forall t \geq 0] \\ \boldsymbol{\Pi} &\overset{\text{def}}{=} [\pi_i : \pi_i = p(C_0 = c_i)] \end{aligned} \quad (4)$$

where random variables C_t, defined over $\boldsymbol{\Omega}_C$, represent concepts at position t (hidden state variable); random variables S_t, defined over Ω_S, represent synsets at position t (hidden intermediate variable); and random variables L_t, defined over $\boldsymbol{\Omega}_L$, represent lemmas observed at position t (observable variable).

The model makes several assumptions. The second-order version of the usual HMM *Markov assumption* asserts that the decision regarding the concept to associate with the t-th lemma depends solely on the decisions made regarding the last *two* concepts.

Moreover, the model relies on the usual HMM *stationary assumption*: the transition probability is independent of t. Because of this assumption, HMMs are able to model transitions in a very compact way, since just one probability distribution is needed. Section 6.2 will detail the transition probability distributions $p^{\diamond}(C_t|C_{t-1},\ \boldsymbol{I})$ and $p^{\diamond}(C_t|C_{t-1}, C_{t-2},\ \boldsymbol{I})$, and discuss our decision to include the linguistic context information $\boldsymbol{I}$ in this part of the HMM model.

HMMs also rely on the *output independence assumption*. In particular, the value of S_t, the hidden intermediate variable at time t, depends solely on the value of the hidden state C_t at the same time t. Moreover, the value of L_t, the observable variable at time t, depends solely on the value of the hidden variables S_t and C_t at that same time t.

Finally, just like in regular HMMs, our lemma emission and synset emission probabilities are stationary.

Notice that we need to incorporate an intermediate hidden level of synsets since we use a lexical database to handle synonyms. Since a given lemma could appear in one or more synsets, we need a first disambiguation phase just to select the right synset. Then we also need a second disambiguation phase, since a given synset could once again be mapped to several concepts (and one concept could be mapped to several synsets).

Mapping concepts directly to lemmas makes it more difficult to incorporate the full lexical database structure into the model. In fact, without using such a two-step approach, we could retrieve all the synonyms of a given lemma L by collecting all the lemmas L_s appearing in at least one synset S_i along with L but, this way, we would be mixing all the synsets together; as a consequence, we should consider these lemmas and the relationships R that connect them to the rest of the Lexical database:

$$L \overset{\text{find}\, S_i}{\rightarrow} \{S_i : L \in S_i\} \overset{\text{find}\, L_s,\ R}{\rightarrow} \begin{cases} \{L_s : L_s \in \bigcup_i S_i\} \\ \{R : R\, \text{involves}\, S_i\} \end{cases}$$

Instead, using our approach, we start from L and associate a single synset; thus we consider only its lemmas, and only the relationships that connect that synset to the rest of the lexical database:

$$L \overset{\text{find}\, S}{\rightarrow} S : L \in S \wedge S\, \text{has max probability} \overset{\text{find}\, L_s,\ R}{\rightarrow} \begin{cases} \{L_s : L_s \in S\} \\ \{R : R\, \text{involves}\, S\} \end{cases}$$

So, $p(L|S)$ and $p(S|C)$ are required. We use $p(L|S, C)$, instead of $p(L|S)$, since the former allows us to assert that different concepts, even if mapped to the same synset, could have different lemmatization "preferences". For example, assume we have the concepts `age` and `size`, and that both are (at least) mapped to the synset {big, large}. In this case, the following probabilities can be specified:

$$p(L = \text{big}|S = \{\text{big, large}\}, C = \text{size}) = p_1$$
$$p(L = \text{large}|S = \{\text{big, large}\}, C = \text{size}) = p_2$$
$$p(L = \text{big}|S = \{\text{big, large}\}, C = \text{age}) = p_3$$
$$p(L = \text{large}|S = \{\text{big, large}\}, C = \text{age}) = 0$$

Section 6.3 will detail the emission probability distributions $p(S|C)$ and $p(L|S, C)$, as well as the prior probability distribution $p(C)$.

Fig. 5 shows a graphical representation of the model, unrolled for some time slices, as well as the probability distributions we introduced.

Given the tth word W_t of a text, variable L_t takes values from the set of lemmas present in at least one of the synsets of $\boldsymbol{S}_{\mathbf{t}}^{\mathbf{eff}}$; see Eq. (1).

The variable S_t is defined over $\boldsymbol{S}_{\mathbf{t}}^{\mathbf{eff}}$; the set $\boldsymbol{S}_{\mathbf{t}}^{\mathbf{eff}}$ also contains the following special (void) synsets: `out of context` and `off-topic`. The former is mapped to a lemma whenever the lemma, evaluated in its linguistic context, is not meaningful for the domain (e.g., considering the sentence "My car is red", the word "red" should not be mapped neither to the `red` concept, nor to the `red wine` concept). The latter is mapped to words that are not part of the domain model (e.g., the word "car"). These two special synsets are referred to as *null synsets*.

Moreover, the variable C_t takes values from $\boldsymbol{C}_{\mathbf{t}}^{\mathbf{eff}}$, the set of concepts (classes, individuals, relationships) defined in the domain ontology and mappable to at least one synset in $\boldsymbol{S}_{\mathbf{t}}^{\mathbf{eff}}$; the set $\boldsymbol{C}_{\mathbf{t}}^{\mathbf{eff}}$ also contains the *null concepts* `out of context` and `off-topic`, whose meanings are equivalent to the ones defined for the synsets.

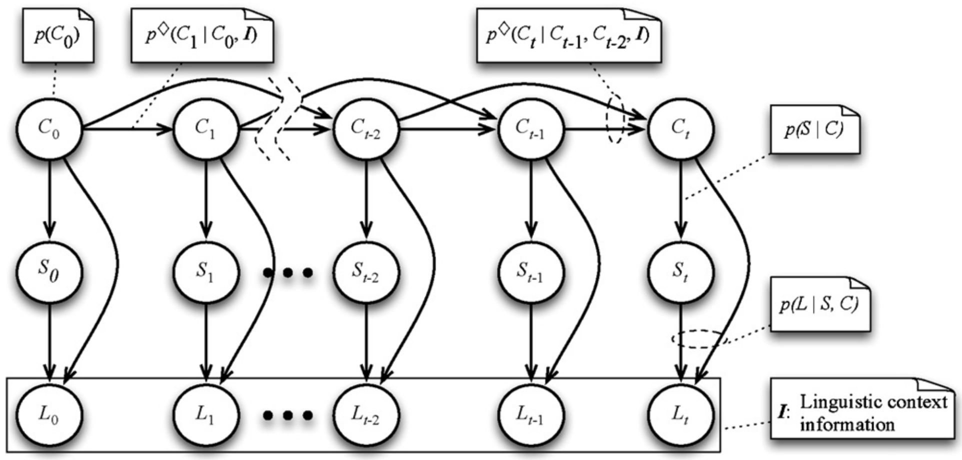
In Sections 6.2 and 6.3 we shall describe how transition, prior, and emission distributions are defined.

6.2. *Defining transition probability distributions: the MaxEnt models*

In order to take into account linguistic context information $\boldsymbol{I}$, we added it to the transition probability distributions. We shall now explain our motivations.

6.2.1. *Adding context information*

One could argue that the probability of a given sequence of concepts could be related to the sequence's context in the sentence. For example, consider the following sentences: "Sicilian Nero d'Avola produced by Donnafugata..." and "Sicilian Nero d'Avola, which. . .".



$p(C_0)$
$p^{\diamond}(C_1 | C_0, \boldsymbol{I})$
$p^{\diamond}(C_t | C_{t-1}, C_{t-2}, \boldsymbol{I})$
C_0
C_1
C_{t-2}
C_{t-1}
C_t
$p(S | C)$
S_0
S_1
S_{t-2}
S_{t-1}
S_t
$p(L | S, C)$
L_0
L_1
L_{t-2}
L_{t-1}
L_t
$\boldsymbol{I}$: Linguistic context information

Fig. 5. The ME-2L-HMM2, unrolled for some time slices.

The first sentence refers to a particular Nero d'Avola wine, the one produced by Donnafugata; the words "Nero d'Avola" should therefore be mapped to the concept `donnafugata nero d'avola`. Instead, the second sentence (probably) refers to the generic "Nero d'Avola" wine; therefore, we should map the words to the concept `nero d'avola`.

To do so, one could vary the transition probability distribution and/or the emission probability distribution, depending on the linguistic context information $\boldsymbol{I}$ of the lemmas being mapped. If we modify the transition probability distribution so that $p(C_t|C_{t-1}, \boldsymbol{I})$, we assert that the concept following `sicily` depends on the presence of the word "Donnafugata" in the sentence:

$$p(C_t = \text{donnafugata nero d'avola}\, C_{t-1} = sicily, \text{“Donnafugata”} \in \boldsymbol{I}) > p(C_t = \text{nero d'avola}\, C_{t-1} = sicily, \text{“Donnafugata”} \in \boldsymbol{I})$$

as the presence of "Donnafugata" is a hint that "Nero d'Avola" does not refer to the generic wine appellation.

If we modify the synset emission probability distribution so that $p(S_t|C_t, \boldsymbol{I})$, we assert that the concept to map to the synset {nero d'avola} depends on the presence of the word "Donnafugata":

$$p(S_t = \{\text{nero d'avola}\}|C_t = \text{donnafugata nero d'avola}, \text{“Donnafugata”} \in \boldsymbol{I}) > p(S_t = \{\text{nero d'avola}\}|C_t = \text{nero d'avola}, \text{“Donnafugata”} \in \boldsymbol{I})$$

since, once again, the presence of "Donnafugata" is a hint that {nero d'avola} does not refer to the generic wine appellation.

Finally, if we modify the lemma emission probability distribution so that $p(L_t|S_t, C_t, \boldsymbol{I})$, we state that the choice of the couple synset/concept to map to the lemma "nero d'avola" depends on the presence of the word "Donnafugata":

$$p(L_t = \text{“nero d'avola”}|S_t = \{\text{nero d'avola}\}, C_t = \text{donnafugata nero d' avola}, \text{“Donnafugata”} \in \boldsymbol{I}) > p(L_t = \text{“nerod'avola''}|S_t = \{\text{nerod'avola}\}, C_t = \text{nerod'avola}, \text{“Donnafugata”} \in \boldsymbol{I})$$

The simplest solution one could adopt is to modify the synset emission or the lemma emission probability distribution, since these distributions are small. Notice, however, that $\boldsymbol{I}$ is a potentially huge set, and the resulting distributions should provide a value for *each element* of this set. This leads to distributions that are hard to learn and represent.

Modifying the transition probability distributions seems the least desirable choice, since such distributions are already big. But, on the other hand, one could decide to concentrate all the model's complexity within these distributions, keeping the other distributions small and easy to calculate.

We opted for this last solution, and defined *functions* that could approximate transition probability distributions. Thus, not only does the mapping of the current word depend on the last two concepts, it also depends on the information that is stored in $\boldsymbol{I}$ (the presence of a particular word, morphologic or syntactic information, etc.). Conversely, the emission distribution probabilities do not incorporate $\boldsymbol{I}$; this way, they remain small and easy to calculate.

A similar approach, where the authors estimate transition probabilities of a second-order HMM by means of a model (in that case, a decision tree) can be found in Schmid (1994).

As a model for the definition of such functions, we chose the maximum entropy (MaxEnt) model (Jaynes, 1968; Ratnaparkhi, 1997; Manning and Schütze, 1999), which is widely used to represent probability distributions, and can easily include heterogeneous information.

6.2.2. Approximating transition probability distributions with a maximum entropy model

With MaxEnt models it is possible to represent both joint probability distributions $p(Y, X)$ and conditioned probability distributions $p(Y|X)$:

$$p(Y, X) = \frac{exp\left(\sum_i^n \gamma_i f_i(Y, X)\right)}{Z} \tag{5}$$

$$p(Y|X) = \frac{\exp\left(\sum_i^n \gamma_i f_i(Y, X)\right)}{Z(X)} \tag{6}$$

where the *partition functions* Z and $Z(X)$ – which normalize the function, as the definition of probability requires – are defined as:

$$Z \overset{\text{def}}{=} \sum_{Y,X} exp\left(\sum_i^n \gamma_i f_i(Y, X)\right) \tag{7}$$

$$Z(X) \overset{\text{def}}{=} \sum_{Y} \exp\left(\sum_i^n \gamma_i f_i(Y, X)\right) \tag{8}$$

and the so-called *features* $f_i(Y, X)$ are defined as pattern indicator functions: features return 1 whenever a given (y, x) pattern is found, otherwise they return 0.

The constants γ_i weight the "importance" of each feature for the calculation of the distribution values. The features set $\boldsymbol{F} \overset{\text{def}}{=} \{f_i(Y, X) : i = 1, 2, \ldots\}$, along with the weights set $\boldsymbol{\Gamma} \overset{\text{def}}{=} \{\gamma_i : i = 1, 2, \ldots\}$, constitute the *parameters* of the model.

Applying Eq. (6) to our probability distribution $p(C_t|C_{t-1}, \boldsymbol{I})$, we obtain:

$$p(C_t|C_{t-1}, \boldsymbol{I}) = \frac{exp\left(\sum_i^n \gamma_i f_i(C_t, C_{t-1}, \boldsymbol{I})\right)}{Z(C_{t-1}, \boldsymbol{I})} \tag{9}$$

where the partition function follows Eq, (8):

$$Z(C_{t-1}, \boldsymbol{I}) \overset{\text{def}}{=} \sum_{C_t \in \mathbf{C_t^{eff}}} exp\left(\sum_i^n \gamma_i f_i(C_t, C_{t-1}, \boldsymbol{I})\right) \tag{10}$$

Unfortunately, using Eq. (9) to define the transition function of an HMM implies the calculation of the partition function $Z(C_{t-1}, \boldsymbol{I})$. In fact, during the decoding phase, at a given time t, the probability associated with a given hidden state C_t is:

$$p(C_t) = \max_{C_{t-1} \in \mathbf{C_{t-1}^{eff}}} (p(C_{t-1}) \cdot p(C_t|C_{t-1}, \boldsymbol{I})) \cdot p(L_t|C_t);\ \forall C_t \in \mathbf{C_t^{eff}} \tag{11}$$

where each C_t reaches, in general, the maximum for a different value of C_{t-1} ($\boldsymbol{I}$ is a constant, since its value is set once the document to decode has been selected). Then, the "best" hidden state selected by the decoding procedure will be the one with maximum probability:

$$\tilde{C}_t \overset{\text{def}}{=} \underset{C_t \in \mathbf{C_t^{eff}}}{\text{argmax}}\ p(C_t) \tag{12}$$

Avoiding to calculate $Z(C_{t-1}, \boldsymbol{I})$ we are actually multiplying the internal of $\max(\cdots)$ of Eq. (11) by a function that depends on C_{t-1}; this will change $p(C_t)$'s values and, likely, modify the $\tilde{C}_t$ selected by the decoding procedure. Thus, in order to get $p(C_t|C_{t-1}, \boldsymbol{I})$, we need to calculate $Z(C_{t-1}, \boldsymbol{I})$ for each value of C_t, performing $|\mathbf{C_t^{eff}}|$ summations, and repeating such summations for each t (pre-calculating the values of $Z(C_{t-1}, \boldsymbol{I})$ is impractical, as the presence of $\boldsymbol{I}$ makes the domain extremely big).

The calculation of the partition function can be avoided, however, by recalling the definition of conditional probability

$p(a|b) \overset{\text{def}}{=} p(a, b)/p(b)$. By defining $p(C_t, C_{t-1}, \boldsymbol{I})$ and $p(C_{t-1}, \boldsymbol{I})$ as joint MaxEnt models, according to Eq. (5), we can define:

$$p(C_t|C_{t-1}, \boldsymbol{I}) = \frac{p(C_t, C_{t-1}, \boldsymbol{I})}{p(C_{t-1}, \boldsymbol{I})} = \frac{(1/Z_2)\exp(\sum_i^n \gamma_i f_i(C_t, C_{t-1}, \boldsymbol{I}))}{(1/Z_1)\exp(\sum_i^m \psi_i f_i(C_{t-1}, \boldsymbol{I}))} \tag{13}$$

where Z_1 and Z_2, according to Eq. (7), are constants. Thus, dropping the constants, we can define the following *scaled transition distribution*:

$$p^{\diamond}(C_t|C_{t-1}, \boldsymbol{I}) \overset{\text{def}}{=} \frac{exp\left(\sum_i^n \gamma_i f_i(C_t, C_{t-1}, \boldsymbol{I})\right)}{\exp\left(\sum_i^m \psi_i f_i(C_{t-1}, \boldsymbol{I})\right)} \tag{14}$$

whose parameters are $\Phi_1 \overset{\text{def}}{=} (\boldsymbol{F_2}, \boldsymbol{F_1}, \boldsymbol{\Gamma}, \boldsymbol{\Psi})$, where $\boldsymbol{\Gamma}$ has been defined before, $\boldsymbol{\Psi} \overset{\text{def}}{=} \{\psi_i : i = 1, 2, \ldots\}$, $\boldsymbol{F_2} \overset{\text{def}}{=} \{f_i(C_t, C_{t-1}, \boldsymbol{I}) : i = 1, 2, \ldots\}$, and $\boldsymbol{F_1} \overset{\text{def}}{=} \{f_i(C_t, \boldsymbol{I}) : i = 1, 2, \ldots\}$.

Notice that if we substitute $p^{\diamond}(C_t|C_{t-1}, \boldsymbol{I})$ to $p(C_t|C_{t-1}, \boldsymbol{I})$ in our ME-2L-HMM2 model, Eq. (12) is not affected, since in Eq. (11) we are now multiplying by a positive constant.

The second-order transition probability distribution $p(C_t|C_{t-1}, C_{t-2}, \boldsymbol{I})$ implies the calculation of $Z(C_{t-1}, C_{t-2}, \boldsymbol{I})$, again performing $|\boldsymbol{C_t^{eff}}|$ summations; or we can define the scaled transition distribution:

$$p^{\diamond}(C_t|C_{t-1}, C_{t-2}, \boldsymbol{I}) \overset{\text{def}}{=} \frac{exp\left(\sum_i^h \lambda_i f_i(C_t, C_{t-1}, C_{t-2}, \boldsymbol{I})\right)}{\exp\left(\sum_i^n \gamma_i f_i(C_{t-1}, C_{t-2}, \boldsymbol{I})\right)} \tag{15}$$

whose parameters are $\Phi_2 \overset{\text{def}}{=} (\boldsymbol{F_3}, \boldsymbol{F_2}, \boldsymbol{\Lambda}, \boldsymbol{\Gamma})$, where $\boldsymbol{\Gamma}$ and $\boldsymbol{F_2}$ have been defined before, $\boldsymbol{\Lambda} \overset{\text{def}}{=} \{\lambda_i : i = 1, 2, \ldots\}$ and $\boldsymbol{F_3} \overset{\text{def}}{=} \{f_i(C_t, C_{t-1}, C_{t-2}, \boldsymbol{I}) : i = 1, 2, \ldots\}$.

Thanks to the stationary assumption, weights and feature sets in the numerator of Eq. (14) and denominator of Eq. (15) are the same; thus, the whole model is composed of three feature sets and the associated three sets of weights.

Notice that, adopting our approach, instead of calculating the partition function, results in a faster model computation, as long as $|\boldsymbol{C_t^{eff}}| > 1$ (for unambiguous lemmas, $|\boldsymbol{C_t^{eff}}| = 1$, the transition function is actually not computed). In particular, the time complexity in our approach for the calculation of the transition function is *independent* of $|\boldsymbol{C_t^{eff}}|$, while the time complexity of calculating the partition function depends linearly on $|\boldsymbol{C_t^{eff}}|$.

The set $|\boldsymbol{C_t^{eff}}|$ is often small, as the experiment in Section 11 will show; however, given the decoding algorithm needed for our second-order model (see Section 8.1), even a small advantage in the computation of the transition probability distribution could have an impact on the decoding performance (see Section 11.3.3 for more details on performances).

Our approach, however, does not come without drawbacks; in particular, the training phase is slower since we need to calculate three sets of weights, instead of the two we would need if we chose to calculate the partition function.

6.2.3. Defining indicator functions: feature templates

The definition of a good feature set is crucial for the MaxEnt model to give a good approximation of the probability distributions. This is why we developed a set of pattern indicator functions, and experimented their effects on the system's performance.

In MaxEnt models, *feature templates* represent a way to define indicator functions. A feature template is a model that describes a pattern structure that predicates on both the concept set $\boldsymbol{C}$ and the information $\boldsymbol{I}$. During the training process, the system generates all the patterns that meet the structure; these patterns become the template's *feature instances*. During the decoding process, the concepts under investigation, as well as the linguistic context information, are checked against feature instances; the feature instance whose pattern matches (if found) returns 1.

As an example, a feature template T defining a pattern that predicates on concepts C_t, C_{t-1}, C_{t-2}, and on lemmas L_t, L_{t-1}, L_{t+1} (recall that lemmas are part of $\boldsymbol{I}$) is represented by the following function:

$$T(C_t, C_{t-1}, C_{t-2}, \boldsymbol{I}) \overset{\text{def}}{=} \begin{cases} 1; & C_t = c_1 \wedge C_{t-1} = c_2 \\ & \wedge C_{t-2} = c_3 \wedge \\ & L_t = l_1 \wedge L_{t-1} = l_2 \\ & \wedge L_{t+1} = l_3 \\ 0; & \text{otherwise} \end{cases} \tag{16}$$

where c_1, c_2, c_3, l_1, l_2, and l_3 represent placeholders that will be instantiated with actual data. Given the following piece of training data, in the form (t, *lemma*, *associated concept*):

{(0, "light", lightWine), (1, "red", redWine), (2, "wine", Wine), (3, "bottle", wineBottle)}

and assuming that the current step is $t = 2$, a generated ith feature instance would be:

$$f_{i,T}(C_t, C_{t-1}, C_{t-2}, \boldsymbol{I}) = \begin{cases} 1; & C_t = Wine \wedge \\ & C_{t-1} = redWine \wedge \\ & C_{t-2} = lightWine \wedge \\ & L_t = \text{"wine"} \wedge \\ & L_{t-1} = \text{"red"} \wedge \\ & L_{t+1} = \text{"bottle"} \\ 0; & otherwise \end{cases} \tag{17}$$

Thus, a feature template generates a *set* of feature instances. Each feature instance has a counter; every time the pattern specified by the instance is found in the training data, its counter increases. This is required for the feature selection that will be described in Section 7.2.

There are three types of templates: the first one, called *trigram*, considers C_t, C_{t-1}, and C_{t-2}; the second one, called *bigram*, considers C_t, C_{t-1}; the third one, called *unigram*, only considers C_t. Templates can focus on a specific information type in $\boldsymbol{I}$ (for example, the aforementioned template T only used lemmas) or can mix several types (predicating, for example, on lemmas, POS's, synsets, etc.). Finally, templates specify time interval(s) for the pieces of information that need to be considered (for example, the template T uses $t-1$, $t+1$ as a time interval for lemmas).

When defining templates, one could argue that a single trigram template that predicates on a large time interval, and considers every information type, could result in very informative instances. This is true but, unfortunately, patterns with several components are more difficult to find in the training data, and, therefore, the associated counter could be quite small, or even zero. In other words, complex patterns could be very informative, but need huge training sets to work properly.

Carefully choosing the template type, the time intervals, and the information types, we tried to balance informativeness and complexity. We experimented with dozens of templates, training our model and running the evaluation tests explained in Sections 11.1 and 11.2. At the end, the best template set was the one shown in Table 1. Templates T4 and T11 predicate on concepts and lemmas; T27 predicates on lemmas and "candidate concepts" (CA_t: the set

Table 1
Templates.

Name	3gram condition	2gram condition	1gram condition
T11	C_t, C_{t-1}, C_{t-2} $L_{t-3}, L_{t-2}, L_{t-1}$ $L_t, L_{t+1}, L_{t+2}, L_{t+3}$	C_t, C_{t-1} $L_{t-3}, L_{t-2}, L_{t-1}$ $L_t, L_{t+1}, L_{t+2}, L_{t+3}$	C_t $L_{t-3}, L_{t-2}, L_{t-1}$ $L_t, L_{t+1}, L_{t+2}, L_{t+3}$
T27	C_t, C_{t-1}, C_{t-2} $CA_{t-3}, CA_{t+1}, CA_{t+2}$ $CA_{t+3}, CA_{t+4}, CA_{t+5}$	C_t, C_{t-1} $CA_{t-2}, CA_{t-1}, CA_{t+1}$ $CA_{t+2}, CA_{t+3}, CA_{t+4}, CA_{t+5}$	C_t $CA_{t-3}, CA_{t-2}, CA_{t-1}$ $CA_{t+1}, CA_{t+2}, CA_{t+3}, CA_{t+4}, CA_{t+5}$
T28	C_t, C_{t-1}, C_{t-2} $POS_t, POS_{t-1}, POS_{t-2}$	C_t, C_{t-1} POS_t, POS_{t-1}	C_t POS_t
T4	C_t, C_{t-1}, C_{t-2} L_t, L_{t-1}, L_{t-2}	$C_t, C_{t,1}$ L_t, L_{t-1}	C_t L_t

of concepts mappable to L_t); and T28 predicates on concepts and POSs.

The feature instances undergo a filtering process that reduces their number, while retaining the most informative ones; in particular, our approach will be presented in Section 7.2.

6.3. *Defining prior and emission probability distributions: probabilistic mapping*

In the sections above we defined the scaled transition distribution for our ME-2L-HMM2 model. The prior probability $p(C)$, and the emission probabilities $p(L|S, C)$ and $p(S|C)$ are much more simple to calculate:

$$p(C) = \frac{c(C)}{\sum_{C'} c(C')}; \quad p(S|C) = \frac{c(S, C)}{c(C)}; \quad p(L|S, C) = \frac{c(L, S, C)}{c(S, C)} \qquad (18)$$

where the function $c(\ldots)$ counts the number of occurrences.

Notice that these probability distributions encode the domain ontology-lexical database mappings. The example of Fig. 6 shows the probability values associated with a fragment of the mapping depicted in Fig. 3. For example, the probability for synset {color, colour} to emit the word “color” (or, in other words, the probability for the word “color” to carry the meaning intended by the synset {color, colour}) depends on the concept we are considering (`WineColor` or `AppellationColor`).

The lemma emission probability distribution depends on both synsets and concepts; this is meant to capture the fact that concepts, although lexicalized by the same synset, might not have, in principle, the same lemma distribution (see Section 6.1).

Fig. 6 depicts three domain ontology-lexical database types of mapping. The first type of mapping is one-to-one: concept and synset are univocally connected to each other; this is of course the simplest possible type of mapping. The second type of mapping is one-to-many: a concept is connected to many synsets; this is often used to lexicalize a concept by means of different synset types (e.g., a noun synset and a adjective synset). The third type of mapping is many-to-one: many concepts are connected to the same synset; this is the most interesting type because, during the WSD procedure, the model must be able to select the concept with the highest probability; it is easy to recognize that the probability distributions $p(L = \text{“color''}|S = \{\text{color, colour}\}, C)$, in conjunction with prior probability distribution $p(C)$, carry enough information to allow us to select the concept with the highest probability.

Fig. 7 shows two particular cases. On the right, the probability distributions $p(L = \text{“nerod'avola''}|S = \{\text{nerod'avola}\}, C)$ are both equal to 1, since the synset contains just one lemma; in this case, the selection of the best concept is led by the prior probability distribution $p(C)$. On the left, two synsets contain a common word (“state”); in this case, probability distributions $p(L = \text{“state''}|S, C)$ carry different values, since the two synsets are not identical, and can both contribute to the selection of the concept with the highest probability.

We now have all the distributions required by our ME-2L-HMM2 model: the two scaled transition distributions $p^{\Diamond}(C_t|C_{t-1}, \boldsymbol{I})$ and $p^{\Diamond}(C_t|C_{t-1}, C_{t-2}, \boldsymbol{I})$ defined in Section 6.2, the concept prior probability distribution $p(C)$, and the emission probability distributions $p(L|S, C)$ and $p(S|C)$.

6.4. *Comparing ME-2L-HMM2 against alternative stochastic approaches*

HMMs are useful because they are compact, easy to train, and efficient during decoding. Our approach tries to retain these characteristics, while extending the model to become second order, and while adding information $\boldsymbol{I}$. The MaxEnt models we define allow us to obtain a simple and powerful model. One could

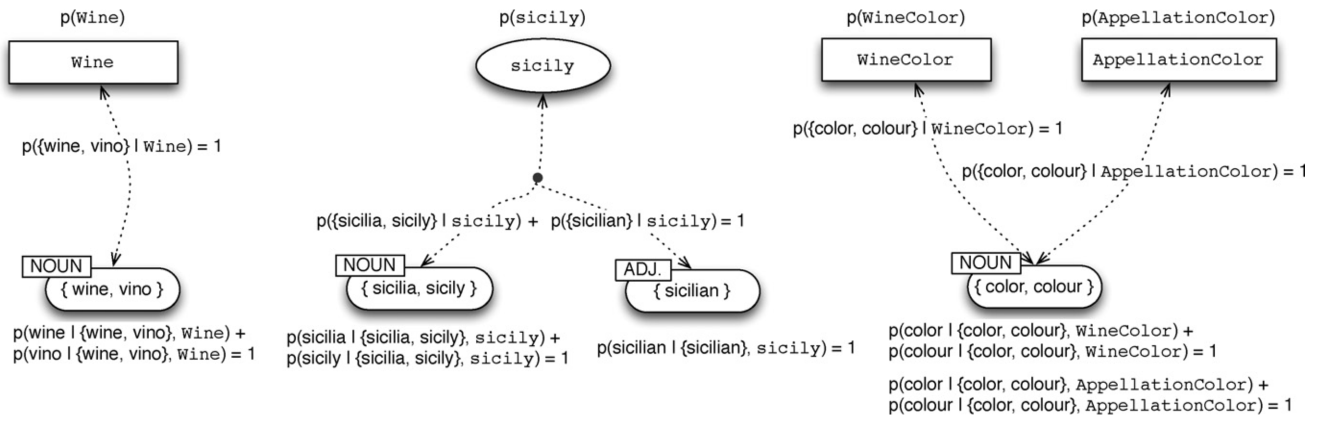

p(Wine)
Wine
p({wine, vino} | Wine) = 1
NOUN
{ wine, vino }
p(wine | {wine, vino}, Wine) +
p(vino | {wine, vino}, Wine) = 1
p(sicily)
sicily
p({sicilia, sicily} | sicily) + p({sicilian} | sicily) = 1
NOUN
{ sicilia, sicily }
ADJ.
{ sicilian }
p(sicilia | {sicilia, sicily}, sicily) +
p(sicily | {sicilia, sicily}, sicily) = 1
p(sicilian | {sicilian}, sicily) = 1
p(WineColor)
WineColor
p(AppellationColor)
AppellationColor
p({color, colour} | WineColor) = 1
p({color, colour} | AppellationColor) = 1
NOUN
{ color, colour }
p(color | {color, colour}, WineColor) +
p(colour | {color, colour}, WineColor) = 1
p(color | {color, colour}, AppellationColor) +
p(colour | {color, colour}, AppellationColor) = 1

Fig. 6. Domain ontology-lexical database mapping types.

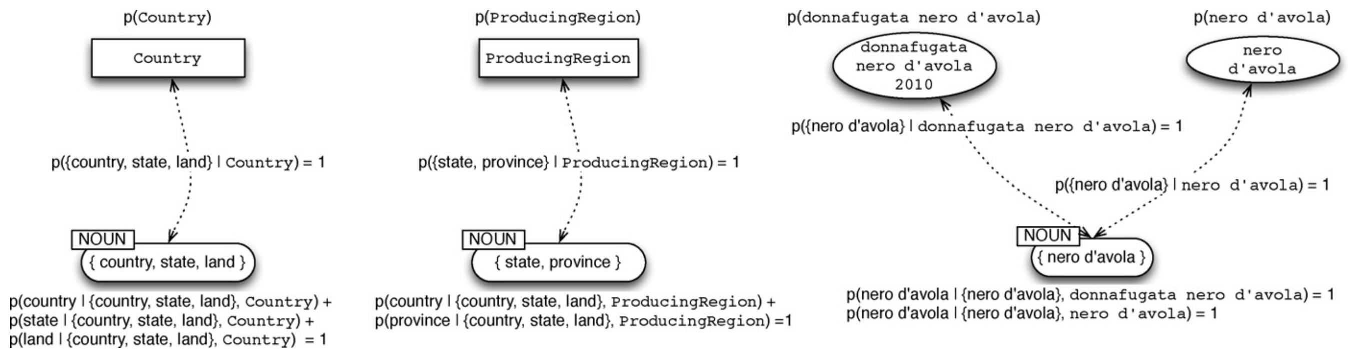

p(Country)
Country
p({country, state, land} | Country) = 1
NOUN
{ country, state, land }
p(country | {country, state, land}, Country) +
p(state | {country, state, land}, Country) +
p(land | {country, state, land}, Country) = 1
p(ProducingRegion)
ProducingRegion
p({state, province} | ProducingRegion) = 1
NOUN
{ state, province }
p(country | {country, state, land}, ProducingRegion) +
p(province | {country, state, land}, ProducingRegion) =1
p(donnafugata nero d'avola)
donnafugata nero d'avola 2010
p(nero d'avola)
nero d'avola
p({nero d'avola} | donnafugata nero d'avola) = 1
p({nero d'avola} | nero d'avola) = 1
NOUN
{ nero d'avola }
p(nero d'avola | {nero d'avola}, donnafugata nero d'avola) = 1
p(nero d'avola | {nero d'avola}, nero d'avola) = 1

Fig. 7. Two particular cases.

wonder, however, if a pure MaxEnt approach could have produced a better model. Maximum entropy markov models (MEMM) and conditional random fields (CRF) are two popular MaxEnt-based models that handle sequences of events.

MEMMs (McCallum et al., 2000) combine maximum entropy models and HMM, making it possible to decode sequences of words by means of a slightly modified Viterbi procedure. MEMMs replace transition and emission probability distributions with a single probability distribution like $p(C_t|C_{t-1}, L_t)$, modeled by means of a MaxEnt model. This probability distribution, however, becomes very complex if one attempt to extend the model to the second order and incorporate an intermediate level for synsets (as our model does).

CRFs (Lafferty et al., 2001) combine maximum entropy models and HMM, and transform the constant transition probabilities into arbitrary functions that vary across the positions in the sequence of hidden states, depending on the input sequence. CRFs outperform both MEMMs and HMMs on a number of real-world sequence labeling tasks (Wallach, 2004). Training CRFs, however, is difficult, since the function to optimize (in order to calculate the model parameters) is in general quite big and cannot be written in closed form; yet methods do exist that can approximate the function (L-BFGS is a popular algorithm). Moreover, as in MEMMs, the model becomes even more complex if we try to add an intermediate level.

Our second-order model is much simpler and can be trained by means of the GIS algorithm. Like MEMM and CRF, features defined in our model can predicate on the past, current, and future observable information. Even if the model does not explicitly connect the choice of current hidden state to future states, incorporating future observable information into $\boldsymbol{I}$ allows us to take into account the "right part" of the sentence when choosing the concept for the current lemma. Notice that HMM independence assumptions, which are retained by our model, add undesired constraints but, once again, the presence of $\boldsymbol{I}$ alleviates the problem; in fact, transition probabilities will depend on information at different time instants.

In the following section we introduce the three-phase training procedure, which calculates the distributions required by the ME-2L-HMM2 mapping model.

7. Training the ME-2L-HMM2 mapping model

The training process is divided into several steps. First of all, a specific set of documents in the document repository enter the linguistic context extractor, which retrieves the morphologic and syntactic information. Then, the domain expert associates each word with the right concept and synset, creating two sub-sets: the *training set* and the *test set*. If a given lemma, evaluated in its linguistic context, is not meaningful for the domain, the domain expert associates it to the `out of context` concept and synset. Lemmas that are not part of the domain model are associated to the `off-topic` concept and synset. The three-phase training procedure relies on both the domain ontology and the lexical database, analyzes the training set documents, and generates the statistical model. Finally, the domain model is validated using the test set (see Section 11).

The training set is defined as $\boldsymbol{TR} \overset{\text{def}}{=} \{(C_t, S_t, L_t) : t = 0, 1, \ldots\}$, wherein L_t represents the observed lemma, while S_t and C_t represent the hidden values. Then, the system adds the (observable) information $\boldsymbol{I}$, as explained in Section 5. The test set $\boldsymbol{TS}$, which will be used in Section 11 for the evaluation of the model, shares the same structure as $\boldsymbol{TR}$.

As an example, consider the sentence "The bottle with the red label contains Barolo, a ruby wine". The domain expert associates words, synsets, and concepts, producing "The bottle/{bottle}/`WineBottle` with red/{out of context}/`out of context` label/{label}/`WineLabel` contains Barolo/{barolo}/`barolo wine`, a ruby/{red}/`red wine` wine/{wine}/`Wine`". Notice that the adjective "red" just before the noun "label" is tagged with `out of context`, since its meaning (the color of a label) is not considered useful for the domain. Instead, just before the noun "wine", the adjective "ruby" refers to a wine color, and therefore it is tagged with the related synset and concept. Words that are not tagged are considered `off-topic`.

Fig. 8 shows the internal components of the training module. The probability distribution generator calculates the prior and emission probability distributions; the feature instances generator reads the set of feature templates and generates the set of feature instances; finally, the MaxEnt models generator calculates the MaxEnt models' weights.

Notice that, given the fact that emission probability distributions $p(L|S, C)$ and $p(S|C)$ are not huge (as lemmas are usually mapped to a few synsets and concepts), and that transition

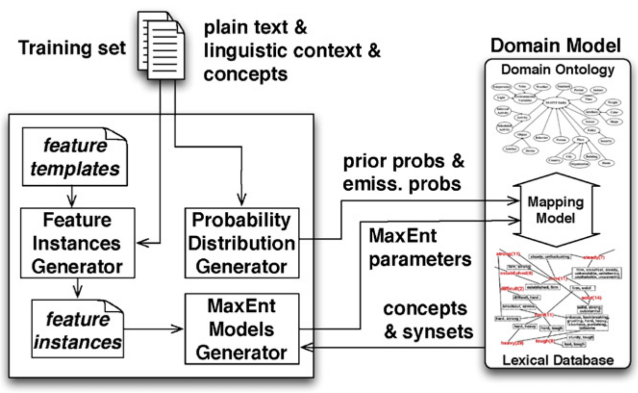


Fig. 8. The training module internal structure, and its connections to the domain model.

probability distributions $p^{\Diamond}(C_t|C_{t-1},\ \boldsymbol{I})$ and $p^{\Diamond}(C_t|C_{t-1},\ c_{t-2},\ \boldsymbol{I})$ are approximated using a model, we expect that a small training set can still generate a model with a good performance (see Section 11).

7.1. Calculating prior and emission probability distributions

The first phase of the training procedure calculates prior and emission probability distributions $p(C)$, $p(S|C)$, and $p(L|S,\ C)$, by applying their definitions (see Eq. (18)) to the training set $\boldsymbol{TR}$. At the end of the calculations, only the *trained lemmas L* – that is lemmas that appear in the training set – will have a $p(L|S,C) > 0$; lemmas that do not appear in the training set (that is *untrained lemmas*) will not take part in the probability distribution, even if they belong to the same synset of some trained lemma.

The fact that only trained lemmas get a distribution probability poses a problem, however, since the system should be able to index and search for all the synonyms of trained lemmas.

Thus, we have to complete the model by adding untrained lemmas that are synonyms of some trained lemmas. If $\boldsymbol{L}'$ is a set containing synonyms of untrained lemmas, $\boldsymbol{S}_{L'}$ is a set comprised of the synsets that contain a given lemma L', and $\boldsymbol{C}'$ is a set containing the concepts that are mapped to synsets in $\boldsymbol{S}_{L'}$, we assign the following distribution probability value to each lemma L':

$$p_{L'} \overset{\text{def}}{=} p(L'|S_{L'},C') = \frac{\min_{L,S,C} p(L|S,\ C)}{|\boldsymbol{L}'|}, \quad \forall (S_{L'},L',C') \tag{19}$$

or, in other words, we assign the minimum lemma emission probability to the entire set $\boldsymbol{L}'$, and then have each lemma L' receive the same fraction of that probability mass. Notice that, for each L', the mapping $p(L'|out\ \ of\ context, out\ \ of\ \ context) = p_{L'}$ is also added, modeling the fact that, since no training samples were found for L', the lemma could, in general, be mapped to the `out of context` concept and synset.

Since we added new probability mass, we need to renormalize the probability distribution, so that the new probability distribution still sums up to one:

$$\sum_L p^{\text{new}}(L|S_{L'},C') + p_{L'} \cdot |\boldsymbol{L}'| = 1, \quad \forall (S_{L'},C') \tag{20}$$

Moreover, the newly added probability mass must not change the relative strength of probability values inside synsets in $\boldsymbol{S}_{L'}$:

$$\frac{p^{\text{new}}(L_i|S_{L'},C')}{p^{\text{new}}(L_j|S_{L'},C')} = \frac{p^{\text{old}}(L_i|S_{L'},C')}{p^{\text{old}}(L_j|S_{L'},C')}, \quad \forall (i,j), \quad \forall (S_{L'},C') \tag{21}$$

or, in other words, the modified values should be calculated multiplying the old values by a constant.

From these two constraints, it is easy to show that the new probability distribution is normalized with:

$$p^{\text{new}}(L|S_{L'},C') = p^{\text{old}}(L|S_{L'},C') \cdot (1 - p_{L'} \cdot |\boldsymbol{L}'|), \quad \forall (L,S_{L'},C') \tag{22}$$

7.2. Generating the set of feature instances

The second phase of the training procedure consists in generating the set of feature instances needed by the MaxEnt models we use to calculate the scaled transition distributions.

Applying a feature template to the training set generates a set of feature instances; each feature instance contains a particular instantiation of the pattern defined by the corresponding template. These feature instances are not equally informative, however; for example, one could argue that rare pattern instances should be less significant than frequent ones. Moreover, the number of feature instances could simply become too high for the system to cope with. Then, a feature selection method is needed, in order to reduce the total amount of feature instances, retaining the most informative ones.

Actually, there are several methods for addressing the feature selection problem; from the sophisticated approach proposed in Ratnaparkhi et al. (1994), wherein each feature instance is evaluated according to its contribution to the log-likelihood of the resulting MaxEnt model, to the simple method adopted in Charniak and Johnson (2005), where a threshold is used.

Our approach makes use of two thresholds and tries to select feature instances that are both specific and frequent. Let us rewrite a generic template as $T_i(C,\ \boldsymbol{X})$, wherein $\boldsymbol{X}$ encloses all the "conditioning" template variables (the ones that appear as conditioning variables in the scaled transition distributions), while C represents the concept under evaluation (appearing as the conditioned variable in the scaled transition distributions). Then, the jth feature instance belonging to the template T_i is defined by the data it contains, and is written as: $f_{j,T_i}(C = c_j, \boldsymbol{X} = \mathbf{x_j})$.

The occurrence counter $K_{j,T_i}(C = c_j, \boldsymbol{X} = \mathbf{X_j})$ stores how many times the jth feature instance $f_{j,T_i}(C = c_j, \boldsymbol{X} = \mathbf{x_j})$ is found into the training set. In other words, such a counter expresses how many times the conditioning pattern $\mathbf{x_j}$ was associated with the concept c_j in the training set.

We argue that a feature instance carries useful information if it is likely that its conditioning pattern $\mathbf{x_j}$ predicts a single and specific concept c_j. On the other hand, a feature that is found many times should be considered more important than a rare one.

In other words, the selection rule retains feature instances that are both specific and frequent; i.e., feature instance $f_{j,T_i}(C = c_j, \boldsymbol{X} = \mathbf{x_j})$ is retained iff:

$$\begin{cases} \dfrac{K_{j,T_i}(C = c_j, \boldsymbol{X} = \mathbf{x_j})}{\sum_{c' \in \mathbf{C}} K_{j,T_i}(C = c', \boldsymbol{X} = \mathbf{x_j})} & > k_{\text{spec}} \\ K_{j,T_i}(C = c_j, \boldsymbol{X} = \mathbf{x_j}) & > k_{\text{count}} \end{cases} \tag{23}$$

where $k_{count} \geq 1$ is the count threshold, while $0 \leq k_{\text{spec}} \leq 1$ represents the "specificity" of the feature instance – the minimum fraction of times that the conditioning pattern $\mathbf{x_j}$ must predict the concept c_j in order for the feature to be considered specific enough.

7.3. Calculating scaled transition distributions: the GIS algorithm

Once the feature instances have been generated, the third phase of the training procedure puts them into the MaxEnt models generator, which calculates the scaled transition distributions; in particular, we use the generalized iterative scaling (GIS) algorithm for unconditioned models (Darroch and Ratcliff, 1972; Manning and Schütze, 1999) to calculate the parameters $\boldsymbol{\Phi}$ of the MaxEnt models that approximate such distributions. Since finding these parameters is a convex problem, the GIS algorithm is guaranteed to converge to the global optimum (Darroch and Ratcliff, 1972). The proposed algorithm is the simplest implementation of the GIS training procedure, and was fast enough for our experimentation; more efficient variations exist, like SCGIS (Goodman, 2002) or IIS (Della Pietra and Della Pietra, 1997).

The GIS algorithm for unconditioned models $p(C, \mathbf{X})$ uses the training set $\boldsymbol{TR}$, the information $\boldsymbol{I}$, and a set of features $\boldsymbol{F} \overset{\text{def}}{=} \{f_i(C, \boldsymbol{X}) : i = 1, 2, \ldots\}$; the algorithm iteratively calculates the set of weights $\boldsymbol{\Theta} \overset{\text{def}}{=} \{\theta_i : i = 1, 2, \ldots\}$ so that, in the generated MaxEnt model, the following condition holds:

$$E_{\text{observed}}[f_i(C, \boldsymbol{X})] = E_{\text{expected}}[f_i(C, \boldsymbol{X})], \quad \forall i \tag{24}$$

where $E_{\text{observed}}[f_i(C,\ \boldsymbol{X})]$ is the expectation (i.e., the mean value) of the ith feature, calculated on the training set samples, while $E_{\text{expected}}[f_i(C,\ \boldsymbol{X})]$ is the expectation of the same feature, calculated using the model the algorithm is training. If these two expectations

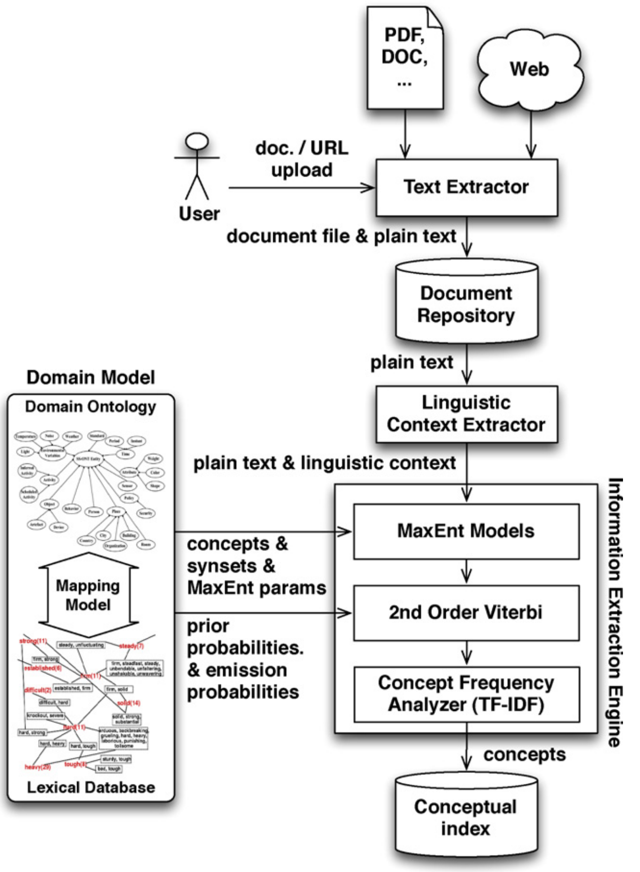

PDF, DOC, ...
Web
User
doc. / URL upload
Text Extractor
document file & plain text
Document Repository
plain text
Domain Model
Domain Ontology
Linguistic Context Extractor
plain text & linguistic context
Information Extraction Engine
MaxEnt Models
concepts & synsets & MaxEnt params
Mapping Model
2nd Order Viterbi
prior probabilities. & emission probabilities
Concept Frequency Analyzer (TF-IDF)
concepts
Lexical Database
Conceptual index

Fig. 9. The indexing process, with the information extraction engine internal structure and its connections to the domain model.

are equal, for all the features, the trained model fits the training set. Appendix A depicts our implementation of the GIS algorithm, and its complexity.

Recall that we need to calculate the two models defined by parameters $\Phi_1 \overset{\text{def}}{=} (\boldsymbol{F_2}, \boldsymbol{F_1}, \boldsymbol{\Gamma}, \boldsymbol{\Psi})$ and $\Phi_2 \overset{\text{def}}{=} (\boldsymbol{F_3}, \boldsymbol{F_2}, \boldsymbol{\Lambda}, \boldsymbol{\Gamma})$. Therefore, the GIS algorithm runs three times: the first run poses the initial time $t_0 = 2$ and $\boldsymbol{X_t} = (C_{t-1}, C_{t-2}, \boldsymbol{I})$, uses features $\boldsymbol{F_3}$, and assigns $\boldsymbol{\Theta}$ to weights $\boldsymbol{\Lambda}$, obtaining the numerator of Eq. (15); the second run poses $t_0 = 1$ and $\boldsymbol{X_t} = (C_{t-1}, \boldsymbol{I})$, uses features $\boldsymbol{F_2}$, and assigns $\boldsymbol{\Theta}$ to weights $\boldsymbol{\Gamma}$, obtaining both the denominator of Eq. (15) and the numerator of Eq. (14); finally, the third run poses $t_0 = 0$ and $\boldsymbol{X_t} = (\boldsymbol{I})$, uses features $\boldsymbol{F_1}$, and assigns $\boldsymbol{\Theta}$ to weights $\boldsymbol{\Psi}$, obtaining the denominator of Eq. (14).

Now that the domain model has been trained, the meaning of the words in the text set can be disambiguated, and the index can be created. Section 8 introduces this procedure.

8. Indexing documents

As Fig. 9 shows, the indexing process exploits the domain model to perform the WSD procedure and update the conceptual index. The linguistic context extractor generates the linguistic context information $\boldsymbol{I}$; then, the information extraction engine performs the decoding and indexing. As a part of the decoding procedure, the MaxEnt models calculate the scaled transition distributions; then, the 2nd order Viterbi algorithm selects the lemma-synset-concept association with the highest probability. Finally, the concept frequency analyzer builds the conceptual index.

Continuing the example of Section 7, assume the system be trained to recognize concepts in the following set {`WineBottle`, `WineLabel`, `barolo wine`, `red wine`, `Wine`}. If the document set is composed of D_1 = “Barolo is a small village where good wine is produced” and D_2 = “Barolo is a red wine”, the system will extract the following sets of concepts [3] $\boldsymbol{C_1}$={`Wine`} and $\boldsymbol{C_2}$={`barolo wine`, `red wine`, `Wine`}. The concept frequency analyzer will calculate the weight $w_{c,d}$ associated with each concept c, for each document d; thus, the following vectors are generated: $\boldsymbol{V_1}$=[0, 0, 0, 0, w_{Wine,D_1}] and $\boldsymbol{V_2}$=[0, 0, $w_{barolo\ wine,D_2}$, $w_{red\ wine,D_2}$, w_{Wine,D_2}].

We shall now discuss the indexing process in more detail.

8.1. Word sense disambiguation: extended Viterbi

The Viterbi algorithm allows us to find the most likely sequence of hidden variables that results in a sequence of observed variables. In other words, starting from a sequence of n lemmas $\boldsymbol{L} \overset{\boldsymbol{def}}{=} \langle \boldsymbol{L_0}, \boldsymbol{L_1}, \ldots, \boldsymbol{L_{n-1}} \rangle$, the algorithm calculates the best sequence of concepts $\boldsymbol{C} \overset{\text{def}}{=} \langle \tilde{C}_0, \tilde{C}_1, \ldots, \tilde{C}_{n-1} \rangle$ and the best sequence of synsets $\boldsymbol{S} \overset{\text{def}}{=} \langle \tilde{S}_0, \tilde{S}_1, \ldots, \tilde{S}_{n-1} \rangle$.

Actually, Viterbi only works for first order HMMs. However, there is an extended algorithm for second order models (He, 1988) that is simple and fast enough for our experiments. In fact, given a certain L_t there are only a few S_t and C_t for which $p(L_t|S_t, C_t) \neq 0$, and only a few C_t for which $p(S_t|C_t) \neq 0$. In general, we define, for a given t, $\boldsymbol{S_t^{\text{eff}}}$ and $\boldsymbol{C_t^{\text{eff}}}$ to be the sets of synsets and concepts that map to L_t. Thus, the actual state space the algorithm must explore is much smaller than the theoretical one; this way the second order algorithm performs reasonably well for our goals.

Finally, notice that the actual input of the algorithm is a sequence of n information items $\boldsymbol{I_t}$; the sequence of lemmas is the observable input of the decoder, while the entire $\boldsymbol{I}$ is used for the transition probabilities. The algorithm is depicted in Appendix B.

The output of the decoding procedure (the list of concepts and the list of synsets) is stored in the conceptual index. We shall now present the model we adopted for the index.

8.2. Document representation: concept frequency analyzer

The conceptual index stores the lemma-synset-concept mappings, for each jth document, as a sequence of triples: $\langle (L_0, \tilde{S}_0, \tilde{C}_0), (L_1, \tilde{S}_1, \tilde{C}_1), \ldots, \rangle_j$. In order to support the ranking mechanism, each jth document is represented as a *vector of concept weights* $\boldsymbol{d_j} \overset{\text{def}}{=} (w_{1,j}, w_{2,j}, \ldots)$, similarly to the common vector space model (VSM) (Salton et al., 1975). Each weight $w_{i,j}$ in the vector refers to the ith concept of the domain ontology, while the vector size is given by the amount of concepts in the domain ontology. Notice that we do not consider the space of synsets for the VSM representation; the reason is that the space of concepts provides the most abstract representation possible.

For example, consider the following two documents: (1) “(...) red bottle (...) ruby wine (...)”; (2) “(...) red wine (...) ruby liqueur (...)”, and assume we map “red” and “ruby” to `red wine`, while “wine” and “liqueur” to `Wine`. For the sake of simplicity, assume that weights represent concept occurrences. Document (1) is represented by [`WineBottle`, `Wine`, `red wine`] = [1, 1, 1], while document (2) is represented by [`WineBottle`, `Wine`, `red wine`] = [0, 2, 2].

Notice that the word “red” in document (1) is not mapped to any concept, as it does not refer to a wine’s color. Moreover, notice that both words “ruby” and “red” are mapped to the same concept, and therefore contribute to its occurrence count; this effectively captures the fact that these two words represent, in their linguistic context, the same concept. The same holds for “wine” and “liqueur”.

[3] Notice that, in D_1, “Barolo” refers to a village and should not be considered.

However, the concept frequency analyzer does not just calculate weights as occurrences. In fact, it follows the TF-IDF schema:

$$w_{i,j} \overset{\text{def}}{=} tf_{i,j} \cdot idf_i; \quad tf_{i,j} \overset{\text{def}}{=} \frac{f_{i,j}}{\sum_i f_{i,j}}; \quad idf_i \overset{\text{def}}{=} \log \frac{N+1}{df_i} \tag{25}$$

where $w_{i,j}$ is the weight of the ith concept in the jth document; $tf_{i,j}$ defines the *frequency* of the ith concept in the jth document; idf_i represents the rarity of the ith concept inside the collection; $f_{i,j}$ is the number of occurrences of the ith concept in the jth document; N represents the number of documents in the collection; and df_i specifies how many documents contain at least one occurrence of the ith concept.

Such weights are stored in the conceptual index. Notice that, since the WSD procedure also calculates the most probable synsets to associate to lemmas, the index contains the lemma-synset-concept mappings. It also contains the linguistic context information and the full text of the documents. In particular, the synsets could be used for further calculations – for example, an application could leverage our engine to compute the distance between the words in a document collection, and by starting from the most probable synsets it might be easier to use the relationship structure of WordNet.

After the indexing procedure, the system is ready to be used. In particular, the User can submit her/his queries, as explained in Section 9, while the domain expert can supervise the extension procedure, and add new concepts to the domain model (see Section 10).

9. Querying documents and the domain ontology

Queries can be issued in several ways (see Fig. 10). In particular, we consider the following two: keyword-based and natural language-based querying. The former are composed of a sequence of words, optionally connected by Boolean logic operators. This kind of query is useful for documental search, wherein the user specifies the concepts to find. Translation from words to concepts can be done in many ways, as we shall see. The latter consist of questions or descriptions, and are submitted as natural language phrases. These queries can be used for both documental and conceptual search, where users can express a large variety of different requests.

9.1. Keyword-based queries

Keyword-based queries are composed of a sequence of words; these words can be connected by either AND or OR Boolean logic operators. In the first case, only documents containing all the concepts are returned; in the second case, documents containing at least one concept are returned.

As a preliminary step, the morphologic and POS tagger analyzer generates, for each word, the lemma with the highest probability. Then, *un-disambiguated keyword queries* search for all the concepts that can possibly be mapped to the lemmas, while *disambiguated keyword queries* just consider the meanings with the highest probabilities.

9.1.1. Un-disambiguated keyword queries

This is a simple, exhaustive approach, in which we simply avoid meaning disambiguation. The query engine retains, for each lemma L_i in the sequence of n lemmas, the set of associated concepts $\mathbf{C_i}$. Then, it executes the Cartesian product among the $\mathbf{C_i}$ sets, leading to a set composed of $\prod_i |\mathbf{C_i}|$ n-tuples. Thus, the system performs $\prod_i |\mathbf{C_i}|$ searches, one for each combination of concepts, and returns several ranking lists (each list is ranked separately). Finally, each list is presented to the user, according to its dimension (we assume that lists with many documents are more important than lists with fewer ones).

This approach is useful when the user specifies a single keyword (or maybe just a few), since the system simply returns a result list for each of the possible intended meanings. The approach does not scale well, however, since the number of different lists is the Cartesian product of the concepts sets that are associated with each word.

9.1.2. Disambiguated keyword queries

The query engine tries to select, for each lemma L_i in the sequence, the most probable concept C_i; the result is the con-

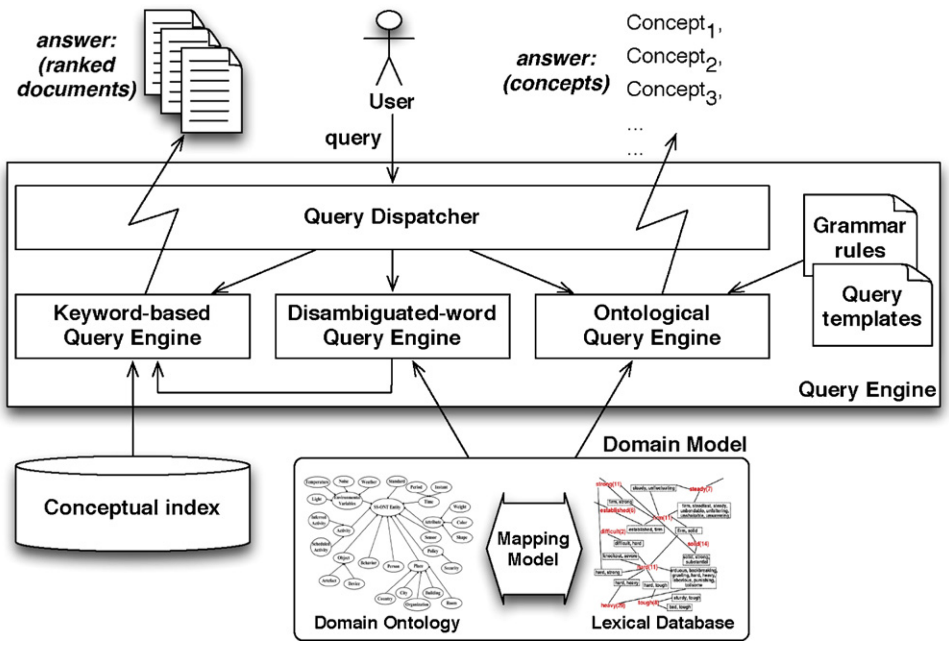


Fig. 10. Querying the system, with the query engine internal structure and its connections to the conceptual index and the domain model.

cept sequence $\boldsymbol{C}$. However, the approach described in Section 8 does not work in this case, since the given lemma sequence cannot be treated as a natural language sentence (it lacks the syntactic structure exhibited by natural language sentences; thus, our model cannot exploit the context and no decoding is possible).

Given a lemma L, without considering its context, the probability distribution of concepts C is:

$$p(C|L) = \frac{p(L|C) \cdot p(C)}{p(L)} = \frac{\sum_S p(L|S, C) \cdot p(S|C) \cdot p(C)}{\sum_{C'} \sum_S p(L|S, C') \cdot p(S|C') \cdot p(C')} \tag{26}$$

Consider, now, a sequence of lemmas $\boldsymbol{L} \overset{\text{def}}{=} \langle \boldsymbol{L_1}, \boldsymbol{L_2}, \ldots, \boldsymbol{L_m} \rangle$ connected by the AND Boolean operator. Let us make two assumptions: that there is linear independence among concepts (i.e., $p(C_j|\{C_{i \neq j}\}, \boldsymbol{L}) = p(C_j|\boldsymbol{L})$), and that the selection of the concept to map to the j-th word does not depend on other words (i.e., $p(C_j|\{L_{i \neq j}\}) = 0$). Both assumptions do not actually hold, but we argue that they represent a good approximation, given the fact that $\boldsymbol{L}$ contains a simple sequence of words and no phrase structure. Eq. (26) can then be simply extended to the sequence $\boldsymbol{L}$, and the best concept sequence $\boldsymbol{C}$ can be chosen by means of the following formula (notice that the calculation of the denominator of Eq. (26) can be avoided):

$$\boldsymbol{C} \overset{\text{def}}{=} \text{argmax}_{\boldsymbol{C}}\, p(\boldsymbol{C}|\boldsymbol{L}) = \text{argmax}_{\boldsymbol{C}} \prod_j p(C_j|L_j) \tag{27}$$

9.1.3. Query representation and enhancement

Once the keywords have been mapped to concepts, queries can be represented as vectors $\mathbf{q}_k = (w_{1,k}, w_{2,k}, \ldots)$, wherein each concept C_i is represented by weight $w_{i,k}$:

$$w_{i,k} \overset{\text{def}}{=} \left(0.5 + \frac{0.5 \cdot tf_{i,k}}{\max_l tf_{l,k}} \right) \cdot idf_i \tag{28}$$

The degree of similarity between document jth $\mathbf{d_j}$ and query kth $\mathbf{q_k}$ is given by the following well-known formula, used to calculate and rank the list of documents to return:

$$\text{sim}(\boldsymbol{d_j}, \boldsymbol{q_k}) \overset{\text{def}}{=} \frac{\mathbf{d_j} \cdot \mathbf{q_k}}{|\boldsymbol{d_j}| \cdot |\boldsymbol{q_k}|} = \frac{\sum_i^N w_{i,k} \cdot w_{i,j}}{\sqrt{\sum_i^N w_{i,k}^2} \cdot \sqrt{\sum_i^N w_{i,j}^2}} \tag{29}$$

In our model, however, one could exploit the structure of the domain ontology to enhance the query, and search – for example – for sub-classes, part-of classes, and individuals of a given class. In other words, searching for “Italy”, one could expect the system to find “Piemonte”, since the Domain Ontology knows it is one of Italy’s regions. In practice, once the system has mapped the keywords to their related concepts, it can start adding so-called *derived concepts*, according to the following rules:

- For each class in the query → add sub-classes, add part-of classes, add individuals of these classes
- For each individual in the query → add part-of individuals

Such concepts are *merged* with the concept they derive from. To understand why, consider the aforementioned query “Italy”. From the user’s point of view, and for the purpose of the query, the meanings of “Piemonte” and “Italy” overlap, since they simply represent two ways to lexicalize the idea of “italian territory”. Thus, one could merge the concepts `italy` and `piemonte` (and the related axes) into a new merged concept {`italy`, `piemonte`}.

The creation of this new concept implies the calculation of a related weight. In general, given a concept $\tilde{C}$ with weight w_c, and m derived concepts $\mathbf{C_d}$ with weights $\boldsymbol{w_d}$, the new merged concept $\tilde{C}'$ has weight w_c', which is calculated as a sum of vectors, which is then normalized to guarantee that $0 \leq w_c' \leq 1$. The result is the quadratic mean:

$$w_c' \overset{\text{def}}{=} \sqrt{\frac{w_c^2 + \sum_{d=1}^m w_d^2}{m+1}} \tag{30}$$

Thus, the expanded concept $\tilde{C}$ is replaced in the query vector by the new concept $\tilde{C}'$ with weight w_c'. By expanding each concept in the query we generate the new sequence $\boldsymbol{C}'$. Notice that the same process is performed, on the fly, on each document vector that the system needs to consider during the search process.

The conceptual index is searched for documents containing all the concepts (for AND), or at least one concept (for OR) in $\boldsymbol{C}'$; the resulting list is ranked using the aforementioned similarity measure, and shown to the user.

9.2. Natural language-based queries

We define two different kinds of natural language-based queries: disambiguated word queries and ontological queries.

9.2.1. Disambiguated word queries

Disambiguated word queries are issued as a natural language sentence. The system treats the issued phrase as a small document, and applies the same indexing techniques we presented in Section 8. The system, however, assumes that the goal of the query is to retrieve concepts that are part of the domain; therefore, during the indexing phase, it never associates the `out-of-context` concept to one of the query’s words (except, of course, for words that are only mapped to the `out-of-context` concept). The resulting set of concepts is then transformed into a vector q_k as explained in Section 9.1.3. For example, with query Q = “red wine”, the system could disambiguate the word “red” and map it to `redwine` (indeed it refers to a wine appellation, and not to a specific red wine); on the other hand, with query Q = “red Donnafugata Nero d’Avola” the word “red” should be mapped to `red` (see the domain ontology in Fig. 3).

Sometimes, users want to prevent the system from searching for specific words. For example, in the sentence “wine ruby color”, the word “wine” is needed to disambiguate the meaning of “ruby” and “color”; the user, however, does not want the system to discard documents where “wine” is not present. In other words, “wine” should be used in the disambiguation phase, but should not be used for document selection. The system provides a special syntax to allow this kind of query.

9.2.2. Ontological queries

Ontological queries are peculiar, since they allow us to search the domain ontology for concepts that meet a description or question. The methodology we have adopted is to describe the structure of the requests/questions that the system should recognize, as well as the queries to perform.

Notice that, since this description captures the *structure* of the phrase, and not the actual meaning, it is parametric in nature; for example, the sentences “what are the colors of wines?” and “what are the tastes of wines?” share the same structure, yet “colors” and “tastes” should be considered parameter instances.

The process is divided into two phases (see Fig. 11). In the first off-line phase the domain expert prepares the model by describing the phrase structures and the associated query templates. In the second phase the system recognizes the phrase, extracts the parameters, and prepares and executes the related query.

As an example of an ontological query, assume that the domain ontology contains information about the facts that wines have colors, and that Barolo’s color is red. Moreover, assume that the domain expert prepared the following phrase

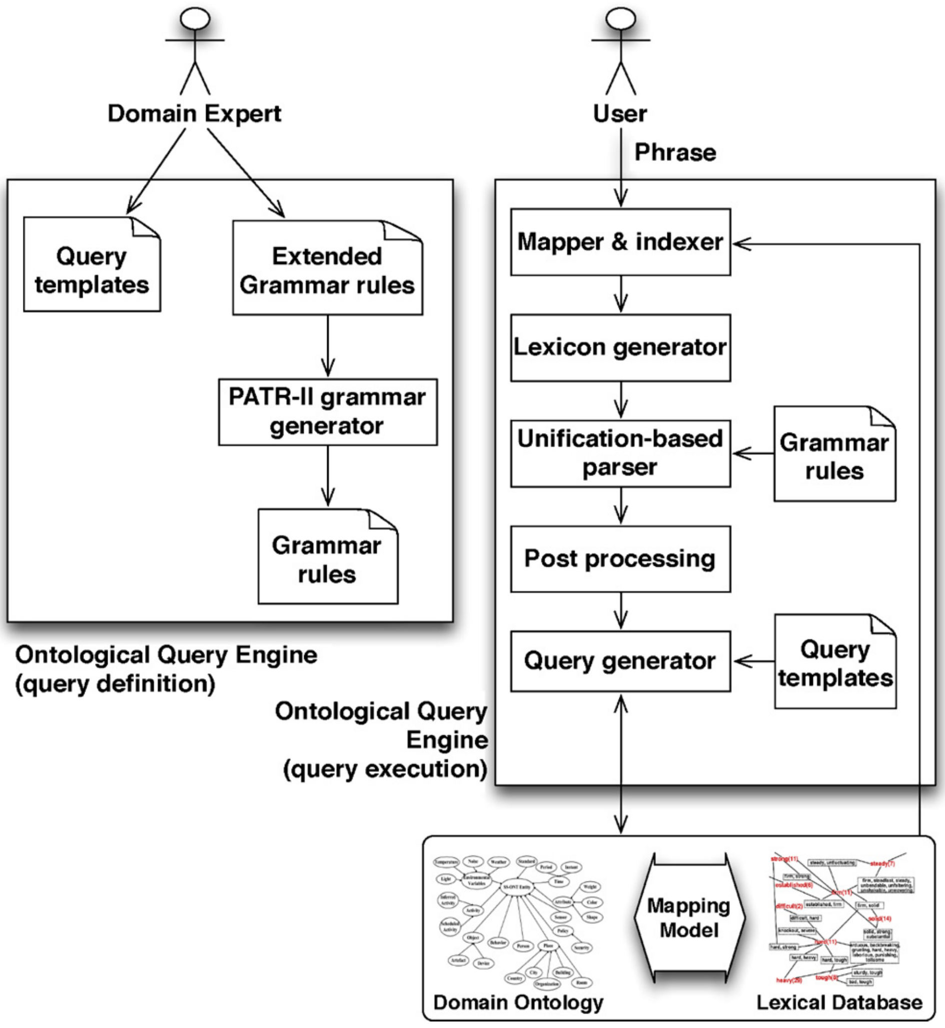


Fig. 11. Ontological queries: phrase description (left) and recognition (right).

template: TP= ⟨"what", wine-attribute-name, wine-instance⟩. If the user issues Q = "what is the color of Barolo?", the system extracts $\mathbf{C_Q}$ = {`WineColor`, `barolo wine`}, finds the phrase template TP, fills the placeholders ⟨"what", `WineColor`, `barolo wine`⟩, and executes the parametric query S_{TP} that has been associated to that template by the domain expert. In our prototype the parametric query is a SPARQL query and/or a SQL query (since individuals are stored in a DBMS).

Notice that the phrase template TP is actually able to capture several requests (e.g., the color of wine, the price of wine, etc.) In other words, requests with similar structure (in terms of sequences and types of concepts) can be captured by means of the same template. We shall now describe how we define and recognize ontological queries.

Query definition. In our approach we used a feature-based grammar to define phrase templates. In particular, we adopted the PATR-II language (Shieber, 1984), implemented by the PC-PATR unification-based syntactic parser (Shieber, 1986).

Feature-based grammars allow us to associate attributes with terminal symbols; the rules can then predicate on these attributes, and create complex conditions that affect their evaluation. As an example, the following PATR-II rule:

```
A = b c
  ⟨b rel⟩ = ⟨c rel⟩
  ⟨b rel⟩ = subj
```

which contains the nonterminal symbol A and terminal symbols b and c, only applies when the attribute `rel` of `b` equals the same attribute of `c`, and attribute `rel` of `b` has value `subj`; the grammar lexicon defines attributes and assigns them to terminal symbols.

We would like to use such approach to represent relationships between terminal symbols (words, in our case). For example, we would like to assert that a `subj` relationship exists between `b` and `c`. Unfortunately, the aforementioned example only asserts that `b` and `c` both have `rel` attributes, and that the attributed must have the same value `subj`. There is no easy way to assert that the *same* `subj` relationship should involve both `b` and `c`, and that the relationship should go from `b` to `c`. Thus, we decided to define an extended syntax for the semantics we needed. The following example, expressed in our own syntax:

```
@REQ2 = «Class» Class_1#
     @<Class_1 relprep_of> = <Class relprep_of> #
```

describes a rule that matches phrases composed of two words. Both words are mapped to classes (non-mapped words are ignored), and the first one represents the attribute we are searching for. The condition asserts that there must be a `relprep_of` relationship from the second word to the first one. Such extended rules are then translated into the standard PATR-II formalism; semantics lost during this translation will be enforced by means of a post-processing filter.

Query recognition. The Information Extraction Engine indexes the sentence issued by the user, just like it does with sentences taken from documents; the output consists of a sequence of concepts and the linguistic context information: $(\langle C_0, C_1, \ldots, \rangle, \boldsymbol{I})$. The Lexicon generator transforms each concept in a *lexical statement*, associating it with information $\boldsymbol{I}$ (e.g., the concept type, the lemma mapped to the concept, the lemma POS tag, relationships connecting the lemma to other lemmas, etc.). Such statements form the PATR-II *grammar lexicon*. Notice that, in contrast with the usual grammar usage, the lexicon is generated on-the-fly and contains only the lexical statements derived from the phrase being analyzed.

For example, the request "color of wines" produces the following lexicon:

```
\w color
\c Class
\f NN prep_of1 <id> =43 classIdNil
\w wines
\c Class
\f NNS prep_of1 <id> =56 classIdNil
```

where the field `\w` indicates a word, `\c` declares the concept type (two classes in the example), and `\f` lists the attributes of the word. For example, "wines" is a plural noun (`NNS`), it is involved in the `prep_of` relationship with id = 1 (this allows us to disambiguate the case where the same relationship is present multiple times), and has been mapped to the concept of id = 56. The `classIdNil` is a placeholder that, if the lemma were mapped to an individual, would be replaced with the id of the corresponding class. Notice that the lexicon does not indicate the "direction" of the `prep_of` relationship, since this is part of the semantics that cannot be expressed with the PC-PATR formalism, and that will be enforced through post processing.

The unification-based PC-PATR parser loads the lexicon and the rules, analyzes the sequence of concepts, and generates a set of parse trees. The parser's output then undergoes a post-processing analysis to enforce the semantics lost during the translation. Thus, parse trees generated by PC-PART that do not satisfy the full set of conditions asserted by our extended rules are discarded. At the end of this filtering phase, the retained trees (hopefully, just one tree) carry the query parameters (for the above example, the words that substitute the `Class` and `Class_1` symbols). These trees enter the query generator, which finds the related SQL template, fills the template placeholders with the parameters, and executes the query. The result is the set of concepts that answers the user's request.

10. Extending the domain model

A domain model can be extended by adding new individuals and relationships to its ABox and RBox.

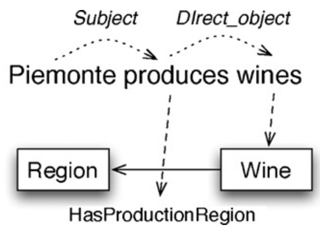


Fig. 13. Extending the domain ontology: an example.

As the system discovers new concepts, the probability distributions and the conceptual index must be updated. This is achieved on-demand, in batch mode, and is followed by probability distribution adjustments and a new indexing process.

As shown in Fig. 12, the domain model extender relies on Drools, a production-rules engine. The rules predicate on $\boldsymbol{I}$, the information collected while indexing the text; they infer that a given word could be a new individual of some class, or a new instance of some relationship. However, rules cannot guarantee that the newly discovered concept is correct. This is why the system tags new concepts as "guessed", to distinguish them from concepts that were defined by the domain expert.

Fig. 13 depicts a simple example that could be captured by means of a Drools rule. The sentence is "Piemonte produces wines". The dependency parser finds that "Piemonte" is the sentence's subject and that "wines" is the direct object. Assuming that the domain ontology contains information about the fact that wines have production regions, the information extraction engine maps "wines" to `Wine` and "produces" to `HasProductionRegion`. Defining a simple rule that asserts:

```
IF subject_of($W_1$, "to produce") $\wedge$
  direct_object_of($W_2$, "to produce") $\wedge$
  mapped_on_class($W_2$, Wine) $\wedge$
  mapped_on_relationship("to produce", $R$) $\wedge$
  connected_by_relationship($R$, Wine, $C$)
THEN
  is_a_new_individual_of($W_1$, $C$)
```

the system infers that W_1 = "Piemonte" should generate a new individual `piemonte` belonging to the class C = `Region`.

Notice that the system must ensure that $\sum_C p(C) = 1$ still holds. Thus, if $\boldsymbol{C}$ represents the old set of concepts, and $\boldsymbol{C_n}$ is the set of newly discovered concepts, the system must calculate new prior probability values as:

$$p(C_n) = \frac{1}{|\boldsymbol{C}| + |\boldsymbol{C_n}|}, \; \forall C_n \in \boldsymbol{C_n} \tag{31}$$

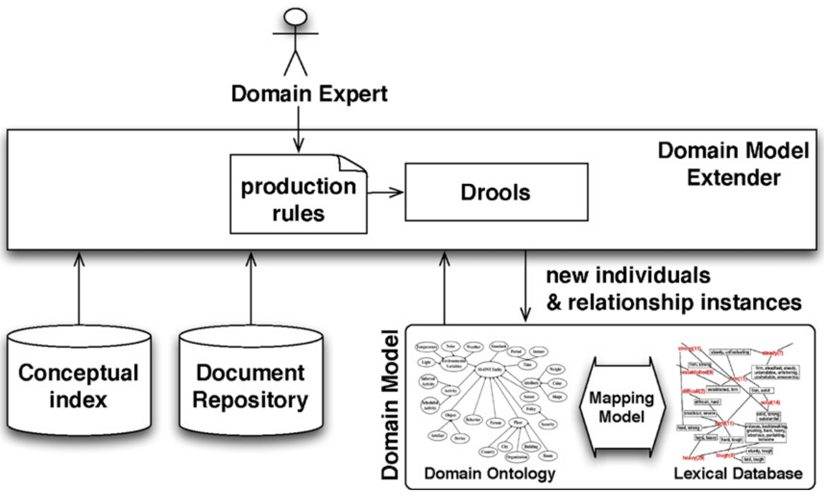


Fig. 12. The domain model extension process.

and update the prior probabilities of the old concepts as:

$$p^{\text{new}}(C) = p^{\text{old}}(C) \cdot \frac{|\boldsymbol{C}|}{|\boldsymbol{C}| + |\boldsymbol{C}_\mathbf{n}|}, \forall C \in \boldsymbol{C}_\mathbf{n} \tag{32}$$

If L_n is the lemma mapped to C_n, then, for concept C_n, the system must assign probability to each of the synsets that contain L_n (if any). Thus, calling $\boldsymbol{S}''$ the set of synsets containing L_n:

$$\begin{cases} p(S''|C_n) = 1/|\boldsymbol{S}''|, \forall S'' \in \boldsymbol{S}''; & \boldsymbol{S}'' \neq \emptyset \\ p(S_n|C_n) = 1; & \boldsymbol{S}'' = \emptyset \end{cases} \tag{33}$$

where S_n is a new synset, that is added when there is no synset that contains L_n. Probability distribution $p(L_n|S'', C_n)$ follows a similar rule:

$$\begin{cases} p(L_n|S'', C_n) = 1/|\boldsymbol{S}''|, \forall S'' \in \boldsymbol{S}''; & \boldsymbol{S}'' \neq \emptyset \\ p(L_n|S_n, C_n) = 1; & \boldsymbol{S}'' = \emptyset \end{cases} \tag{34}$$

11. The prototype: evaluation and results

DIESIRAE, the prototype system we developed, allowed us to conduct a set of tests on our approach. We developed a domain ontology for the "wine" domain composed of 70 classes, 45 relationships, and 449 individuals; the domain ontology concepts were mapped to 270 WordNet synsets.

The document repository was populated with several HTML pages, gathered from RSS feeds. The topic of some of these feeds was "wine", and thus they provided on-topic documents; other feeds were intentionally chosen to provide off-topic documents, so that we could check the system ability to discard words whose meanings do not match the domain ontology. We collected 124 HTML documents: 74 on-topic, from websites specialized on wine; and 50 off-topic, from Wikipedia and other websites. The latter contained words belonging to the "wine" vocabulary, but that actually carrying different meanings (for example, colors, fruits, tastes, etc.). In total, we gathered 70,504 words.

This is not a huge dataset, but nevertheless it allowed us to quickly experiment with our model, and to try several templates and parameters. Moreover, the data was significant enough to emphasize the advantages of our approach with respect to traditional techniques. Furthermore, using standard resources, like TREC, was not a feasible option, since the ArtDeco project forced us to focus on a very specific domain (wine), and our model required an ad hoc tagging to the dataset.

The evaluation explained in Sections 11.1 and 11.2 was performed during the feature template selection phase (see Section 6.2.3). The best performing model was then selected (data in Section 11.2 refers to that model) and compared against Lucene (see Section 11.3).

11.1. Validating the domain model: training set and test set

As a method for cross-validating our domain model, we adopted a repeated, random, sub-sampling validation procedure. We started by randomly selecting 10 training set/test set pairs from the 124 documents. Each train set contained an average of 29,718 samples (with standard deviation S = 684), representing 70% of the samples; each test set contained an average of 13,010 samples (S = 761), representing 30% of the samples. The training sets contained, on average, 346 concepts (17 classes and 329 individuals), while the test sets contained 218 concepts (16 classes and 202 individuals).

We defined the *degree of ambiguity* of a word $d_w \overset{\text{def}}{=} |\mathbf{C}_\mathbf{W}^\mathbf{eff}|$ as the number of different concepts that, in the training set, were associated to that word (notice that if W is not part of the domain, it is mapped to the `off-topic` concept, and therefore $d_w = 1$; for all other words, $d_w \geq 1$).

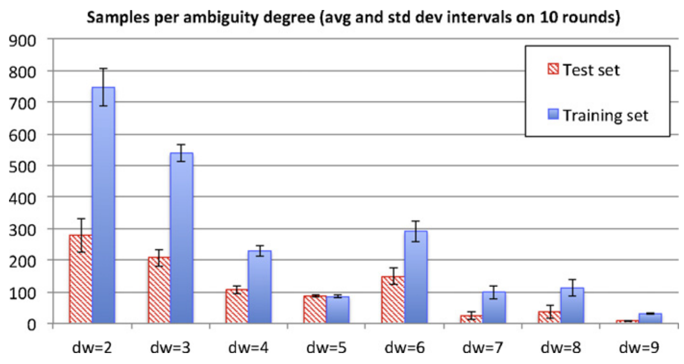


Fig. 14. Training and test samples.

In the training and test sets, the majority of words (most of them being off-topic words) had $d_w = 1$; however, we ignored these words – all the following indexes were calculated for words with $d_w > 1$. Fig. 14 shows the average number of samples, per ambiguity degree, for the training set and the test set.

11.2. Validating the domain model: training and evaluation

We trained the ME-2L-HMM2 mapping model, which is the core of our domain model, using the feature templates defined in Section 6.2.3 and the following constants: k_{spec} = 0.8, k_{count} = 1, *maxError* = 0.1, and *avgError* = 0.03. On average, the trigram, bigram, and unigram models generated, respectively, 82,952 (S = 3777), 64973 (S = 2826), and 46,543 (S = 1834) feature instances; Fig. 15 shows the number of instances, per feature template, generated by the models.

We applied the model on the test set at the end of each training session, and calculated a set of indexes in order to guess the "fitness" of the generated model.

We defined a_{i,d_w} as the number of words of the test set, with ambiguity d_w, that correctly mapped to the ith concept appearing in the test set; N_{c,d_w} as the number of concepts of the test set involved in mappings with words having ambiguity d_w; and N_{w,d_w} as the number of words in the test set having ambiguity d_w.

Then, we defined the accuracy AC_{d_w} for the d_w ambiguity as:

$$AC_{d_w} \overset{\text{def}}{=} \frac{\sum_i^{N_{c,d_w}} a_{i,d_w}}{N_{w,d_w}} \tag{35}$$

and, by training and testing the model with 10 different training set/test set pairs, we obtained the average accuracy per ambiguity degree $MAC_{d_w} \overset{\text{def}}{=} \sum_i^{10} AC_{d_w}^{(i)}/10$ (see Fig. 16).

Defining N_w as the number of words in the test set s.t. $d_w > 1$ and N_{d_w} as the number of ambiguity classes, the global accuracy AC

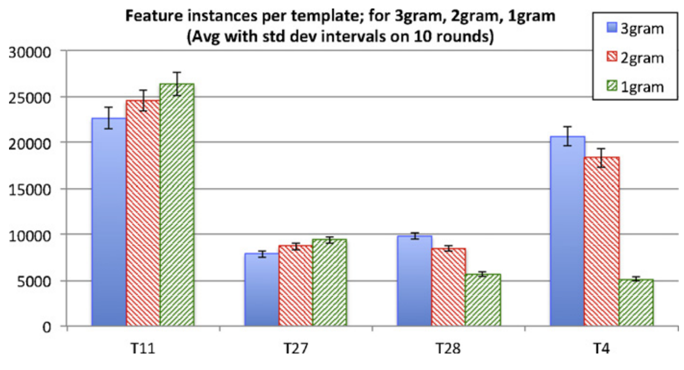


Fig. 15. Feature instances.

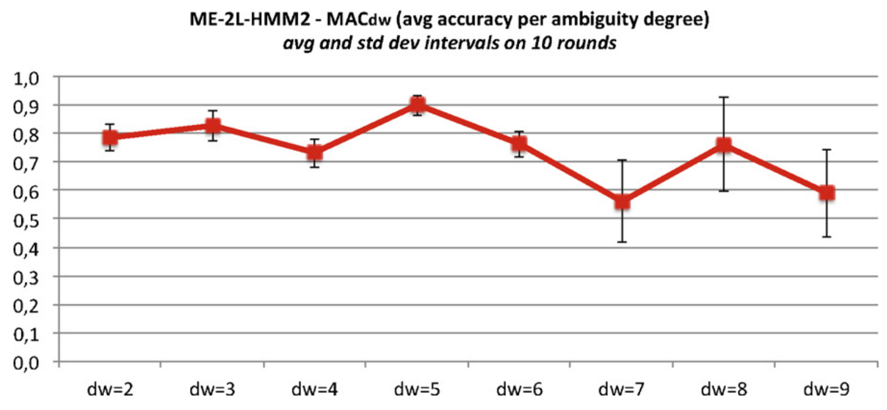


Fig. 16. Average accuracies for our ME-2L-HMM2.

(the probability for a given word to be mapped to the right concept) is defined as a weighted mean:

$$AC \overset{\text{def}}{=} \frac{\sum_{d_w}^{N_{d_w}} AC_{d_w} \cdot N_{w,d_w}}{N_w} \tag{36}$$

and, by training and testing the model with 10 different training set/test set pairs, we obtained the average global accuracy $MAC \overset{\text{def}}{=} \sum_{i}^{10} AC^{(i)}/10 = 0.79$ ($S = 0.037$).

To understand whether this accuracy can be considered good, we compared the ME-2L-HMM2 against a classifier (called the no-context model) that, given a lemma L from the test set, chooses a concept C_{nc}, among the concepts that are mapped to L in the test set, s.t: $C_{nc} = \text{argmax}_{C \in \mathbf{C}_L^{\mathbf{eff}}} p(C|L)$, wherein $p(C|L)$ is defined as in Eq. (26). The no-context model is actually a reduced version of our model, where the transition probabilities have been removed, and can be thought of as a sort of Naïve Bayes approach. Accuracies AC_{nc,d_w}, MAC_{nc,d_w}, AC_{nc}, and MAC_{nc} are defined as in the equations above. By relying on the test set (and, just like for AC, ignoring words with $d_w = 1$), we were able to obtain an average global accuracy $MAC_{nc} = 0.60$ ($S = 0.03$). Fig. 17 shows the average accuracy, per ambiguity degree.

Fig. 18 shows the increment in average accuracy $\Delta MAC_{d_w} \overset{\text{def}}{=} MAC_{d_w} - MAC_{nc,d_w}$, per ambiguity degree. The significance of such values was evaluated with the P-value (see Appendix D); the P-values resulted $P(t, N = 10) < 0.01$ (i.e., highly significant) for all the ambiguity degrees, except $d_w = 2$, whose $P(t, N = 10) = 0.053$, and $d_w = 7$, whose $P(t, N = 10) = 0.081$. The significance of the increment in global accuracy $\Delta MAC \overset{\text{def}}{=} MAC - MAC_{nc} = 0.19$ can be calculated using the same approach, and leads to $P(t, N = 10) < 0.01$ (see Appendix D.1 for details).

Given these results, we can conclude that our ME-2L-HMM2 definitely bests the No-context model for d_w between 3 and 6. For $d_w = 2$ and $d_w = 7$, the two models seem to have similar performances, but the comparison is not statistically significant and there is not a clear winner; in particular, for $d_w = 7$ the two models perform poorly, and exhibit a big S (especially for the No-context model), which is a hint that the sample set was not good enough for this ambiguity degree. For $d_w > 7$, ME-2L-HMM2's increment in accuracy is statistically significant but the big S is a hint that the training set was barely sufficient. Calculating the accuracy as in Eq. (36), but using the "best range" ambiguity degrees $2 \le d_w \le 6$ (which accounts for 94% of the test samples having $d_w > 1$), leads to $MAC_{2:6} = 0.81$ ($S = 0.032$) and $MAC_{nc,2:6} = 0.59$ ($S = 0.008$).

11.3. Comparing DIESIRAE against a pure TF-IDF search engine

In order to evaluate the effectiveness of our approach, we used the ME-2L-HMM2 with the best AC ($AC = 0.83$ and $AC_{2:6} = 0.84$), among the 10 models we generated, as the mapping model inside the DIESIRAE prototype, and compared our results to Lucene (McCandless et al., 2010), a pure TF-IDF search engine. We used our disambiguated keyword-based queries (see Section 9.1.2), since they allowed us to use exactly the same search strings both in our system and in Lucene, ensuring the comparison was significant.

11.3.1. Experimental setting

The search engine built with Lucene used a simple TF-IDF, with stemming; the algorithm did not index the titles of the documents,

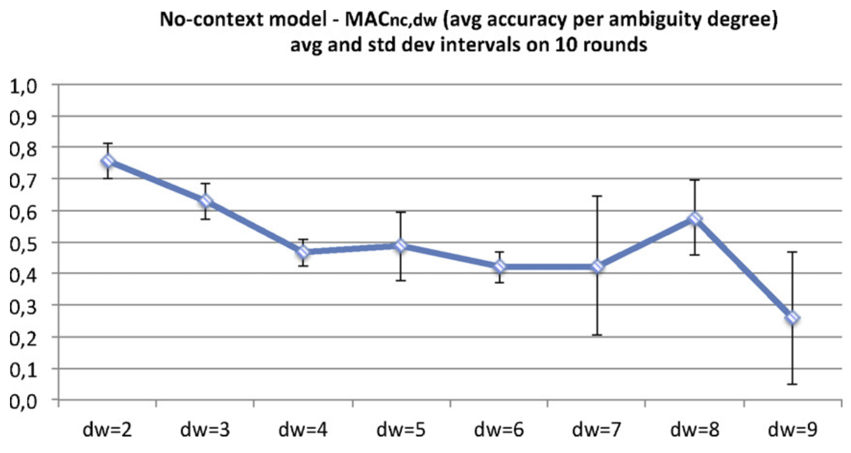


Fig. 17. Average accuracies for the no-context model.

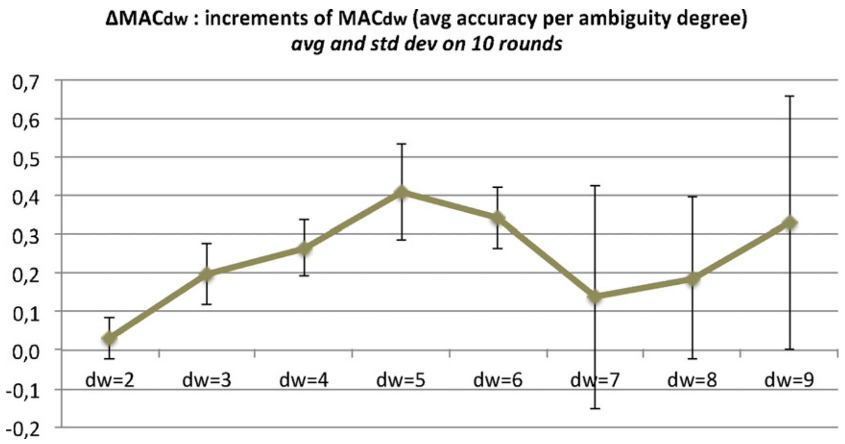


Fig. 18. Increment in average accuracies.

just like our system does not.[4] DIESIRAE was set to use the disambiguated word queries (see Section 9.2.1). The experimental setting involved:

- Document set: the test set used to evaluate the domain model.
- Relevance judgment: five human raters; concordance among them was evaluated with the K_{free} index (Randolph, 2005); see Appendix E for details.
- The same 25 queries for both engines.
- For each engine and for each query, we collected the top-k documents, where $k \overset{\text{def}}{=} \min(\text{length}(\text{listofdocuments}), 10)$.
- For each of the 25 queries, the two result lists were merged, eliminating duplicate documents; the length of such a mixed list is the *combined query cardinality* k_q; notice that $1 \le k_q \le 20$.
- To avoid any kind of bias in raters' evaluations, there was no way to associate documents in the combined result list with the engine that retrieved them; moreover, the list was ordered by document ID (not by ranking) and the raters were explicitly notified of this.
- Metrics:
  - *Precision*: Precision, for a given query q (Pr_q); mean precision (MPr), which is the average of Pr_q calculated on the whole set of 25 queries.
  - *Information gain* (calculated using k_q, see Appendix C.1 for details): Normalized discounted cumulative gain ($NDCG_q$), which measures, for a given query q, the total information gain accumulated at rank k_q; $MNDCG$, which averages $NDCG_q$ on all the queries.
  - *Engine comparison* (see Appendix C.2): Delta mean precision (ΔMPr); percentage delta in mean precision ($\%\Delta MPr$), which represents the percentage average precision increment of DIESIRAE over Lucene; percentage delta in mean recall ($\%\Delta MRe$), which represents the percentage of average recall increment of DIESIRAE over Lucene; percentage delta in mean fallout ($\%\Delta MFa$), which represents the percentage of average fallout decrement of DIESIRAE over Lucene; and the increment in information gain $\Delta MNDCG$.
- Significance test (see Appendix D.2 for details): P-values on ΔMPr and $\Delta MNDCG$.

[4] We are aware that titles carry useful hints about the topics of documents, and often search engines give them a special weight in their TF-IDF formulas. Our goal, however, was to evaluate the effectiveness of our approach, without making use of such tricks.

Table 2
Accuracy of the ME-2L-HMM2.

MAC	MAC_{nc}
ME-2L-HMM2 vs No-context	
0.79 (S = 0.037)	0.60 (S = 0.030)
$P(t, N = 10) < 0.01$	
Best ME-2L-HMM2	
AC	0.83

Each rater evaluated the combined result list, for each query. Raters read the text of the returned documents and ranked the relevance using a 0–5 scale (0 = irrelevant); the relevance level *rel* of a document was then calculated as the average of such ranks. The aforementioned indexes were calculated considering the documents with $rel \ge 3$ to be *relevant*.

A second experiment leveraged the fact that each document in the collection is tagged as on-topic or off-topic; such tagging can be thought of as a synthetic rater that gives a binary evaluation on how well the two engines are able to ignore off-topic documents. In this case, the performance indexes were calculated considering the on-topic documents to be *relevant*.

11.3.2. Evaluation and discussion

Table 2 recaps the data collected on ME-2L-HMM2's accuracy. The mean accuracy is good, and the small S confirms the robustness of the approach. The gap between ME-2L-HMM2's and no-context's accuracies is a clear evidence that by exploiting the linguistic context information of the words we are able to increase the effectiveness of the system during the WSD task.

Table 3 summarizes the most important evaluation indexes for the experiment with the five human raters; a full list can be found in Appendix C.2, Table C.6. The data confirmed that our approach provides better precision than TF-IDF. Absolute precision values were not exceptionally good, but we argue this was mainly due to the document set, which was – for the most part – composed of wine reviews; thus, the document set was probably too homogeneous, making it difficult to select the "right" set of documents for the queries we inserted.

The experimental setting we chose did not allow us to calculate absolute recall and fall-out indexes, but, on the other hand, we were mostly interested in evaluating the *gap* between our approach and the TF-IDF method. The data confirmed that our system provides better recall and fall-out than TF-IDF, when considering the top-k result lists.

Table 3
Experimental indexes.

MPr_{DIE}	MPr_{Luc}	$MNDCG_{DIE}$	$MNDCG_{Luc}$
DIESIRAE vs Lucene – experiment with human raters			
0.61	0.41	0.92)	0.81
($S = 0.33$)	($S = 0.33$)	($S = 0.10$)	($S = 0.23$)
$P(t, N = 30) < 0.01$		$P(t, N = 30) < 0.01$	

%ΔMPr	%ΔMRe	%ΔMFa
50.38%	43.86%	−31.58%

Human raters concordance	
K_{free}	0.43

MPr_{DIE}	MPr_{Luc}	$MNDCG_{DIE}$	$MNDCG_{Luc}$
DIESIRAE vs Lucene – experiment with synthetic rater			
0.89	0.54	0.89	0.74
($S = 0.24$)	($S = 0.40$)	($S = 0.12$)	($S = 0.27$)
$P(t, N = 30) < 0.01$		$P(t, N = 30) < 0.01$	

%ΔMPr	%ΔMRe	%ΔMFa
64.23%	52.94%	−83.30%

The information gain index indicated that our approach also bests the TF-IDF in ranking the result lists: the most relevant documents tend to appear at the beginning of the ranked lists.

Finally, the *P*-values we calculated on the aforementioned indexes indicated that the results we obtained were statistically significant.

Table 3 also shows the concordance among the five raters, calculated with Randolph's multi-rater K_{free}. The index assumes values in $[-1, 1]$ (where 1 means perfect agreement, 0 means that agreement is equal to chance, and −1 means "perfect" disagreement). The agreement level we measured was reasonably – but not exceptionally – good, confirming that the document set was probably not so easy to evaluate.

The results we obtained from the experiment with the synthetic rater are shown in Table 3. A full list can be found in Appendix C.2, Table C.7. The results indicated that our system definitely did a better job than Lucene in filtering out the off-topic documents from the result lists.

To summarize, the data we collected confirmed that out approach provides an effective technique for the WSD problem, and that it allows us to implement a search engine that clearly bests the TF-IDF approach.

11.3.3. *Performance gain removing the partition function*

As described in Section 6.2.2, we do not compute the partition function of the MaxEnt models. To understand the impact on decoding performance, we executed the indexing procedure ten times; then, we ran a modified indexing procedure, which computed the MaxEnt models using the partition function. The results showed that our approach was consistently 30% faster.

Since the test set was composed of 94% of words with $d_w = 1$ (mostly due to off-topic words), we argue that the speedup we obtained is significant and should provide even better results when applied on texts with a higher percentage of ambiguous words. More importantly, the time complexity of the MaxEnt model computation does not depend on word ambiguity (see Section 6.2.2), and this removes a strong scalability issue associated with traditional approaches.

12. Conclusions

We presented a model for knowledge extraction from natural language documents. In our model, we use a domain ontology to model the domain knowledge, while a lexical database represents the domain vocabulary. ME-2L-HMM2, an HMM-inspired, MaxEnt-enhanced stochastic model, stores the mappings between these two levels. A second order Viterbi decoder disambiguates documents' words during the indexing phase. Both keyword-based and natural language-based queries are supported by the system, which is able to search both the document collection and the domain ontology. Finally, a rule-based component allows us to discover new individuals in the document collection, and to add them to the domain ontology.

Compared with other popular stochastic models (like HMM, MEMM, and CRF), our approach is simpler, yet powerful enough to allow a second-order model to be efficiently implemented.

ME-2L-HMM2 significantly outperforms the simpler no-context classifier, especially for words with high ambiguity. The DIESIRAE prototype significantly outperforms pure TF-IDF engines.

For future work we plan to: adopt a multilingual lexical database; refine the model, applying smoothing techniques to the MaxEnt models (Toutanova et al., 2003); test new feature templates, as well as new feature filter algorithms; and investigate the adoption of standard frameworks, like UIMA (Ferrucci et al., 2006). Finally, we plan to further evaluate the model; in particular, an experiment with a group of human taggers will allow us to place an upper-bound on the accuracies, and understand how well our model compares with such a gold standard. Moreover, the processes behind the ontological queries and the domain model extension module will be evaluated.

Appendix A. The GIS algorithm

The algorithm shown in Fig. A.19 presents the GIS training procedure, adapted from Goodman (2002). Variable *observed*[*i*] contains the observed (i.e., calculated from the training set) value of the *i*th feature, and *expected*[*i*] is the value of the *i*th feature calculated by the current model; the slowing factor f_{max_active} counts the number of active (i.e., non-zero) features. Notice that the two innermost loops only consider concepts in $\mathbf{C}_t^{\mathbf{eff}}$ as, for each given time *t*, this set contains all the candidate concepts; if we call $|\mathbf{C}^{\mathbf{eff}}|$ the mean size of the set $\mathbf{C}_t^{\mathbf{eff}}$, the temporal (mean) complexity of a single algorithm loop is: $O(|\boldsymbol{TR}| \cdot |\mathbf{C}^{\mathbf{eff}}| \cdot |\boldsymbol{F}|)$.

The algorithm sets weights so that features $f_i(C, \boldsymbol{X})$, calculated on the distribution observed in the training set, and calculated on the distribution generated by the MaxEnt model, give the *same* expectation (see Eq. (24)). We actually relaxed this condition, introducing *maxError*, a threshold that specifies the maximum, fractional error allowable on expectations, and *avgError*, a threshold that specifies the average, fractional error allowable on expectations; these thresholds are used as termination conditions for the algorithm (we used *maxError* = 0.1 and *avgError* = 0.003).

Appendix B. The extended Viterbi algorithm

The algorithm shown in Fig. B.20 depicts the second order Viterbi algorithm (He, 1988), extended to manage our ME-2L-HMM2 model. Log-probabilities of MaxEnt models are calculated as:

$$\log p^{\diamond}(C_t|C_{t-1}, \boldsymbol{I}) = \sum_i^n \gamma_i f_i(C_t, C_{t-1}, \boldsymbol{I}) - \sum_i^m \psi_i f_i(C_{t-1}, \boldsymbol{I}) \tag{B.1}$$

```
**Input**: $t_0$; $\boldsymbol{X}_t$; $\boldsymbol{F}$.
**Output**: $\boldsymbol{\Theta} = \theta[1], \ldots, \theta[|\boldsymbol{F}|]$.
$observed[i] := \sum_t^{|\boldsymbol{TR}|} f_i(C_t, \boldsymbol{X}_t), \forall i = 1 \ldots |\boldsymbol{F}|$ ;
$\theta[i] := 1, \forall i = 1 \ldots |\boldsymbol{F}|$ ;
$\epsilon_{sum} := 0; \epsilon_{max} := 0; f_{max\_active} := 0$ ;
**repeat**
    $f_{max\_active} := 0$ ;
    $expected[i] := 0, \forall i = 1 \ldots |\boldsymbol{F}|$ ;
    **for** *training instance t* **from** $t_0$ *to* $|\mathbf{TR}|$ **do**
        $f_{active} := 0$ ;
        **for** *each output* $C \in \mathbf{C}_t^{\text{eff}}$ **do**
            $m[C] := \sum_i^{|\boldsymbol{F}|} \theta[i] \cdot f_i(C, \boldsymbol{X}_t)$ ;
        **end**
        $z := \sum_C e^{m[C]}$ ;
        **for** *each output* $C \in \mathbf{C}_t^{\text{eff}}$ **do**
            **for** *each i in* $|\boldsymbol{F}|$ *such that* $f_i(C, \boldsymbol{X}_t) > 0$ **do**
                $expected[i] \mathrel{+}= f_i(C, \boldsymbol{X}_t) \cdot e^{m[C]}/z$ ;
                $f_{active}$++ ;
            **end**
        **end**
        **if** $f_{active} > f_{max\_active}$ **then** $f_{active} := f_{max\_active}$ ;
    **end**
    **for** *each feature i in* $\boldsymbol{F}$ **do**
        $\theta[i] \mathrel{+}= \log(observed[i]/expected[i])/f_{max\_active}$ ;
        $\epsilon := |observed[i] - expected[i]|/observed[i]$ ;
        $\epsilon_{sum} \mathrel{+}= \epsilon$ ;
        **if** $\epsilon > \epsilon_{max}$ **then** $\epsilon_{max} := \epsilon$ ;
    **end**
**until** $\epsilon_{max} \le maxError \ \wedge \ \epsilon_{sum}/|\boldsymbol{F}| \le avgError$;
```

Fig. A.19. The GIS algorithm.

$$\log p^{\diamondsuit}(C_t|C_{t-1}, C_{t-2}, \boldsymbol{I}) = \sum_i^h \lambda_i f_i(C_t, C_{t-1}, C_{t-2}, \boldsymbol{I}) - \sum_i^n \gamma_i f_i(C_{t-1}, C_{t-2}, \boldsymbol{I}) \qquad \text{(B.2)}$$

while the emission probability, needed by Viterbi, is:

$$p(L_t|C_t) = \sum_{S \in \boldsymbol{S}_t^{\text{eff}}} p(L_t|S, C_t) \cdot p(S|C_t) \qquad \text{(B.3)}$$

and the synset probability, used to calculate $\tilde{S}_t$, is:

$$p(S|L_t, \tilde{C}_t) = \frac{p(L_t|S, \tilde{C}_t) \cdot p(S|\tilde{C}_t)}{p(L_t|\tilde{C}_t)} \qquad \text{(B.4)}$$

The algorithm makes use of the scaled transition distributions defined in Eqs. (14) and (15). These distributions do not break the algorithm, which still calculates the most likely concept sequence. The only shortcoming is that the algorithm is no longer able to calculate the true probability of the best sequence (because the probability is scaled by an unknown value). Our algorithm does not even return this value, since our model does not make use of it.

Moreover, our transition probability distributions, generated by the MaxEnt models, depend on linguistic context information $\boldsymbol{I}$, but, once again, Viterbi still works, since $\boldsymbol{I}$ is a constant.

If we call $|\boldsymbol{C}^{\text{eff}}|$ and $|\boldsymbol{S}^{\text{eff}}|$ the mean sizes of the sets $\boldsymbol{C}_t^{\text{eff}}$ and $\boldsymbol{S}_t^{\text{eff}}$, the temporal and spatial (mean) complexities of the algorithm are: $O(n \cdot |\boldsymbol{C}^{\text{eff}}|^3 \cdot |\boldsymbol{S}^{\text{eff}}|)$ and $O(n \cdot |\boldsymbol{C}^{\text{eff}}|)$. These complexities are usually small, since $\boldsymbol{C}_t^{\text{eff}}$ and $\boldsymbol{S}_t^{\text{eff}}$ are in general small sets. In fact, the performance of the decoding process in our prototype was satisfactory.

```
**Input**: The list of $n$ lemmas $L_t$, $\boldsymbol{C}$, $\boldsymbol{S}$, $\boldsymbol{I}$.
**Output**: The list of $n$ concepts $\tilde{C}_t$; the list of $n$ synsets $\tilde{S}_t$.
**for** $t$ **from** $0$ **to** $n-1$ **do**
    **for** *each concept* $C \in \mathbf{C}_t^{\text{eff}}$ **do**
        **if** $t = 0$ **then**
            $newLogProb := \log(p(C_t) \cdot p(L_t|C_t))$ ;
            $backPointer := 0$ ;
        **else if** $t = 1$ **then**
            $newLogProb := \max_{C_{t-1} \in \mathbf{C}_{t-1}^{\text{eff}}} (viterbiMatrix[C_{t-1}, t-1]+$
                $+ \log p^{\diamond}(C_t \mid C_{t-1}, \boldsymbol{I}) +$
                $+ \log p(L_t|C_t))$ ;
            $backPointer := \operatorname{argmax}_{C_{t-1} \in \mathbf{C}_{t-1}^{\text{eff}}}(viterbiMatrix[C_{t-1}, t-1]+$
                $+ \log p^{\diamond}(C_t \mid C_{t-1}, \boldsymbol{I}) +$
                $+ \log p(L_t|C_t))$ ;
        **else**
            $newLogProb := \max_{C_{t-1} \in \mathbf{C}_{t-1}^{\text{eff}}, C_{t-2} \in \mathbf{C}_{t-2}^{\text{eff}}} (viterbiMatrix[C_{t-1}, t-1]+$
                $+ \log p^{\diamond}(C_t \mid C_{t-1}, C_{t-2}, \boldsymbol{I}) +$
                $+ \log p(L_t|C_t))$ ;
            $backPointer :=$
            $\operatorname{argmax}_{C_{t-1} \in \mathbf{C}_{t-1}^{\text{eff}}}\Big( \max_{C_{t-2} \in \mathbf{C}_{t-2}^{\text{eff}}} (viterbiMatrix[C_{t-1}, t-1]+$
                $+ \log p^{\diamond}(C_t \mid C_{t-1}, C_{t-2}, \boldsymbol{I}) +$
                $+ \log p(L_t|C_t))\Big)$ ;
        **end**
        $viterbiMatrix[C_t, t] := maxLogProb$ ;
        $backPointerMatrix[C_t, t] := backPointer$ ;
    **end**
**end**
$bestState := \operatorname{argmax}_x(viterbiMatrix[x, n-1])$ ;
**for** $t$ **from** $n-1$ **to** $0$ **do**
    $\tilde{C}_t := bestState$ ;
    $bestState := backPointerMatrix[bestState, t]$ ;
    $\tilde{S}_t := \operatorname{argmax}_{S \in \mathbf{S}_t^{\text{eff}}}(p(S|L_t, \tilde{C}_t))$ ;
**end**
```

Fig. B.20. The extended, second order Viterbi.

However. we are considering testing more sophisticated and efficient algorithms, like the A* decoding (Jelinek, 1969; Jurafsky and Martin, 2000), for dealing with the (rare) situations where $\boldsymbol{C}_t^{\text{eff}}$ and $\boldsymbol{S}_t^{\text{eff}}$ are big (Table B.4).

Appendix C. Evaluation indexes

C.1. Information gain

The following indexes on information gain, taken from Croft et al. (2009), require that, for any given query, the two engines retrieve the same number of results. It was often the case, however, that DIESIRAE's queries would return fewer results that Lucene. Moreover, for some queries, Lucene and/or DIESIRAE would return less than 10 results. To solve this issue, we added "dummy" entries to the two result lists, so that their lengths would be equal to the combined query cardinality k_q. For each of the two result lists, the relevance associated to each "dummy" entry was calculated as $5 - r$, where r is the average relevance associated, in the other list, to the missing documents.

Table B.4
Queries. Keywords issued in Lucene, with their ambiguity degrees, and the equivalent sentence used in DIESIRAE.

q	Meaning	Lucene keywords	DIESIRAE phrase
1	red wines[a]	wine red colour wine red	wine red colour
2	wines with fruity aroma[a]	wine fruit aroma fruit aroma	wine fruit aroma
3	wine aromas	wine aroma	wine aroma
4	wine acidity typologies	acidity	acidity
5	countries producing wine	countries wine	countries producing wine
6	oaky wines	wine oak	wine oak
7	bordeaux wines	bordeaux wine	bordeaux wine
8	wines sweet on the nose	wine sweet nose	wine sweet nose
9	white wines	wine white color	wine white color
10	liqueur, considered as a synonym of wine	liqueur	liqueur
11	wine selection	selection	selection
12	apple, as a wine aroma or taste	apple	apple
13	chocolate, as a wine aroma or taste	chocolate	chocolate
14	wine aroma	aroma	aroma
15	california	california	california
16	chardonnay wine	chardonnay wine	chardonnay wine
17	bordeaux wine	bordeaux	bordeaux
18	state (as a synonym of country) producing wine	state	state
19	coffee, as a wine aroma or taste	coffee	coffee
20	floral, as a wine aroma or taste	floral	floral
21	cherry, as a wine aroma or taste	cherry	cherry
22	red wine	red wine	red wine
23	acid aroma	acidity aroma	acidity aroma
24	floral wine taste	floral taste	floral taste
25	floral wine aroma	floral aroma	floral aroma

[a] Lucene returned very few results; the query was re-issued, deleting some keywords, in an attempt to increase the recall.

Table C.5
Relevance assignment.

Lucene rank ID list	DIESIRAE rank ID list	ID list shown to raters	Rater X's relevance assessment	Rater X's relevance assigned to Lucene ranked list	Rater X's relevance assigned to DIESIRAE ranked list
6	34	6	1	6 →1	34 → 5
34	26	26	4	34 →5	26 →4
28	300	28	0	28 →0	300 →4
75		34	5	75 →1	**6** →3.63
50		50	4	50 →4	**28** →3.63
82		75	1	82 →2	**75** →3.63
240		82	2	240 →2	**50** →3.63
103		103	0	103 →0	**82** →3.63
340		240	2	340 →1	**240** →3.63
26		300	4	26 →4	**103** →3.63
		340	1	**300 →1**	**340** →3.63

Table C.5 shows an example of this approach, where Lucene returns a ranked list composed of 10 items, while DIESIRAE returns a ranked list of 3 items. The raters are presented with a combined list, ordered by document ID, composed of 11 unique IDs. The relevance values in the list were added by rater *X*. The document present in Lucene's list that is no in the mixed list is document 300, and its *r* is 4; thus, the “dummy” result 300 is added to the bottom of Lucene's ranked list, with relevance 5 – 4 = 1. The documents in DIESIRAE's list that are not present in the mixed list are <6, 28, 75, 50, 82, 240, 103, 340>, and their average relevance *r* is (1 + 0 + 1 + 4 + 2 + 2 + 0 + 1)/8 = 1.375; thus, the fake results ⟨6, 28, 75, 50, 82, 240, 103, 340⟩ are added to the bottom of the DIESIRAE's list, with relevance 5 – 1.375 = 3.625.

Let us explain our approach with the following example. If two documents, ID$_1$ and ID$_2$, in the Lucene's list are assigned with, say, relevance 4 (“quite relevant”) and 5 (“highly relevant”), the *absence* of such a document in DIESIRAE's list should be credited with, respectively, 1 and 0 (after all, the rater assessed that these particular documents should be in the list). The problem is where in DIESIRAE's ranked list should those “dummy” entry IDs be placed. We chose to put these missing documents at the bottom of the list, and to assign the same average relevance value to both of them. This way, the relative order between ID$_1$ and ID$_2$, in the DIESIRAE's ranked list, is irrelevant for the calculation of the evaluation indexes.

Once we have modified the ranked list, we define DCG_q as the discounted cumulative gain at rank k_q for a given query *q*. This measures the usefulness, or gain, from examining a document. The gain is accumulated starting at the top of the ranking and is discounted at lower ranks. DCG_{q,k_q} is the total gain accumulated at a particular rank k_q:

$$DCG_q \overset{\text{def}}{=} rel_{q,i} + \sum_{i=2}^{k_q} \frac{rel_{q,i}}{\log_2 i} \tag{C.1}$$

$rel_{q,i}$ is the relevance level of the document in the *i*th position of the document list that was returned by the system for query *q*. The ideal discounted cumulative gain at rank k_q ($IDCG_q$), for a given query *q*, measures the DCG_q for an ideal, perfect ranking, and is defined as:

$$IDCG_q \overset{\text{def}}{=} ideal\_rel_{q,i} + \sum_{i=2}^{k_q} \frac{ideal\_rel_{q,i}}{\log_2 i} \tag{C.2}$$

where the document list is ordered per relevance level (higher to lower), and $ideal\_rel_{q,i}$ is the relevance level of the document in the i-th position. $IDCG_q$ allows us to normalize the values of DCG_q, facilitating the averaging across queries with different numbers of relevant documents. Thus, the normalized discounted cumulative gain at rank k_q ($NDCG_q$), for a given query q, is defined as:

$$NDCG_q \overset{\text{def}}{=} \frac{DCG_q}{IDCG_q} \tag{C.3}$$

Finally, $MNDCG$ is defined as the average of the $NDCG_q$s on all the queries.

C.2. Engine comparison

The comparison between DIESIRAE and Lucene that we achieved by means of the experiment with human raters was based on the following indexes:

$$\Delta MPr \overset{\text{def}}{=} MPr_{DIE} - MPr_{Luc} \tag{C.4}$$

$$\Delta MNDCG \overset{\text{def}}{=} MNDCG_{DIE} - MNDCG_{Luc} \tag{C.5}$$

$$\%\Delta MPr \overset{\text{def}}{=} \frac{\Delta MPr}{MPr_{Luc}} \cdot 100 \tag{C.6}$$

$$\%\Delta MRe \overset{\text{def}}{=} \frac{\Delta MRe}{MRe_{Luc}} \cdot 100 = \frac{\sum_{q=1}^{25} Nr_{q,DIE} - Nr_{q,Luc}}{\sum_{q=1}^{25} Nr_{q,Luc}} \cdot 100 \tag{C.7}$$

$$\%\Delta MFa \overset{\text{def}}{=} \frac{\Delta MFa}{MFa_{Luc}} \cdot 100 = \frac{\sum_{q=1}^{25} Nnr_{q,DIE} - Nnr_{q,Luc}}{\sum_{q=1}^{25} Nnr_{q,Luc}} \cdot 100 \tag{C.8}$$

where Nr_q and Nnr_q represent, respectively, the number of relevant and non-relevant documents at rank k, for a given query q. The same indexes were used for the experiment with the synthetic rater.

Table C.6 shows the evaluation indexes that were calculated using data collected during the experiment with human raters, for each query, and for both DIESIRAE and Lucene. Table C.7 shows the indexes generated during the experiment with the synthetic rater.

Appendix D. Significance: the *P*-value

We used the P-value $P(t, N)$ to calculate the statistic significance of difference variables $D \overset{\text{def}}{=} B - A$; defined as the following test statistic (see Croft et al., 2009):

$$t \overset{\text{def}}{=} \frac{\overline{D}}{S_D} \cdot \sqrt{N} \tag{D.1}$$

where $\overline{D}$ is the average, S_D is the standard deviation, and t is a Student's t-distribution with N degrees of freedom.

Using the one-tailed test, a P-value smaller that 0.01 is usually considered to indicate that the null hypothesis $H_0 : \mu_D = 0$ should be rejected, allowing us to conclude that the observed difference between $\overline{B}$ and $\overline{A}$ is significant.

D.1. Evaluating the ME-2L-HMM2

To verify the significance of ΔMAC_{d_w}, the increment in average accuracy for a given ambiguity degree, we defined the difference variable D, and calculated it for each of the $N = 10$ training rounds:

$$\begin{aligned} &\Delta MAC_{d_w} && \overset{\text{def}}{=} && MAC_{d_w} - MAC_{nc,d_w} \\ &\text{for significance, let } D && \overset{\text{def}}{=} && AC_{d_w} - AC_{nc,d_w} \end{aligned} \tag{D.2}$$

To verify the significance of ΔMAC, the increment in global accuracy, we defined the difference variable D, and calculated it for each of the $N = 10$ training rounds:

$$\begin{aligned} &\Delta MAC && \overset{\text{def}}{=} && MAC - MAC_{nc} \\ &\text{for significance, let } D && \overset{\text{def}}{=} && AC - AC_{nc} \end{aligned} \tag{D.3}$$

Table C.6
Performance indexes for the experiment with human raters.

	DIESIRAE				Lucene			
q	Pr_q	Nr_q	Nnr_q	$NDCG_q$	Pr_q	Nr_q	Nnr_q	$NDCG_q$
1	0.625	5	3	0.940	0.6	6	4	0.905
2	1	1	0	1	0.5	1	1	1
3	1	10	0	0.932	0.964	6	1	0.982
4	0.5	5	5	0.639	0	0	10	0.386
5	0.875	6	1	0.910	1	2	0	1
6	0	0	0	1	0	0	1	1
7	0.3	3	7	0.99	0.3	3	7	0.98
8	1	2	0	1	1	1	0	0.95
9	0.5	1	1	1	0	0	0	1
10	0	0	10	0.96	0	0	2	0.70
11	0.666	2	1	0.97	0.571	4	3	0.74
12	0.5	1	1	0.82	0.111	1	8	0.32
13	0	0	1	0.90	0.333	2	4	0.47
14	0.9	9	1	0.73	0.75	6	2	0.65
15	0.5	3	3	0.97	0.6	3	2	0.93
16	0.8	4	1	0.90	0.8	4	1	0.98
17	0.3	3	7	0.90	0.3	3	7	0.99
18	0.625	5	3	0.81	0.2	2	8	0.41
19	0.5	1	1	0.80	0.333	1	2	0.65
20	1	2	0	0.99	0.666	2	1	0.86
21	0.75	3	1	0.85	0.333	3	6	0.52
22	0.75	6	2	0.90	0.6	6	4	0.89
23	0.25	1	3	0.92	0.333	1	2	0.88
24	1	2	0	1	0	0	0	1
25	1	1	0	1	0	0	0	1

Table C.7
Performance indexes for the experiment with synthetic rater.

	DIESIRAE				Lucene			
q	Pr_q	Nr_q	Nnr_q	$NDCG_q$	Pr_q	Nr_q	Nnr_q	$NDCG_q$
1	1	8	0	0.84	1	10	0	0.79
2	1	1	0	1	0.5	1	1	1
3	1	10	0	1	0.86	6	1	0.98
4	0.7	7	3	0.61	0	0	10	0.39
5	1	7	0	1	1	2	0	1
6	0	0	0	1	0	0	1	0
7	1	10	0	0.79	1	10	0	0.79
8	1	2	0	1	1	1	0	1
9	1	2	0	1	0	0	0	1
10	1	10	0	0.79	0	0	2	0.70
11	1	3	0	0.95	0.71	5	2	0.75
12	0.5	1	1	0.79	0.11	1	8	0.32
13	1	1	0	0.97	0.5	3	3	0.55
14	0.9	9	1	0.79	0.75	6	2	0.63
15	1	6	0	1	0.8	4	1	0.93
16	1	5	0	1	1	5	0	1
17	1	10	0	0.79	1	10	0	0.79
18	0.86	7	1	0.74	0.30	3	7	0.47
19	0.50	1	1	0.77	0.33	1	2	0.63
20	1	2	0	0.93	0.67	2	1	0.82
21	0.75	3	1	0.74	0.33	3	6	0
22	1	8	0	0.83	1	10	0	0.79
23	1	4	0	0.94	0.67	2	1	0.81
24	1	2	0	1	0	0	0	1
25	1	1	0	1	0	0	0	1

D.2. Comparing DIESIRAE and Lucene

To verify the significance of ΔMPr, the increment in mean precision, we defined the difference variable D, and calculated it for each of the $N = 25$ queries:

$$\begin{aligned} \Delta MPr &\overset{\text{def}}{=} MPr_{Die} - MPr_{Luc} \\ \text{for significance, let } D &\overset{\text{def}}{=} Pr_{q,DIE} - Pr_{q,Luc} \end{aligned} \tag{D.4}$$

To verify the significance of $\Delta MNDCG$, the increment in total information gain, we defined the difference variable D, and calculated it for each of the $N = 25$ queries:

$$\begin{aligned} \Delta MNDCG &\overset{\text{def}}{=} MNDCG_{Die} - MNDCG_{Luc} \\ \text{for significance, let } D &\overset{\text{def}}{=} NDCG_{q,DIE} - NDCG_{q,Luc} \end{aligned} \tag{D.5}$$

Appendix E. Concordance among raters

Several statistic indexes have been proposed for the evaluation of agreement among raters. All of these indexes compute agreement as $(P(A) - P(E))/(1 - P(E))$, where $P(A)$ is the *observed* agreement among the raters, and $P(E)$ is the *expected* agreement (i.e., $P(E)$ represents the probability that the raters agree by chance). The values of the agreements are constrained to the $[-1, 1]$ interval, where a value of one means perfect agreement, zero means that the agreement is equal to chance, and a value of minus one means "perfect" disagreement (Di Eugenio and Glass, 2004).

The term $P(E)$ can be computed in two ways, according to whether the distribution of proportions over the categories is taken to be equal for the raters, or not. In other words, the first approach computes $P(E)$ using a single distribution that is built using data from all the raters. The second approach, instead, uses as many $P(E)$'s as the number of raters. Well-known multi-rater indexes belonging to the first approach are Fleiss' Kappa (Fleiss, 1971) and Siegel & Castellan's Kappa (Siegel and Castellan, 1998), while Randolph's K_{free} (Randolph, 2005) belongs to the second approach.

According to Brennan and Prediger (1981), the first approach is appropriate for *fixed-marginal* validity studies, while it is not well-suited for studies that have *free-marginal* distributions. In fixed-marginal validity studies, raters know a priori the quantity of cases that should be distributed into each category. For example, this might be the case in which a rater is free to assign cases to categories as long as there is a certain, predetermined amount of cases in each category in the end. In free-marginal validity studies raters do not know a priori the quantities of cases that should be distributed to each category. For example, this is the case in which a rater is free to assign cases to categories with no limits on how many cases must go into each category (Randolph, 2005).

According to the aforementioned description, our experiment – described in Section 11.3 – belongs to the family of free-marginal validity studies, and therefore, we used Randolph's K_{free}:

$$K_{\text{free}} \overset{def}{=} \frac{(1/(N \cdot n \cdot (n-1))) \cdot \left(\sum_{i=1}^{N} \sum_{j=1}^{k} n_{i,j}^2 - N \cdot n \right) - (1/k)}{1 - (1/k)} \tag{E.1}$$

where N is the number of cases to rate (in our experiment, a *case* means a given document appearing in the result list of a given query; if a document appears in, say, three query results, it represents three different cases); n is the number of collected ratings per case (5, the number of raters, in our experiment); k is the number of categories (6, in our experiment); and $n_{i,j}$ is the number of raters who assigned the i-th case to the j-th category (composing the so-called *agreement table*).

Appendix F. HMMs vs second-order HMMs

As described in Section 6, our model makes use of a second-order HMM. We initially chose a second-order model as a trade-off between the simplicity of first-order HMMs and the higher accuracy to be expected from higher-order HMMs. Our intuition was that, like higher-order n-grams, higher order HMMs would provided better results. To test this intuition, we compared the best ME-2L-HMM2 model we trained, with its first-order version (we tweaked the second-order Viterbi so that it behaved like the first-order version). We found the accuracy of the first-oder model to be $AC_1 = 0.83$ and $AC_{1,2:6} = 0.83$, while our second-order model obtained $AC = 0.83$ and $AC_{2:6} = 0.84$ (see Section 11.3).

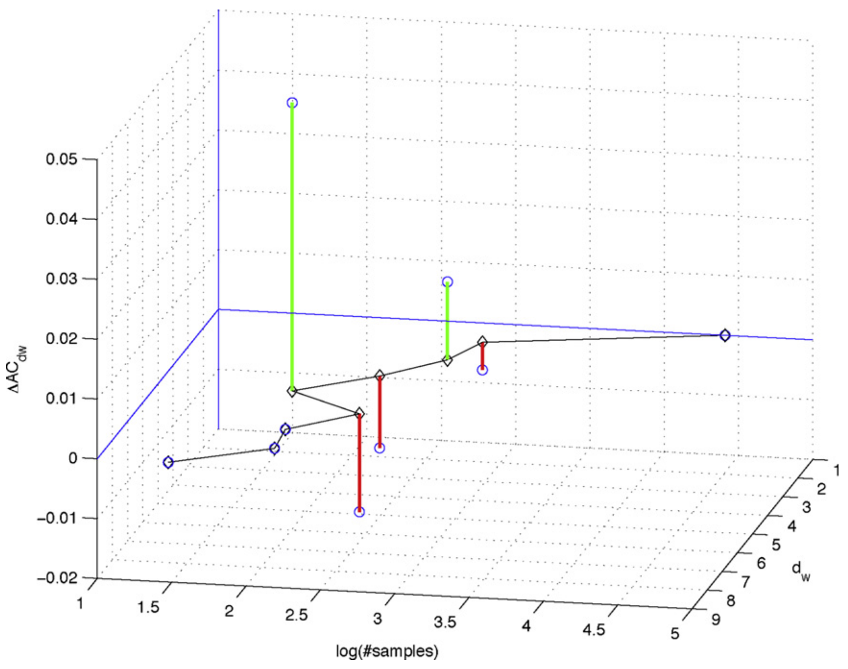


Fig. F.21. Advantage in AC_{d_w} of the second-order model on the first-order version.

The second-order version is actually only marginally better than the first-order one. To understand whether these disappointing results were due – as we argued – to an insufficient size of the training set (the well-known n-gram "sparsity problem", which gets worse as n increases; an analysis on this topic can be found in Allison et al., 2006), we analyzed how the accuracy was related to the degree of ambiguity (AC_{d_w}). We found that the two models performed similarly in the following two situations. Whenever the classification task was trivial or easy ($1 \le d_w \le 2$) and the number of samples was high (since words with small d_w were frequent) the two models performed equally well. Whenever the classification task was very hard ($7 \le d_w \le 9$) and the number of samples was low (since words with high d_w were rare) the two models performed equally poorly.

Fig. F.21 shows how the number of samples correlates to the degree of ambiguity, and how both influence the advantage of having a second-order model instead of a first-order one; $\Delta AC_{d_w} \overset{\text{def}}{=} AC_{d_w} - AC_{1,d_w}$. For $3 \le d_w \le 6$, which represents the vast majority of training samples having $d_w > 1$, the second-order model was slightly better than the other, indicating that by increasing the training set in this area we could have increased the second-order model's edge over the first-order one.

To summarize, we found that the second-order model did not provide a clear advantage in our experiments, but that result was likely due to the size of the training set. Increasing the size of the set, we expect the superior accuracy of the second-order model to emerge.

References

Alani, H., Kim, S., Millard, D.E., Weal, M.J., Hall, W., Lewis, P.H., Shadbolt, N.R., 2003 January. Automatic ontology-based knowledge extraction from web documents. IEEE Intelligent Systems 18, 14–21, URL http://dl.acm.org/citation.cfm?id=642316.642321

Allison, B., Guthrie, D., Guthrie, L., 2006 September. Another look at the data sparsity problem. In: Ninth International Conference on Text, Speech and Dialogue, Brno, Czech Republic, pp. 327–334.

Anastasi, G., Bellini, E., Nitto, E.D., Ghezzi, C., Tanca, L., Zimeo, E. (Eds.), 2012. Methodologies and Technologies for Networked Enterprises. No. 7200 in Lecture Notes in Computer Science. Springer.

Atserias, J., Casas, B., Comelles, E., González, M., Padró, L., Padró, M., 2006. Freeling 1.3: syntactic and semantic services in an open-source NLP library. In: Proceedings of the fifth international conference on Language Resources and Evaluation (LREC 2006), ELRA, Genoa, Italy, pp. 2281–2286.

Aufaure, M.-A., Soussi, R., Zghal, H.B., 2007. Siro: on-line semantic information retrieval using ontologies. In: ICDIM, IEEE, pp. 321–326.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., 2003. The Description Logic Handbook: Theory Implementation and Applications. Cambridge University Press, Cambridge, England.

Bhogal, J., Macfarlane, A., Smith, P., 2007. A review of ontology based query expansion. Information Processing and Management 43 (July), 866–886.

Bratus, S., Rumshisky, A., Magar, R., Thompson, P., 2009. Using domain knowledge for ontology-guided entity extraction from noisy, unstructured text data. In: Proceedings of the Third Workshop on Analytics for Noisy Unstructured Text Data, AND'09, ACM, New York, NY, pp. 101–106.

Brennan, R.L., Prediger, D.J., 1981. Coefficient kappa: some uses, misuses, and alternatives. Educational and Psychological Measurement 41, 687–699.

Charniak, E., Johnson, M., 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In: Proceedings of the 43rd Annual Meeting of the ACM, New York, June, pp. 173–180.

Che-Yu, Y., Hua-Yi, L., 2010. An automated semantic annotation based-on wordnet ontology. In: Proceedings of the Sixth International Conference on Networked Computing and Advanced Information Management NCM 2010, pp. 682–687.

Croft, B., Metzler, D., Strohman, T., 2009. Search Engines: Information Retrieval in Practice. Addison-Wesley, Boston, MA, pp. 301–342 (Ch. 8).

Darroch, J.N., Ratcliff, D., 1972. Generalized iterative scaling for log-linear models. The Annals of Mathematical Statistics 43 (5), 1470–1480.

de Marneffe, M.C., Manning, C.D., 2008 September. Stanford Typed Dependencies Manual. The Stanford Natural Language Processing Group, Stanford University, Stanford, CA.

Di Eugenio, B., Glass, M., 2004. The kappa statistic: a second look. Computational Linguistics 30 (March), 95–101.

Fellbaum, C. (Ed.), 1998. WordNet: An Elettronic Lexical Database. MIT Press, Cambridge, MA.

Ferrucci, D., et al., 2006. Towards an interoperability standard for text and multimodal analytics. Research Report RC24122 (W0611-188), IBM.

Finkel, J.R., Grenager, T., Manning, C., 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In: Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics, pp. 363–370.

Fleiss, J.L., 1971. Measuring nominal scale agreement among many raters. Psychological Bulletin 76 (5), 378–382.

Francis, W.N., Kucera, H., 1979. Brown Corpus Manual. Department of Linguistics, Brown University, Providence, RI.

Goodman, J., 2002. Sequential conditional generalized iterative scaling. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), Philadelphia, USA, July, pp. 9–16.

Gruber, T.R., 1993. A translation approach to portable ontology specifications. Knowledge Acquisition 5 (June), 199–220 (Special issue: Current Issues in Knowledge Modeling).

Guo, J., Xu, G., Cheng, X., Li, H., 2009. Named entity recognition in query. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'09, ACM, New York, NY, pp. 267–274.

He, Y., 1988. Extended Viterbi algorithm for second order hidden Markov Process. In: Proceedings of 9th International Conference on Pattern Recognition, Rome, Italy, January, pp. 718–720.

Jaynes, E.T., 1968. Prior probabilities. IEEE Transactions on Systems Science and Cybernetics sec-4 (3), 227–241.

Jelinek, F., 1969. A fast sequential decoding algorithm using a stack. IBM Journal or Research and Development 13, 675–685.

Jurafsky, D., Martin, J.H., 2000. Speech and Language Processing. Prentice Hall, Upper Saddle River, NJ, pp. 254–259 (Ch. 7).

Kashyap, V., Bussler, C., Moran, M., 2008. Ontology Authoring and Management. Data-Centric Systems and Applications. Springer, Germany (Ch. 6).

Klein, D., Manning, C., 2003. Fast Exact Inference with a Factored Model for Natural Language Parsing. Advances in Neural Information Processing Systems, vol. 15. MIT Press, Cambridge, MA.

Knublauch, H., Musen, M.A., Rector, A.L., 2004. Editing description logic ontologies with the protege owl plugin. In: Proceedings of International Workshop on Description Logics, Whistler, Canada.

Kontostathis, A., Pottenger, W.M., 2006. A framework for understanding latent semantic indexing (LSI) performance. Information Processing and Management 42 (January), 56–73.

Lafferty, J., McCallum, A., Pereira, F., 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In: Proceedings of 8th International Conference on Machine Learning (ICML-2001), Williamstown, MA, June, pp. 282–289.

Long Qiu, M.-Y.K., Chua, T.-S., 2004. A public reference implementation of the RAP anaphora resolution algorithm. In: Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004), vol. 1, Lisbon, Portugal, May, pp. 291–294.

Manning, C., Schütze, H., 1999 May. Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA, pp. 589–597 (Ch. 16).

Manning, C., Schütze, H., 1999 May. Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA, pp. 591–594 (Ch. 16).

McCallum, A., Freitag, D., Pereira, F., 2000 June. Maximum entropy markov models for information extraction and segmentation. In: Proceedings of 7th International Conference on Machine Learning (ICML-2000), Stanford, USA, pp. 591–598.

McCandless, M., Hatcher, E., Gospodnetic, O., 2010 July. Lucene in Action, 2nd ed. Manning Publications Co., Greenwich, CT.

Montedoro, M., Orsi, G., Sbattella, L., Tedesco, R., 2012. Ontology-based knowledge elicitation: an architecture. In: Anastasi, G., Bellini, E., Nitto, E.D., Ghezzi, C., Tanca, L., Zimeo, E. (Eds.), Methodologies and Technologies for Networked Enterprises, Vol. 7200 of Lecture Notes in Computer Science. Springer, Heidelberg, pp. 175–191 (Ch. 9).

Nadeau, D., Sekine, S., 2007. A survey of named entity recognition and classification. Lingvisticae Investigationes 30 (January (1)), 3–26.

Navigli, R., 2009. Word Sense Disambiguation: a survey. ACM Computing Surveys 41 (2), 1–69.

Rabiner, L.R., Juang, B.H., 1986. An introduction to Hidden Markov Models. IEEE ASSP Magazine 7 (1), 4–16.

Randolph, J.J., 2005. Free-marginal multirater kappa: an alternative to Fleiss' fixed-marginal multirater kappa. In: Proceedings of the Joensuu University Learning and Instruction Symposium, Joensuu, Finland, October 14–15.

Ratinov, L., Roth, D., 2009. Design challenges and misconceptions in named entity recognition. In: Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL'09), Association for Computational Linguistics, Stroudsburg, PA, pp. 147–155.

Ratnaparkhi, A., 1997. A simple introduction to maximum entropy models for natural language processing. Tech. rep., Institute for Research in Cognitive Science, University of Pennsylvania.

Ratnaparkhi, A., Roukos, S., Ward, R.T., 1994. A maximum entropy model for parsing. In: Proceedings of the International Conference on Spoken Language Processing, pp. 803–806.

Rinaldi, A.M., 2009. An ontology-driven approach for semantic information retrieval on the web. ACM Transactions on Internet Technology 9 (July), 10:1–10:24.

Della Pietra, S., Della Pietra, V.J.L., 1997. Inducing features of random fields. IEEE Transactions on Pattern Analysis and Machine Intelligence 19 (April (4)), 380–393.

Salton, G., Wong, A., Yang, C.S., 1975. A vector space model for automatic indexing. Communications of the ACM 18 (11), 613–620.

Sbattella, L., Tedesco, R., 2012. Knowledge extraction from natural language processing. In: Anastasi, G., Bellini, E., Nitto, E.D., Ghezzi, C., Tanca, L., Zimeo, E. (Eds.), Methodologies and Technologies for Networked Enterprises, Vol. 7200 of Lecture Notes in Computer Science. Springer, Heidelberg, pp. 193–219 (Ch. 10).

Schmid, H.,1994. Probabilistic part-of-speech tagging using decision trees. In: Proceedings of the International Conference on New Methods in Language Processing. Manchester, UK.

Shieber, S.M., 1984. The design of a computer language for linguistic information. In: Proceedings of Coling84, 10th International Conference on Computational Linguistics, Stanford University, Stanford, CA, July, pp. 362–366.

Shieber, S.M., 1986. An Introduction to Unification Based Approaches to Grammar, Stanford University Edition. Vol. 4 of CSLI Lecture Notes Series. Center for the Study of Language and Information.

Siegel, S., Castellan, N.J., 1998. Nonparametric Statistics for the Behavioral Sciences. McGraw Hill, Boston.

Smith, M.K., Welty, C., McGuiness, D.L. (Eds.), 2004. OWL Web Ontology Language Guide. W3C.

Toutanova, K., Klein, D., Manning, C., Singer, Y., 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of HLT-NAACL, vol. 1, Edmonton, Canada, pp. 173–180.

Wallach, H.M., 2004. Conditional random fields: an introduction. Technical Report MS-CIS-04-21, University of Pennsylvania, CIS, February.

Wimalasuriya, D.C., Dou, D., 2010. Ontology-based information extraction: an introduction and a survey of current approaches. Journal of Information Science 36 (June), 306–323.

Licia Sbattella received her PhD degree in computer science in 1990 and her MSc degree in psychology in 2002. She has been an associate professor at Politecnico di Milano since 1992. Her research interests are Natural Language Processing, human–computer interaction and human–human interaction, ICT for people with special needs, and knowledge and meta-knowledge representation. She is teaching as an invited professor at Università della Svizzera Italiana (Lugano, CH) and at Université Charles De Gaulle (Lille, France). She is a member of the G3ict Steering Committee and Chair of the G3ict Education Task Force (G3ict is a UN Flagship Advocacy for Accessibility).

Roberto Tedesco received the PhD degree in computer engineering in 2006 from Politecnico di Milano, from which he graduated in December 2001. He has been a visiting scholar at the L3S research center in Hannover, working on user modeling for e-learning platforms. He is now a contract researcher at Politecnico di Milano – MultiChancePoliTeam. His research interests include Natural Language Processing, assistive technologies, human–computer interaction, and e-learning.