Textual document pre-processing and feature extraction in OLEX

R. Curia¹, M. Ettorre¹, L. Gallucci¹,³, S. Iiritano¹,³ & P. Rullo¹,²

¹Exeura s.r.l.
²Dipartimento di Matematica
³DEIS–Dipartimento di Elettronica, Informatica e Sistemistica

Abstract

Knowledge Discovery in Text (KDT) has emerged as a challenging application due to the large amount of textual documents available from heterogeneous sources. OLEX is a KDT system for text classification developed at Exeura.

A critical step of a KDT process is the pre-processing phase, consisting of a number of complex tasks aimed at making documents “machine readable”. This paper describes the OLEX Pre-processing Module (OPM), an advanced software based on a general framework supporting the extraction from texts of linguistic, syntactic and structural relevant features. A main aspect of OPM is its capability to provide support for parallel text annotation.

1 Introduction

Managing the huge amount of textual documents available on the web and on the intranets has become an important problem of Knowledge Management. Thus, techniques and tools for text categorization are needed [1].

Textual document collections can be seen as sources of unstructured data for which knowledge mining can be made by using Knowledge Discovery in Text (KDT) [2], an interactive and iterative process based on four phases:

- Document Acquisition
- Document Pre-Processing
- Text Mining
- Result Interpretation
KDT is in general a very complex process, as all tasks needed to “prepare” documents (pre-processing) and to extract information (mining) are related to the difficult problems of flexibility and ambiguity proper of the natural language.

A particular Text Mining application is Text Categorization, used to assign documents to predefined categories on the basis of their content. OLEX [3, 4] is an ontology-driven categorization system, developed at Exeura (www.exeura.it), that supports the entire KDT process: document storage and organization, pre-processing and classification.

In this paper we describe the OLEX Pre-processing Module (OPM) for textual document annotation. OPM is an advanced software which supports both linguistic and structural analysis, whereby documents are annotated [5, 6] by meta-textual information (features), so obtaining a machine-readable representation of texts [7, 8] in terms of logic facts. A fundamental aspect of OPM is its capability to parallelize the feature annotation process by splitting a document into its sentences and then running one “annotation chain” for each of them. Such an approach allows us to drastically reduce the pre-processing time.

The paper is organized as follows. In Section 2 we provide an overview of OLEX. In Section 3 we describe the general framework for textual document pre-processing and its implementation in the OPM module. In Section 4 we discuss the implementation of OPM and show some experimental results.

2 OLEX: an overview

OLEX is based on a combined use of ontologies, linguistic processing techniques and logic programming. It supports manual as well as automatic generation of categorization rules.

Ontologies. In the context of classification, ontologies are used to provide the specific knowledge about the universe of discourse concerning a given corpus of
documents. Classifying a document w.r.t. an ontology means assigning it to one or more concepts (which thus represent our categories). Unlike the ML approach, the names of our categories bear “semantics” that can be exploited for the purpose of categorization. Clearly, as a borderline case, an ontology may consist of just a set of labelled concepts (that is the case of classical text categorization approaches). The ontology language of OLEX supports the specification of the following basic constructs: Concepts, Attributes, Properties (attribute values), Taxonomic (is-a) and Non-Taxonomic binary associations, Concept Instances, Links (association instances), Constraints (i.e., association cardinalities, exists link, for all links), Synonyms.

**Preprocessing.** Documents can be in several different formats (pdf, word, etc.). After their conversion into plain text, they are subjected to a morpho-syntactic analysis (that will be described in details in the next sections) and then transformed into sets of ground predicates of the form

\[
\text{word}(\text{DocId}, \text{TokenId}, \text{Token}, \text{Stem}, \text{Pos}, \text{SentId})
\]  

where: \(\text{TokenId}\) is the position of \(\text{Token}\) within document \(\text{DocId}\), \(\text{Stem}\) is the stem of \(\text{Token}\) computed by the Porter algorithm [9], \(\text{Pos}\) is the part-of-speech of \(\text{Token}\) and \(\text{SentId}\) is the sentence identifier where \(\text{Token}\) occurs. A token is either a noun, a proper noun, a verb or an adjective, according to the value of the \(\text{Pos}\) argument (thus, no stopwords occur in our document representation).

**Logic.** To exploit both the domain and the linguistic knowledge provided by the ontology and the document preprocessing, respectively, a powerful yet simple formalism for the specification of the categorization rules is needed. To this end, logic languages seem to fulfill all the requirements: high expressive power, simplicity and conciseness of knowledge representation and easy implementation of new reasoning tasks. Our approach relies on an extension of Datalog [10], called in this paper Datalog\(^{TC}\) (Datalog for Text Categorization), by suitable constructs, namely, *aggregate functions* [11] and *external predicates*. Datalog\(^{TC}\) is supported by the logic system DLV [12]. Each concept \(C\) of the given ontology is equipped with a set of logic rules, the *categorizer* of \(C\), denoted \(\text{CAT}_C\), which embeds the knowledge needed for the recognition of \(C\) within documents. \(\text{CAT}_C\) is a function \(\text{CAT}_C: D \rightarrow \{0, 1\}\) which, given a document \(d \in D\), returns a boolean “categorize \(d\) under \(C\)” or “don’t categorize \(d\) under \(C\)”. As we said before, OLEX supports techniques for the automatic generation of categorization rules, namely:

- automatic generation of logic rules from the background knowledge provided by the ontology; we call this set of rules the *default categorizer* of \(C\) (\(\text{DEF}_C\));
- automatic rule extraction from a set of training documents; we call this component the *inductive categorizer* of \(C\) (\(\text{IND}_C\)).

Clearly, the knowledge engineer can manually add any further piece of knowledge he considers useful for the purpose of a good categorization; we call this component the *manual categorizer* of \(C\) (\(\text{MAN}_C\)). Thus, the categorizer of \(C\) is defined as \(\text{CAT}_C = \text{DEF}_C \cup \text{IND}_C \cup \text{MAN}_C\). It is worth noting how OLEX combines...

The next example gives a hint about the use of Datalog\textsuperscript{TC} rules for categorization.

**Example.** Suppose we want to find out the concept “text mining” within a document; to this end, we may write rules of the following type.

- **Matching the term “text mining”:**
  
  \begin{align*}
  p(D, I) & \leftarrow \text{word}(D, I, \text{text}, \_\_ \text{noun}, S), \\
  \text{word}(D, J, \text{mining}, \_\_ \text{verb}, S), & \quad J = I + 1
  \end{align*}

  Here, \text{word} is a predicate unifying with facts of type (1). In the above rule, \(p(D, I)\) is true if the term “text mining” occurs at position \(I\) as a sequence of two consecutive words within the same section \(S\) of document \(D\).

- **Matching expressions.** The following rule is used to match an expression of the form “... discover(ing) knowledge within text(s)...”.
  
  \begin{align*}
  p(D, I) & \leftarrow \text{word}(D, I, \_\_ \text{X}, \text{verb}, S), \text{same}\_\text{stem}(X, \text{discover}), \\
  \text{word}(D, \_\_ \text{knowledge}, \_\_ \_\_ \text{S}), & \text{word}(D, \_\_ \_\_ \text{Y}, \text{noun}, S), \\
  \text{same}\_\text{stem}(Y, \text{text})
  \end{align*}

  where \text{same}\_\text{stem}() is defined in terms of the external predicate \#stem(), which computes the stem of a word by the Porter algorithm [9]:

  \begin{align*}
  \text{same}\_\text{stem}(X, Y) & \leftarrow \#\text{stem}(X, Z), \#\text{stem}(Y, Z)
  \end{align*}

  The following rule

  \begin{align*}
  p(D, I) & \leftarrow \text{word}(D, I, \_\_ \text{X}, \text{noun}, S), \text{same}\_\text{stem}(X, Y), \\
  \text{synonym}(Y, \text{text}), & \text{word}(D, J, \_\_ \_\_ \text{Z}, \text{verb}, S), \\
  \text{same}\_\text{stem}(Z, W), & \text{synonym}(W, \text{mining}), J = I + 1
  \end{align*}

  is used to match couples of words that are synonyms of “text mining” (note that \text{synonym}() is an ontology predicate).

- **Looking for instances.** Next is a rule aimed at capturing the concept “text mining” within a paragraph talking about the application of mining techniques (such as clustering, classification, etc.) to texts:

  \begin{align*}
  p(D, I) & \leftarrow \text{word}(D, I, \_\_ \_\_ \text{S}), \text{same}\_\text{stem}(X, \text{text}), \\
  \text{word}(D, \_\_ \_\_ \text{Y}, \_\_ \_\_ \text{S}), & \text{instance}\_\text{of}(\text{text}\_\text{mining}, Y)
  \end{align*}

  Here \text{instance}\_\text{of}(\text{text}\_\text{mining}, Y) is an ontology predicate.

- **Classifying.** Finally, we write down the following classification rule

  \begin{align*}
  \text{classify}(D, "\text{text mining}") & \leftarrow \#\text{count}\{I : p(D, I)\} > n.
  \end{align*}

  expressing that document \(D\) is classified under concept “text mining” provided that at least \(n\) occurrences of the previous rules succeed. To “counts” such occurrences we use the aggregate predicate \#\text{count}().
3 OPM: the OLEX pre-processor module

OPM implements all the tasks of the general framework for Feature Annotation from Textual Documents (FATD) shown in Figure 2. As we can see, this framework is based on five phases: Normalization, Pre-analysis, Linguistic pre-processing, Syntactic pre-processing and Conceptual pre-processing.

The annotation process of a document consists in the sequential execution of (a subset of) the above phases. Each phase generates some specific kind of annotation, which we represent by logical facts. The user is given the possibility of cutting off some step of the whole process. Obviously, “shortest” processing chains mean faster processes, but poorer output annotated documents.

3.1 Normalization

Textual documents are heterogeneous w.r.t. their type (doc, pdf, PostScript, ecc.). The Document Normalization phase reduces all texts to a unified format based on XHTML, an XML format that preserves the initial text formatting. In the context
of OLEX, the normalization is performed by a number of Conversion Filters, each being a Java Class that converts a document from a particular input format to a particular output format. The Normalization Manager creates a Filter Chain to obtain a specific output format starting from a given input format. For an instance, consider the problem of transforming a Word document into XHTML, and suppose there are three filters that can be used: F1:DOC-to-PDF, F2:PDF-to-HTML, and F3:HTML-to-XHTML. Then, the Manager builds a Filter Chain applying F1,F2 and F3 in the specified order. When for a specific conversion task there are more possible Filter Chains, the Manager instantiates the one with the smallest length.

3.2 Pre-analysis

The Pre-analysis consists of two main activities: Tokenization and Structural Analysis. It is both language- and domain-independent.

**Tokenization.** The text is divided into tokens (word, symbols, XML tags, numbers, etc.). A token is the smallest piece of information contained in a document. Within OPM, the tokenizer is a finite-state transducer implemented as a variant of the Mealy ASF that segments the input stream in a sequence of tokens.

**Structural Analysis.** XHTML documents produced as the output of the Normalization Phase do have a regular structure, due to the tags that define the role and the formatting of the different parts of the text. In OPM, the Structural Analysis identifies in the text, through the HTML tags, all document sections (chapter, paragraph, etc.). Further, it uses a “dissection” heuristics, based on regular expressions, to identify possible sections that are not explicitly defined. The OPM Structural Analyzer is implemented as an XML Parser that parses the XHTML using the XHTML DTD defined by the W3C (www.w3c.org).

3.3 Morpho-syntactical pre-processing

This phase is language-dependent and, thus, requires some lexical resources concerning the language of the document. As a result, it produces, for each token, the morphological features (stem, lemma and Part of Speech - PoS), which play a very important role in a KDT process, whether using clustering techniques based on the Vector Space Model [7], or classification techniques based on the evaluation of rules [4]. The following tasks are performed by OPM:

**Stemming.** This step is based on a Finite State Machine (FSM) for regular expression recognition used to remove all prefixes and suffixes for each token. The OPM Stemming algorithm is based on the suffix stripping method [9]. The FSM supports 12 different languages, but it can be easily extended to other languages.

**Lemmatization.** For each token the lemma is obtained, thus reducing the syntactical variants of the words to a unique form (by eliminating plurals, tenses, gerund forms, etc.)

**Part-of-Speech Tagging.** The PoS-Tagging is the task of assigning to each token in a sentence a tag indicating its lexical syntactic category, such as noun or verb. The main problem to be faced by a PoS-Tagger is disambiguation. All used tech-
techniques resolve this problem using two restrictions: (1) each token has a limited set of possible tags; (2) when there is more than one tag for a token, the correct tag is assigned using lexical rules based on the context of the token. OPM relies on the Hepple PoS-Tagger [15], a variant of the Brill tagger [16], which produces a part-of-speech tag as an annotation on each word or symbol. For the English language, the tagger uses a default lexicon and a set of transformation rules (both of contextual and lexical type) obtained by a training on a large corpus taken from the Wall Street Journal. For the Italian language, the lexicon and the rule set are the result of a training on a large corpus taken from the 10 most important Italian newspaper and from 10,000 textual documents of the Italian Public Administration.

Sentence splitting. A sentence is a piece of text having its own semantic unity. For an instance, two phrases separated by a semicolon but linked by “thus”, form a sentence. For the time being, OPM is limited to considering only simple sentences, such as sequences of tokens terminating by a period.

Next we show an example of the logical facts that are generated by the Morpho-Syntactical Pre-Processing.

Example. Let us consider a textual document about databases, with 247 different tokens, and suppose that the third sentence of this document contains the following fragment of text: “... A database is a structured collection of data. It.... ”. The representation of this paragraph is of the following type:

```plaintext
... word(57,'a','a','at',3).
word(58,'database','databas','nn',3).
... word(61,'structured','structur','vbn',3).
... sentence(3,57,148).
bold(58).
...
```

Here, the fact `word(58,'database','databas','nn',3)` has the following meaning: “database” occurs in the document as the 58th word, in the 3rd sentence, its stem is “databas” and its part-of-speech is “nn” (noun). In turn, the fact `sentence(3,57,148)` expresses that the 3rd sentence goes from the 57th to the 148th word.

3.4 Quantitative analysis

In OPM, this step essentially consists in the determination of n-gram Frequencies. An n-gram is a sequence of n features annotated by the Morpho-Syntactical Pre-Processing Phase (token, stem, lemma). For a given document, this step evaluates the total number of n-grams w.r.t. each feature, and the absolute frequency of each n-gram. The OPM Frequency Analyzer is based on a Java Map to store all n-gram for each type of annotated feature. In the OPM configuration step, the value m
of the maximum \( n \)-gram length is chosen, and the system performs the frequency analysis for all \( n \)-grams of length in the range \([1, m]\).

**Example (Cont’d)** Let us consider again the fragment of text of the previous example. The facts generated by the Quantitative Analyzer are of the following kind:

\[
\begin{align*}
n\text{Gram}(1, 'token', 'database', 13). \\
n\text{NumberOfGrams}(1, 'token', 247). \\
n\text{Gram}(1, 'stem', 'databas', 16). \\
n\text{NumberOfStems}(1, 'stem', 218).
\end{align*}
\]

Here, a fact like \( n\text{Gram}(1, 'token', 'database', 13) \) says that “database” is a \( n \)-gram of length 1, it is of type “token”, and its frequency within the document is 13. Likewise, \( n\text{NumberOfStems}(1, 'stem', 218) \) expresses that within the analyzed documents there are 218 different \( n \)-grams of type “stem”.

### 3.5 Reduction

The text is submitted to both explicit and implicit stop-feature annotation. Explicit stop-features occur frequently in a textual document and have a very low informative content (e.g., articles, prepositions, conjunctions, adverbs, common verbs and adjectives, recognized through the PoS-Tagging) [17], while implicit stop-features are tokens which occur uniformly in the corpus (i.e. they have similar frequencies in all documents). The OPM Reducer performs the annotation of both implicit and explicit stop-tokens. Concerning the former, the reducer needs to know the results of the Frequency Analysis; as for the latter is concerned, a stopword list, that depends on the language of the document, together with the PoS-Tagging information, are needed.

### 3.6 Terminological pre-processing

The aim of this phase is to find all relevant terms in the documents. A term is a sequence of token that expresses a unique sense. There are two steps:

**Term analysis.** This task uses specialized lexicons to annotate multi-token sequences that represent a unique concept. These lexicons can be referred to specific domains (biology, sport, etc.) and can also contain common terms (names of locations, names of the months, and so on). To this end, the OPM Term Analyzer uses a set of dictionaries (gazetteers); each dictionary is a plain text file with one entry per line. Each file represents a set of names, such as names of cities, organizations, etc. For each list, the annotation type is specified (location, currency, organization, etc.). All lists are compiled into a finite state machine, that is used to annotate all tokens in the text. **Syntactic parsing.** This task defines the syntactic structure of the sentences using a Context Free Grammar for the language of the document. For each sentence a syntagmatic tree is obtained. Using a regular expression engine and the specialized grammars, it is possible to recognize dates,
numbers, phraseological forms. The OPM Syntactical Parser is a Regular Expression engine that uses a Context Free Grammar to annotate sequences of terms. Grammar rules specify the types to be identified in particular circumstances.

**Example.** The following are examples of facts generated by the term analyzer:

\[
\text{term}(3, 21, \text{company}, \text{reuters}, 77).
\text{term}(3, 3, \text{city}, \text{hong kong}, 14).
\text{term}(3, 10, \text{day}, \text{wednesday}, 60). 
\]

3.7 Conceptual analysis

This phase requires semantic computational lexicons that represent the semantic of terms in a machine-manageable way.

**Semantic Interpretation.** Using semantic compositional rules, a significant representation of the best parse for each sentence is obtained [18]. This representation is added to a domain model based on an ontology.

**Co-reference Analysis.** Given a new concept of the domain ontology added by the Semantic Interpretation, this task tries to find instances of this concept in the document [19].

4 OPM implementation and performances

The annotation process is a very complex and heavy task from the computational point of view. Therefore, we have devised a multithreading architecture whereby more Annotation Chains (AC) can be executed in parallel (see Figure 3), an AC being a pipeline of annotation modules. When a document \(d\) is to be annotated, the OPM Manager (a) tokenizes \(d\), (b) generates a partition \(\{d_1, ..., d_n\}\) of \(d\), (c) builds a set of \(n\) AC’s, and (d) assigns each \(d_i\), \(1 \leq i \leq n\), to one AC.

We have run a number of preliminary experiments in order to check the performances of OPM. To this end, we have processed a set of English XHTML documents of 1 Mb average size on a Windows 2000 Server with one Pentium IV 1GHz and 512 Mb RAM.
The first group of experiments was aimed at determining the optimal number of Annotation Chains to be assigned to the (unique) processor, that was 2. Then, we have run the whole annotation process for all the XHTML documents, obtaining an average processing time of 0.98 seconds per document. By excluding the Term Analysis module from the Annotation Chain, the average processing time was 0.80 sec.

References


