Combining logic programming and domain ontologies for text classification

Chiara Cumbo[2], Salvatore Iiritano[1], and Pasquale Rullo[2]

[1] Exeura s.r.l., iiritano@exeura.it
[2] Dipartimento di Matematica, Università della Calabria, 87030 Rende (CS), Italy,
{cumbo,rullo}@mat.unical.it

Abstract This paper describes a prototypical system supporting the entire classification process: document storage and organization, pre-processing, ontology construction and classification. Document classification relies on two basic ideas: first, using ontologies for the formal representation of the domain knowledge; second, using a logic language (an extension of Datalog by aggregate functions that we call Datalog^f) as the categorization rule language. Classifying a document w.r.t. an ontology means associating it with one or more concepts of the ontology. Using Datalog^f provides the system with a natural and powerful tool for capturing the semantics provided by the domain ontology and describing complex patterns that are to be satisfied by (pre-processed) documents. The combined use of ontologies and Datalog^f allows us to perform a high-precision document classification.

1 Introduction

Managing the huge amount of textual documents available on the web and the intranets has become an important problem of knowledge management. For this reason, modern Knowledge Management Systems need for effective mechanisms to classify information and knowledge embedded in textual documents [1,2].
A number of classification approaches have been so far proposed, such as those based on machine learning [3] and those based on clustering techniques using the vector space model [4,5,6].
In this paper we describe a prototypical classification system which relies on two basic ideas: first, using ontologies for the formal representation of the domain knowledge and, second, using a logic language as the categorization rule language.
An ontology is a formal representation of an application domain [7,8]. In the context of a classification process, an ontology is intended to provide the specific knowledge concerning the universe of discourse (categorization based on the domain context). Classifying a document w.r.t. a given ontology means associating it with one or more concepts of the ontology. To this end, each concept is equipped with a set of logic rules that describe features of a document that may relate to the given concept. The logic language we use in our system is an extension of Datalog [9] with aggregate functions [10]. Throughout this paper

we refer to this language as Datalog^f. The advantage of using Datalog^f as the categorization rule language is twofold: first, we can exploit its expressive power to capture the domain semantics provided by the ontology and describe complex patterns that are to be satisfied by documents; second, the encoding of such patterns is very concise, simple, and elegant. We notice that others rule-based techniques have been proposed by several authors, but they are mainly devoted to the resolution of linguistic problems, such as the disambiguation of terms for the reduction of the vector dimensions [11], or for the improvement of the results of the classification task [3].
The execution of Datalog^f programs is carried out by the DLV system [12], which is part of our categorization engine. DLV is a well-known reasoning system which supports a completely declarative style of programming based on a bottom-up evaluation of the stable model semantics of disjunctive logic programs.

The paper is organized as follows. In Section 2 we provide an overview of the system. In Section 3 we describe the ontology management. In Section 4 we discuss the pre-processing phase and in Section 5 we present our classification technique based on Datalog^f. Finally, we give our conclusions.

2 A system overview

The prototype is intended as a corporate classification system supporting the entire process life-cycle: document storage and organization, ontology construction, pre-processing and classification. It has been developed as a Web application based jsp-pages on the client side. A sketch of its architecture is shown in figure 1. In the following sections we shall focus our attention on ontology management, pre-processing and classification.

3 Ontology Management

Ontologies in our system provide the knowledge needed for a high-precision classification. The ontology specification language supports the following basic constructs: Concepts, Attributes, Properties (attribute values), Taxonomic (is-a and part-of) and Non-Taxonomic binary associations, Association cardinality constraints, Concept Instances, Links (association instances), Synonyms. The creation of an ontology is supported by the Ontology Editor which provides a powerful visual interface based on a graph representation.

Example 1. KIMOS is an ontology developed within Exeura (www.exeura.it) with the purpose of classifying all company's software resources and the respective documentation. A fragment of KIMOS is given in figure 2. Here, the central concept is "Software" which is related to the other concepts by both taxonomic and non-taxonomic relations. For an instance, the edge connecting "Software" with "Language" represents the (many-to-many) relation "developed-in", while the one between "Software" to "OS Compatible" represents the relation "runs-on"; the concept "Software" is subdivided into a number of sub-concepts that

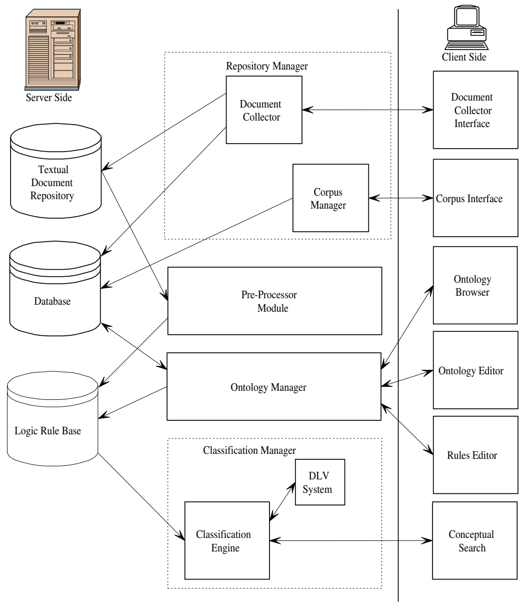

Figure 1. The System Architecture

group the different instances of "Software" into the appropriate categories. In figure we have reported only the concept "DB" that represents the class of softwares for databases. This concept is related to "Software" by an is-a relation and it is classified into "DBMS" and "DB Tool". In turn, "DBMS" is classified as either "Relational DBMS" or "Others" (i.e., DBMS of different types). An instance of "Relational DBMS" is "MySQL" which is related to "Unix-C" (an instance of "OS Compatible") by the link "runs-on". □

Once created, the user can navigate the ontology using the Ontology Browser which offers the following basic facilities:

– for a given concept, the user can easily explore its sub-concepts or, viceversa, collapse the underlying hierarchies
– the user can select the relationships whereby moving away from a given concept
– the user can filters the (possibly large) list of instances of a given concept.

Internally, an ontology is stored as a set of facts. As we will see in section 5, these facts represent an input to categorization programs.

Example 2. The internal representation of KIMOS consists of facts representing:

– concepts, e.g., *concept(DB)*;

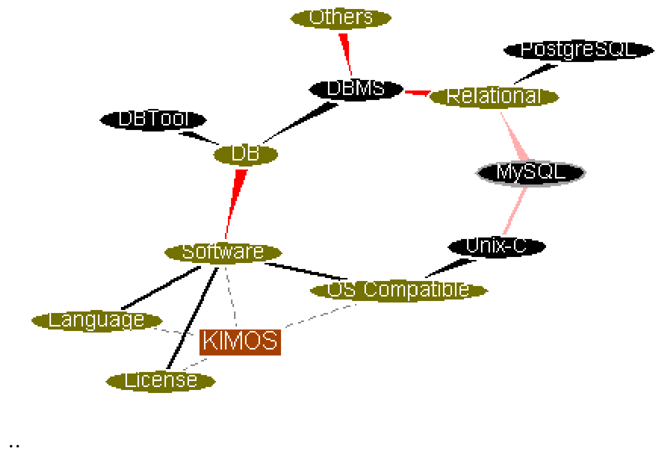

Figure 2. The KIMOS Ontology

- attributes, each identified by an Id, a Data-Type and the Id of the concept which belongs to; for instance, *attribute(size-MB,real,Software)* represents the attribute "size-MB", of type real, of the concept "Software";
- Properties (i.e., attribute instances) each characterized by an attribute name and a value; for instance, *property(size-MB,1.35)*;
- Taxonomic relationships of the form *is-a(DB,Software)*;
- Non-taxonomic relationships such as *association(runs-on,Software,OS Compatible)* which represents the relation "runs-on" between "Software" and "OS Compatible"; we represent also the inverse *inverse-of(runs-on, supports)*;
- association cardinality constraints, e.g., *cardinality(runs-on, "$\geqslant$ 1")* and *cardinality(supports, "$\geqslant$ 1")*;
- Link associations (i.e., binary association between instances), e.g., *link(runs-on, MySql ,Unix-C)*;
- Concept Instances such as *instance-of(Relational DBMS,MySql)*;
- Synonyms such as *synonym(Database,DB).*

□

4 Pre-processing

The aim of the Pre-Processing step is to obtain a machine-readable representation of textual documents [13]. This is done by *annotating* documents with meta-textual information obtained by a linguistic and structural analysis.
The Pre-Processor module supports the following tasks:

- Pre-Analysis, based on three main activities: Document Normalization, Structural Analysis and Tokenization.

– Linguistic Analysis, based on the following steps:
  - Lexical Analysis: for each token, the morpho-syntactical features are obtained (the stem and the Part of Speech - PoS). The PoS Tagging step is based on a variant of the Brill Tagger (a rule-based Pos-Tagger [14]).
  - Quantitative Analysis which provides, for a given document, information about the number of different tokens and stems as well as the absolute frequency for each token and stem.

The output of the Pre-Processing phase is a set of facts representing the relevant information about the processed document. As we shall see in section 5, these facts represent an input to our categorization programs.

Example 3. Consider, for example, a textual document about databases, with 247 different tokens, and suppose that the third paragraph of this document contains the following fragment of text: "... A **database** is a structured.... ". The representation of this paragraph is like this:

```
...
word(57,'a','a','at').
word(58,'database','databas','nn').
word(59,'is','is','bez').
word(60,'a','a','at').
word(61,'structured','structur','vbn').
bold(58).
par(3,57,148).
tokenFrequency('database',13).
stemFrequency('databas',16).
numberOfTokens(247).
numberOfStems(218).
```
□

5 Document Classification

The basic idea is that of using logic programs to recognize concepts within texts. Logic rules, indeed, provide a natural and powerful way to describe features of document contents that may relate to concepts. To this end, we use the logic language Datalog^f [10], an extension of Datalog by aggregate functions. The module of our system which supports the classification process is the Classification Engine which relies on DLV system [12] for the bottom-up evaluation of Datalog^f programs.

5.1 The Datalog^f language

We call Datalog^f the logic language obtained by extending Datalog [9] by aggregate functions. A *function* has the form $f(Vars : Conj)$ where f is the name (count, sum, min, max, sum) and $Vars$ a set of variables occurring in the conjunction $Conj$. Intuitively the expression $Vars : Conj$ represents the set of values assumed by the variables in $Vars$ making $Conj$ true. An *aggregate atom* is an expression of the type $Lg \leq f(Vars : Conj) \leq Ug$ where Lg and Ug are positive integer constants or variables called *guards*. For instance, $count\{V : a(V)\} < value$ is an aggregate atom whose informal meaning is: the number of ground instances of $a(V)$ must be less than $value$. A *Datalog^f program* is a logic program in which aggregate literals can occur in the body of rules. Rules with aggregate atoms are required to be $safe$ [10]. It is worth noticing that the result of an aggregate function can be saved by an assignment. For instance in the following rule $h(X) : -X = \#count\{V : a(V)\}$, all the ground instances of $a(V)$ are counted up and the value of count is assigned to X.

5.2 Categorization programs

By combining the expressive power of Datalog with that of aggregate functions, Datalog^f provides a natural and powerful tool for describing categorization rules within our system. A categorization program relies on a number of predefined predicates, that are of two types:

1. *Pre-processing predicates* representing information generated by the pre-processing phase; examples of such predicates are:
   $word(Id, Token, Stem, PoS)$
   $title(Id, Token, Stem, PoS)$
   where *Id* represents the position of *Token* within the text, *Stem* is the stem of the token and *PoS* its Part-of-Speech, and
   $tokenFrequency(Token, Number)$
   which represents the number of times Token occurs in the text.
2. *Ontology predicates* representing the domain ontology; examples of this kind of predicates are the following: $instance\_of(I, C)$ (I is instance of the concept C), $synonym(C1, C2)$, $isa(C1, C2)$, $part\_of(C1, C2)$, $association(A, C1, C2)$, etc..

In addition, we use the predicate $relevant(D, C)$ to state that document D is relevant for concept C.
Now, we equip each concept C of a given ontology with a set of Datalog^f rules, the *categorization program* P_C of C, used to recognize C within a given document D. The set of facts of P_C consists of the facts representing the domain ontology (see Section 3) as well as those representing the pre-processed document (see Section 4). The rules of P_C represent conditions that are to be satisfied in order D be considered relevant for C.

Example 4. We next provide an incremental construction of a categorization program associated with the concept "DB" of the KIMOS ontology (see example 1).

Rules looking for keyword. We start with the following simple rules looking for the keyword "DB":

r_0: $t0 : -title(\_, "DB", \_, \_)$.

r_1: $t1 : -tokenFrequency("DB", F), F > a$.

In rule r_0 above, the predicate $t0$ is true if "DB" occurs in the title, while t_1 in r_1 is true if the frequency F of the token "DB" is greater than a given constant a.
We can now refine our keyword search by exploiting synonyms; for instance, we can restate r_0 as

r_0: $t0 : -title(\_, X, \_, \_), synonym(X, "DB")$.

and replace r_1 by the following two rules:

r_2: $t2(X, F) : -synonym(X, "DB"), word(\_, X, \_, \_),\ tokenFrequency(X, F)$.

r_3: $t3$:- $F1 = \#sum\{F, X : t1(F, X)\}, F1 > a$.

Rule r_2 above "evaluates", for the concept "DB" and each of its synonyms, the respective frequency F; rule r_3, in turn, determines the total number $F1$ of times the concept "DB" and each of its synonyms appears in the text (this is performed by the aggregate function *sum*).

Rules looking for terms. Using the next rules we look for the term "structured data" within the document:

r_4 : $t4(I)$:- $word(I, "structured", \_, \_), word(J, "data", \_, \_), J = I + 1$.

r_5 : $t5(F)$:- $F = \#count\{I : t5(I)\}$.

We may relax the above condition, requiring the words "structured" and "data" to be found, in the specified order, within a distance of at most 5 words inside the same paragraph:

r_6 : $t6(I)$:- $word(I, "structured", \_, \_), word(J, "data", \_, \_), J > I,$
$L = J - I, L <= 5, sameParagraph(I, J)$.

r_7: $sameParagraph(I, J)$:- $par(Id, Init, Fin), I >= Init, J <= Fin$.

r_8 : $t8(F)$:- $F = \#count\{I : t7(I)\}$.

Rule r_8 above counts the number of times the searched term occurs in the same paragraph.

Rules matching expressions. Next we write rules to recognize, within a paragraph, an expression of the following type: a verb with stem "store", followed by a name having "tabl" or "relat" as its stem (i.e., we are trying to recognize sentences such as "data are stored within tables...").

r_9 : $t9(I)$:- $word(I, \_, \text{"}store\text{"}, \text{"}vb\text{"}), word(J, \_, \text{"}tabl\text{"}, \_), sameParagraph(I, J).$

r_{10} : $t10(I)$:- $word(I, \_, \text{"}store\text{"}, \text{"}vb\text{"}), word(J, \_, \text{"}relat\text{"}, \_), sameParagraph(I, J).$

r_{11} : $t11(F)$:- $F = \#count\{I : t9(I)\}.$

Rules exploiting the ontology knowledge. We can improve the precision of the classification process by using the underlying domain ontology. For instance, if a document talks about some specific instances of the concept "db", such as Oracle, Access, etc. (note that an instance of "relational DBMS", which is a sub-concept of "db", is also an instance of "DB"), it is quite obvious considering the document as pertinent to the concept "db". So, we write the following rules:

r_{12} : $t12(I, F)$:- $instance\_of(\text{"}DB\text{"}, I), tokenFrequency(I, F).$

r_{13}: $t13(N)$:- $N = \#count\{I : t11(I, \_)\}.$

r_{14} : $t14(F)$:- $F = \#sum\{F1, I : t11(I, F1)\}.$

r_{15} : $t15(T)$:- $T = \#count\{I : instance\_of(I, \text{"}DB\text{"})\}.$

where: r_{12} provides the number of occurrences of each instance of "db" in the document; r_{13} counts the number of distinct instances of "db"; r_{14} provides the total number of instances (duplicated included) of "db" and r_{15} gives the number of instances of "db" in the ontology. Finally, the rule

r_{16} : $t16(K, L)$:- $t13(N), t14(F), t15(T), K = N/T, L = F/N.$

expresses a measure, in terms of K (the fraction of the instances of "db" that are cited within the document) and L (which takes into account the fact that each instance might be cited several times), of the presence into the document of words representing instances of the concept "db". $\square$

As we have mentioned before, we use DLV as the categorization engine in our system. DLV is a very powerful system for the bottom-up evaluation of disjunctive logic programs extended by a number of constructs (Datalog^f is a subset of the DLV language). It is used in many real applications where efficiency is a strong constraint.
The evaluation strategy of categorization programs is based on the following two observations:

- there are documents that are straightforward to classify, i.e., for which simple keyword-based rules (like $r_1 - r_2$ above) are enough; suppose, for instance,

that the word "db" is contained in the title or it occurs frequently throughout the text; in such cases we can confidently classify the document at hand as relevant for the given concept only by using few simple rules (like r_1 and r_2) and forgetting of the remaining ones occurring in the rest of the categorization program;
– a deeper semantic analysis is needed only in case of documents that are difficult to classify because concepts do not appear explicitly; to this end, the execution of more complex rules (for instance, rules trying to match complex expressions) is required.

Now, the implementation of the above evaluation strategy proceeds, roughly speaking, as follows: we structure the categorization program P_C, associated to the concept C, into a number of components, say, $c_1, ..., c_n$. Each component groups rules performing some specific retrieval task, such as word-based search, term matching, etc., of increasing semantic complexity – that is, each component is capable to recognize texts that are possibly inaccessible to the "previous" ones. Given a document D, the evaluation of P_C (w.r.t. D) starts from c_1 (the "lowest" component) and, as soon as a component c_i, $1 \leq i \leq n$, is "satisfied" (by D), the process stops successfully – i.e., D is recognized to be relevant for C and the fact $relevant(D, C)$ is stated to be true; if no such a component is found, the classification task fails.

5.3 Ontology-driven Classification Strategy

Let D be a document that has to be classified w.r.t. an ontology O. As we have seen in the previous subsection, each concept C of O is equipped with a suitable categorization program P_C whose evaluation determines whether D is relevant for C or not. An exhaustive approach would require to "prove" D w.r.t. the categorization program of each concept of O, and this could result in a rather heavy computation. However, we can drastically reduce the "search space" if we adopt an ontology-driven classification technique which exploits the presence of taxonomic hierarchies. This technique is based on the principle that if a document is relevant for a concept then it is so for all of its ancestors within an is-a taxonomy (unless the contrary is explicitly stated). This principle is expressed by the following recursive rule:

$relevant(D, X) : -relevant(D, Y), isa(Y, X)$

As an example, if a document is relevant for the concept "Relational DBMS" of the KIMOS ontology, then it is so for the concepts "DBMS", "DB" and "Software". If we want to exclude the latter, we simply write:

$relevant(D, X) : -relevant(D, Y), isa(Y, X), X <> "Software".$

The above inheritance principle suggests us a classification strategy where concepts within a sub-class hierarchy are processed in a bottom-up fashion. As soon as D is found to be relevant for a concept C in the hierarchy H, it is not any more processed w.r.t. any of the ancestors of C in H. The relevance association of D to the ancestors of C is automatically performed by the above recursive rule.

6 Conclusion

We have presented a prototypical text classification system which relies on a combined use of ontologies and logic programming. The former are used to represent the domain knowledge, the latter to recognize concepts within texts. To this end, each concept of an ontology is equipped with a categorization program, i.e., a logic program written in Datalog^f – an extension of Datalog by aggregate functions. A categorization program is designed to discover complex patterns within texts using the knowledge provided by the underlying ontology. The classification process is ontology-driven and, as a result, provides a relationship between concepts and documents. The categorization engine is based on the logic programming system DLV.
So far, we have carried out a number of preliminary tests which seem to be very promising in terms of efficiency even on large documents. For an instance, we can classify a document of over 70000 words w.r.t. a Kimos Ontology (7 concepts) in 0.51 seconds. Further experimentation is currently being performed.
Current work is concerned with the extension of Datalog^f with external functions for the efficient execution of tasks such as stemming, substring matching, etc.

References

1. Ciravegna: (LP)2, an Adaptive Algorithm for Information Extraction from Web-related Texts. In: Proc. IJCAI-2001 Work. on Adaptive Text Extraction and Mining. (2001)
2. Riloff: A Case Study in Using Linguistic Phrases for Text Categorization on the WWW. In: AAAI/ICML Work.Learning for Text Categorization. (2001)
3. Cohen: Text categorization and relational learning. In: Proc. of ICML-95, 12th Int. Conference on Machine Learning. (1995)
4. Díaz, A., Buenaga, M., Urena, L., García-Vega, M.: Integrating linguistic resources in an uniform way for text classification tasks. In: Proc. of LREC-98, 1st Int. Conference on Language Resources and Evaluation. (1998) 1197–1204
5. Hsu: Classification algorithms for NETNEWS articles. In: Proc. of CIKM-99, 8th ACM Int. Conference on Information and Knowledge Management. (1999) 114–121
6. Brank, J., Grobelnik, M., Milic-Frayling, N., Mladenic, D.: Feature selection using support vector machines. In: Proc. of the 3rd International Conference on Data Mining Methods and Databases for Engineering, Finance, and Other Fields. (2002)
7. Decker, S., Erdmann, M., Fensel, D., Studer, R. In: Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information. Proc. of DS-8. Kluwer Academic Publ (1999) 351–369
8. Fensel: OIL: An ontology infrastructure for the semantic web. IEIS **16** (2001) 38–45

9. Ullman: Principles of Database and Knowledge-Base Systems, Rockville (Md.) (1988)
10. Dell'Armi, T., Faber, W., Ielpa, G., Leone, N., Pfeifer, G.: Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV. In: Proc. IJCAI 2003, Acapulco, Mexico, Morgan Kaufmann Publishers (2003)
11. Paliouras: Learning rules for large vocabulary word sense disambiguation. In: Proc. of IJCAI-99. (1999) 674–679
12. Faber, W., Pfeifer, G.: `DLV` homepage (since 1996) `http://www.dlvsystem.com/`.
13. Yang: A comparative study on feature selection in text categorization. In: International Conference on Machine Learning, ACL (1997) 412–420
14. Brill: Tranformation-based error-driven learning and natural language processing: A case study in part of speech tagging. In: Computational Linguistics. (1995) 543–565