An XML Based Environment in Support of the Overall KDD Process

Piero Alcamo, Francesco Domenichini and Franco Turini
Dipartimento di Informatica, Università di Pisa
Corso Italia 40, I-56125 Pisa, Italy
alcamo@di.unipi.it domenich@di.unipi.it turini@di.unipi.it

November 10, 2000

Abstract

An XML based environment for Knowledge Discovery in Databases is presented. The aim is to develop an environment in which several kinds of knowledge extraction operations can be combined, in order to describe and solve complex knowledge extraction problems.

Knowledge extraction problems and their results are specified by means of XML documents, and the environment allows the user to follow the extraction process in an interactive way.

A Prolog engine is integrated in the environment in order to use, in the querying process, domain knowledge coded by means of Prolog facts and rules.

The expressiveness and the flexibility of the environment is illustrated by presenting an example where clustering and classification are combined.

The open architecture of the environment makes it easy to extend it in order to support the entire KDD process.

1 Introduction

Because of the importance of its applications, Knowledge Discovery in Databases is a fast growing research area.
Two are the main issues:

1. Designing better knowledge extraction and data mining tools
2. Creating a uniform environment in support of the overall KDD process.

Important characteristics of such an environment are the possibility of an interactive and stepwise application of the tools, the integration of their results, and the possibility of using background knowledge of the application domain.

Our work follows this line with the objective of making different data mining tools inter-operable and the possibility of interactively perform the KDD process by using a rich query language.

1.1 The problem

We believe that an environment in which all the steps of the KDD process can be viewed in an uniform way is a fundamental research issue.

As a first step towards this objective, we realized an environment that can support the Data Mining step of the KDD process.

In order to do so we thought that a common representation for KDD problems and their results was fundamental; this common representation allows the definition of an environment with the following characteristics:

1. **Inter-operability between different Data Mining tools**
   Data Mining tools have reached a good level of efficiency and scalability. However, due to their use of private data models for their results, they are not able to inter-operate.

2. **Flexibility and extendibility**
   KDD is an emerging field and it is in continuous evolution. An environment for its support must have an open architecture in order to be easily extended when new technologies appear.

3. **Expressiveness of the query language**
   Knowledge extraction problems are complex and heterogeneous and a query language able to describe them must have a good level of expressiveness.

1.2 The approach

Our aim, as mentioned above, is to define a common representation for knowledge extraction problems and their results, and then to show how an environment based on such a representation can support the KDD process. To reach a good level of inter-operability as mentioned in point 1 above we have chosen to use XML for representing inputs and outputs of data mining tools. XML is an emerging standard for data representation and exchange and its use in this context can give us the level of inter-operability that is needed.

The use of XML as a representation language give us the possibility of building a flexible and extensible environment. In fact the rapidly evolving world of XML keeps producing essential technologies for making our environment more and more usable.

Our choice of using XML as the target language for the results of Data Mining algorithms has guided us in the choice of the language for the representation of the knowledge extraction problems. In fact in order to make problems and their results closer, we have chosen to represent also knowledge extraction problems in XML itself.

The choice of XML has led us to the definition of an XML based mark-up language named KDDML in which Knowledge extraction problems and their results are represented as XML documents. Representing Knowledge extraction problems as XML documents makes them independent from the tools used for their solution. In fact an XML document defines only what to do, without specifying how to do it.

Upon this mark-up language we have developed, using some XML related technologies, a complete environment in support of the Data Mining step of the KDD process. The environment is entirely written in Java, and we have used some tools developed by the IBM AlphaWorks group that will be briefly described in the next section.

As to the representation of domain knowledge to be used in the overall KDD process, we have chosen to integrate into our environment, again by using XML, a Prolog engine.

Domain knowledge can then be naturally represented in a logic based fashion

2 XML: eXtensible Markup Language

XML, "Extensible Mark-up Language" is a standard recently approved by the W3C that is widely believed to become a universal format for data exchange on the Web [W3CXML].

XML supports electronic exchange of machine-readable data on the web, whereas HTML currently support exchange of human-readable documents.

XML is a strict fragment of SGML, and it is more powerful than HTML in three major respect:

1. Users can define new tag names at will.
2. Document structures can be nested to any level.
3. Any XML document can contain an optional description of its grammar to be used by applications that need to perform structural validation. The grammar portion is called *Document Type Description* (DTD).

Data in XML are grouped into elements delimited by tags, and elements can be nested. Figure 1 illustrates an example of XML data. The strings enclosed between < and >, like **Person, firstname, lastname, e-mail, link**, are called *tags*. Each start-tag <abc> must have a matching end-tag </abc>, and the XML fragments between these matching tags are called *element*.

Data in XML are self-describing semi-structured data.

As mentioned above, XML allows the user to define the structure of his tags, by means of a simple grammar in which it is possible to specify regular expressions between elements (DTD), in order to define how elements can be nested It is also possible to characterize the elements by means of attributes.

An example of such a declaration is presented in Figure 2 for the Personal data of Fig. 1.

A full data model for XML documents is currently not yet defined. However the W3C has proposed a set of interfaces that allows one to represent, access and modify XML documents. This set of interfaces is known as Document Object Model (DOM) [W3CDOM] in the literature.

```
<Personal>
  <Person id="121">
     <firstname> Paolo</firstname>
     <lastname> Rossi </lastname>
     <e-mail> rossi@azienda.com</e-mail>
     <link subordinated="122"/>
  </Person>
  <Person id="122">
     <firstname> Pietro</firstname>
     <lastname> Verdi </lastname>
     <link manager="121"/>
  </Person>
</Personal>
```

Figure 1: An Example of XML Data

```
<!ELEMENT Personal (Person+)>
<!ELEMENT Person(firstname+, lastname, e-mail?, link*)>
<!ATTLIST Person id ID #REQUIRED>
<!ELEMENT firstname #PCDATA>
<!ELEMENT lastname #PCDATA>
<!ELEMENT e-mail  #PCDATA>
<!ELEMENT link EMPTY>
<!ATTLIST link manager IDREF #IMPLIED
               subordinated IDREFS #IMPLIED>
```

Figure 2: An Example of a DTD

An implementation of DOM, called XML4J [XML4J], entirely written in Java, has been realized by the IBM AlphaWorks research group, and it is widely used in the research community. Since it comes as a Java API, it is easy to use and to integrate with other Java based tools.

Another technology, proposed by W3C, is the Extensible Stylesheet Language (XSL), that includes a transformation language to associate XML documents to stylesheets that define their visualization on common browsers.

Such a transformation is realized by a software module called XSL processor, that reads a XML document and the associated stylesheet, and returns the related HTML document.

An well-known and widely used XSL processor is LotusXSL [LotXSL], developed by IBM, that supports the last XSL specification promoted by W3C.

In our system we have used XML4J to retrieve the information contained in XML documents, and LotusXSL to present documents to the user.

3 Environment architecture

In approaching the problem of defining a common representation for knowledge extraction problems and their results we followed the viewpoint of Imielinsky and Mannila [IM 96], who define the KDD process as a querying process, for which an appropriate language has to be defined.

In particular, according to this approach, two are the classes of objects fundamental for the whole process:

KDD object that is a result of a data mining step, e.g.a rule, a classifier, or a clustering.

KDD query that is a predicate which returns a set of objects that can either be KDD objects or database objects such as records or tuples.

To this purpose, we defined, using XML, a mark-up language called KDDML that allow us to represent in a common way KDD objects and KDD queries.

The use of such uniform language provides us with some important advantages:

- A better integration between data mining results;
- The possibility of reusing of data mining results;
- The possibility of performing the extraction process step by step;
- The possibility of sharing the results among different users

KDDML has been used as the starting point to develop the entire environment.

3.1 KDDML

KDDML is a mark-up language that allow one to represent in a uniform way KDD objects and KDD queries as defined above.

As showed in Fig. 3, the root element of KDDML is KDDML_OBJECT, and it derivatives can be:

KDD_QUERY a generic KDD query, i.e. a composition of invocations to external Data Mining algorithms and of by using appropriate operators. Queries can use a conditional operator and, within a conditional, it is possible to query an expert system, that is implemented in Prolog. Furthermore, KDD_QUERIES can be nested

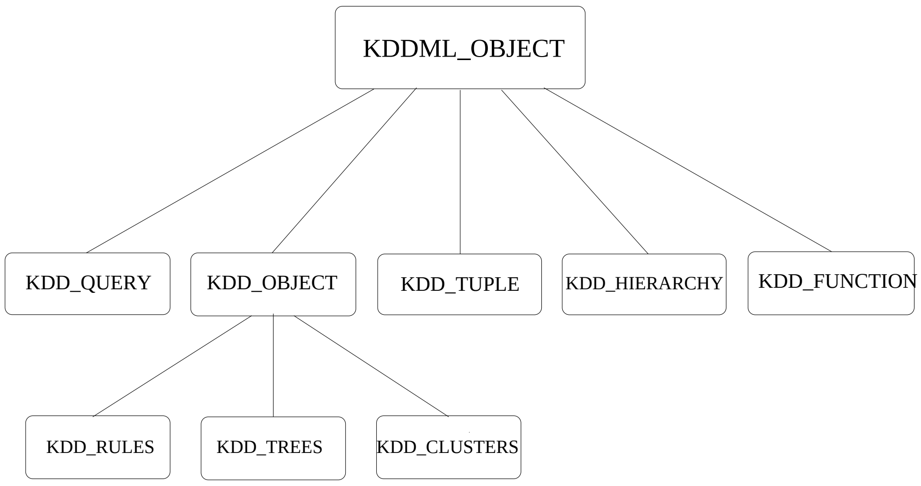


Figure 3: KDDML

KDD_OBJECT i.e. patterns as returned by external Data Mining algorithms. A KDD_OBJECT can be:

- **KDD_RULES** representing associations rules;
- **KDD_TREES** representing classifications trees;
- **KDD_CLUSTERS** representing the result of clustering;

KDD_TUPLE i.e. a database object, such as records or tuples of a database relation, or the result obtained by submitting a query to a Prolog engine;

KDD_HIERARCHY i.e. the representation of a hierarchy on a particular set of items;

KDD_FUNCTION that is used for the representation of the signature of the operators.

It is important to notice that the use of KDDML makes it possible to represent not only KDD objects and KDD queries, but also other objects such as database objects, in order to allow the construction of complex KDD queries that may cross the border between tuples (database objects) and KDD object several times possibly using multiple layers of nesting.

To make the extraction of information at different abstraction levels possible, as suggested in [HF 95], KDDML allows the definition and use of hierarchies on sets of items.

3.2 KDD_RULES

The element KDD_RULES is used to represent a set of association rules as showed in the follow DTD:

```
<!ELEMENT KDD_RULES (RDA+|DOUBLE_RULES+)>
<!ELEMENT RDA (BODY,HEAD,TIME?)>
<!ATTLIST RDA rda_id ID #REQUIRED
              support CDATA #REQUIRED
              confidence CDATA #REQUIRED>
<!ELEMENT BODY (ITEM)+>
<!ELEMENT HEAD (ITEM)+>
<!ELEMENT ITEM EMPTY>
<!ATTLIST ITEM ItemName CDATA #REQUIRED>
<!ELEMENT TIME (CICLE|CALENDAR)>
<!ELEMENT CICLE EMPTY>
<!ATTLIST CICLE length CDATA #REQUIRED
                begin CDATA #REQUIRED>
<!ELEMENT CALENDAR (RANGE)+>
<!ELEMENT RANGE EMPTY>
<!ATTLIST RANGE begin CDATA #REQUIRED
                     end CDATA #REQUIRED>
<!ELEMENT DOUBLE_RULES (RDA,RDA)>
```

In the definition of such DTD we have considered the main components of a generic association rule, that are its Body, Head, support and confidence.

The element KDD_RULES can be composed by one or more RDA elements, representing a single association rule, or by one or more elements named DOUBLE_RULES, used to represent a set of pairs of associations rules, that hold at two distinct level of a hierarchy of items.

The element RDA is composed by an element BODY, representing the body of the rule, an element HEAD, representing the head of the rule, and by an optional element TIME, indicating the time component of an association rule.

The element BODY and the element HEAD are composed by one or more element named ITEM, containing a single item.

The element TIME is composed by an element CYCLE, that represent a cycle of a cyclic association rule, or by an element CALENDAR, representing a calendar of a calendric association rule.

An example of an XML document representing a set of rules is showed on fig. 4.

3.3 KDD_TREES

The element KDD_TREES is used to represent a classification tree or a forest of trees, that can be queried according either to a majority voting or to an and/or strategy, in order to classify new tuples.

The main component of a classification tree are the nodes (the root and the internal nodes), the edges and the leaves; all these components are represented in our language by an XML document.

Furthermore, in our language, it is possible to represent the database schema of the training set.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE KDDML_OBJECT SYSTEM "kdd.dtd">
<KDDML_OBJECT>
  <KDD_RULES>
    <RDA rda_id="rule1" support="0.25" confidence="1">
       <BODY>
          <ITEM ItemName="ski_pants"/>
       </BODY>
       <HEAD>
          <ITEM ItemName="hiking_boots"/>
       </HEAD>
    </RDA>
    <RDA rda_id="rule2" support="0.25" confidence="0.5">
       <BODY>
          <ITEM ItemName="col_shirts"/>
          <ITEM ItemName="jackets"/>
       </BODY>
       <HEAD>
          <ITEM ItemName="brown_boots"/>
       </HEAD>
    </RDA>
</KDD_RULES>
```

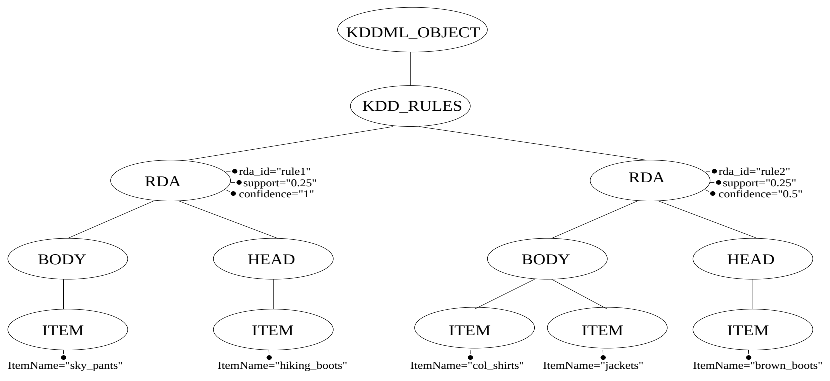


Figure 4: Set of association rules and graphic representation

```
<!ELEMENT KDD_TREES (SCHEMA,(AND_TREES|OR_TREES|COMMITTEE|NODE|LEAF))>
<!ATTLIST KDD_TREES class_att CDATA #REQUIRED>
<!ELEMENT AND_TREES (NODE | LEAF)+>
<!ELEMENT OR_TREES (NODE | LEAF)+>
<!ELEMENT COMITATO (NODE | LEAF)+>
<!ELEMENT EDGE (NODE | LEAF)>
<!ATTLIST EDGE attribute_value CDATA #REQUIRED
                         operator CDATA #REQUIRED>
<!ELEMENT NODE (EDGE)+>
<!ATTLIST NODE attribute_name CDATA #REQUIRED>
<!ELEMENT LEAF EMPTY>
<!ATTLIST LEAF classification CDATA #REQUIRED>
```

The element KDD_TREES is composed by an element SCHEMA, representing the dataset schema, and by an element chosen between:

NODE representing the root;

EDGE if the tree is composed by only one leaf;

AND_TREES, OR_TREES or COMMITTEE , used if we want to represent a forest of trees, combined according to the three different strategies

This element has an attribute class_att, containing the name of the class attribute.

The element SCHEMA is defined by the following DTD:

```
<!ELEMENT SCHEMA (ATTRIBUTE)+>
<!ATTLIST SCHEMA logic_name CDATA #REQUIRED>
<!ELEMENT ATTRIBUTE (INT|ENUMERATED)>
<!ATTLIST ATTRIBUTE name CDATA #REQUIRED>
<!ELEMENT INT EMPTY >
<!ELEMENT ENUMERATED EMPTY>
<!ATTLIST ENUMERATED value CDATA #REQUIRED>
```

This element is composed by one or more elements ATTRIBUTE, each of them specifying the type of a particular field of the training set. The attribute "logic_name" contains the name of such training set.

The element ATTRIBUTE can be composed by an element INT, if the the correspondent field of the training set is numeric, or by an element ENUMERATED, if it refers to an enumerated field.

The element NODE is composed by one or more elements EDGE, representing an outgoing edge from the corresponding node.

The element EDGE can be composed by an element NODE, representing the node linked to the edge, or by an element LEAF, representing a leaf of the tree; this element has two attributes that specify the label of the edge, containing the condition on the attributes.

Finally, the element LEAF is composed by an attribute that specifies the classification value of the leaf.

An example of an XML document representing a classification tree is shown on fig. 5.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE KDDML_OBJECT SYSTEM "kdd.dtd">

<KDDML_OBJECT>
<KDD_TREES class_att="play">
    <SCHEMA logic_name="weather.symbolic">
      <ATTRIBUTE name="outlook">
        <ENUMERATED value="sunny"/>
        <ENUMERATED value="overcast"/>
      </ATTRIBUTE>
      <ATTRIBUTE name="humidity">
        <ENUMERATED value="high"/>
        <ENUMERATED value="normal"/>
      </ATTRIBUTE>
      <ATTRIBUTE name="windy">
        <ENUMERATED value="TRUE"/>
        <ENUMERATED value="FALSE"/>
      </ATTRIBUTE>
      <ATTRIBUTE name="play">
        <ENUMERATED value="yes"/>
        <ENUMERATED value="no"/>
      </ATTRIBUTE>
    </SCHEMA>
    <NODE attribute_name="outlook">
      <EDGE operator="=" attribute_value="sunny">
        <NODE attribute_name="humidity">
          <EDGE operator="=" attribute_value="high">
            <LEAF classification="no"/>
          </EDGE>
          <EDGE operator="=" attribute_value="normal">
            <LEAF classification="yes"/>
          </EDGE>
        </NODE>
      </EDGE>
      <EDGE operator="=" attribute_value="overcast">
        <LEAF classification="yes"/>
      </EDGE>
    </NODE>
  </KDD_TREES>
</KDDML_OBJECT>
```

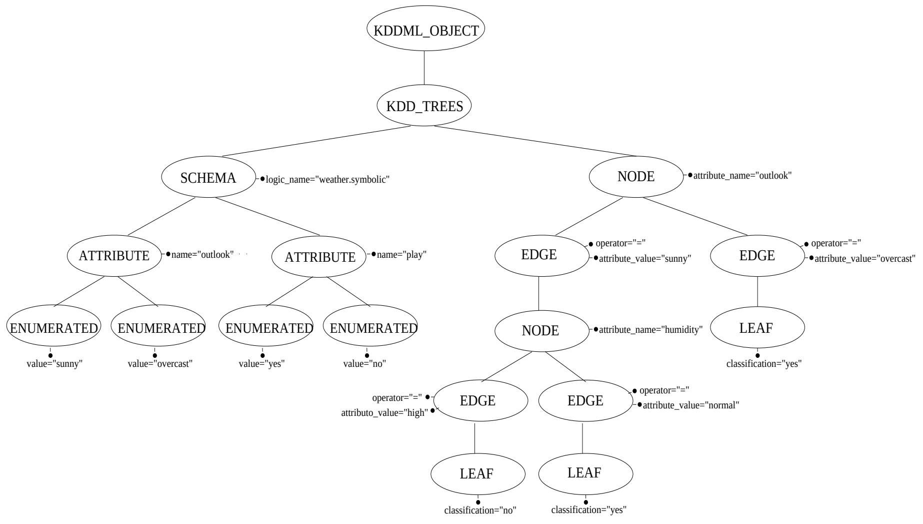


Figure 5: A classification tree and its graphic representation

3.4 KDD_CLUSTERS

This element is used to represent the result of probabilistic clustering.

We chose to represent every single cluster produced by the clustering algorithm by means of its centroid element, from which we can find the tuple set assigned to the cluster.

The elements used to represent a set of clusters are deefined in the following DTD:

```
<!ELEMENT KDD_CLUSTERS (CLUSTER+)>
<!ATTLIST KDD_CLUSTERS source CDATA #REQUIRED
                       algorithm CDATA #REQUIRED
                       num_clusters CDATA #REQUIRED>
<!ELEMENT CLUSTER (CENTROID)>
<!ATTLIST CLUSTER prob CDATA #IMPLIED
                  id CDATA #REQUIRED
                  card CDATA #REQUIRED>
<!ELEMENT CENTROID (ATTR_NOM|ATTR_NUM)+>
<!ELEMENT ATTR_NOM (DISTR_VALUE)+>
<!ATTLIST ATTR_NOM name CDATA #REQUIRED>
<!ELEMENT DISTR_VALORE EMPTY>
<!ATTLIST DISTR_VALORE value CDATA #REQUIRED
                       num_inst CDATA #REQUIRED>
<!ELEMENT ATTR_NUM EMPTY>
<!ATTLIST ATTR_NUM name CDATA #REQUIRED
                   mean CDATA #REQUIRED
                   variance CDATA #REQUIRED>
```

The element KDD_CLUSTERS is composed by one or more elements named CLUSTER, representing a single extracted cluster, and by three attributes that stand for the dataset (“source” attribute), the clustering algorithm in use (“algorithm” attribute), and the number of extracted clusters (“num_clusters” attribute).

The element CLUSTER is composed by an element CENTROID, representing the centroid element of the cluster, and by three attributes that indicates the a priori probability of cluster (“prob” attribute), an identification key (“id” attribute), and the number of tuples assigned to the cluster (“card” attribute).

The element CENTROID is composed by one or more element that can be ATTR_NOM or ATTR_NUM elements, representing a single field of the schema dataset; ATTR_NOM is used if the field is nominal, while ATTR_NUM is used when the field is numeric.

The element ATTR_NOM is composed by one or more element DISTR_VALUE, representing a possible value of the nominal field and the number of instances in which this value is present.

The element ATTR_NUM is composed by three attributes that contain the name of the field, the mean and the standard deviation.

An example of an XML document representing a clustering is showed on fig. 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE KDDML_OBJECT SYSTEM "/tesi/ambiente/kddlang/kdd.dtd" >
<KDDML_OBJECT>
  <KDD_CLUSTERS source="/arff-files/iris.arff" algorithm="EM" num_clusters="3">
    <CLUSTER id="0" prob="0.3333" card="50">
      <CENTROID>
        <ATTR_NUM name="sepallength" mean="5.006" stdev="0.3525"/>
        <ATTR_NUM name="sepalwidth" mean="3.418" stdev="0.381"/>
        <ATTR_NOM name="class">
          <DISTR_VALUE value="Iris-setosa" num_inst="50"/>
          <DISTR_VALUE value="Iris-virginica" num_inst="0"/>
        </ATTR_NOM>
      </CENTROID>
    </CLUSTER>
    <CLUSTER id="1" prob="0.32237" card="48">
      <CENTROID>
        <ATTR_NUM name="sepallength" mean="6.6306" stved="0.5977"/>
        <ATTR_NUM name="sepalwidth" mean="2.9977" stved="0.3042"/>
        <ATTR_NOM name="class">
          <DISTR_VALUE value="Iris-setosa" num_inst="0"/>
          <DISTR_VALUE value="Iris-virginica" num_inst="47.96"/>
        </ATTR_NOM>
      </CENTROID>
    </CLUSTER>
  </KDD_CLUSTERS>
</KDDML_OBJECT>
```

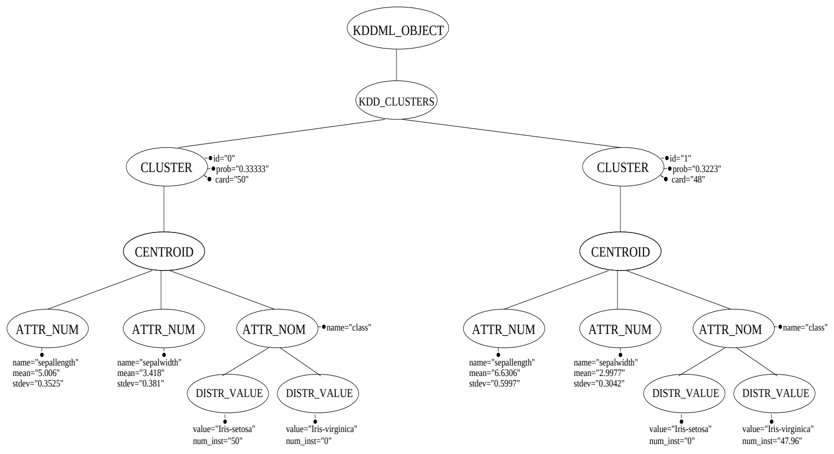


Figure 6: A set of clusters and graphic representation

3.5 KDD_QUERY

The element KDD_QUERY is used to represent a generic query.

As mentioned above, the result of a KDD query must be a set of either KDD objects or database objects. Furthermore the query language must allow the reuse of previous knowledge, and, above all,it must support a closure principle in order to combine and refine the extracted knowledge.

To do that, a query must have a nested structure, in which it should be possible to combine an arbitrary number of sub-queries containing invocations to external Data Mining algorithms, combined by operators. Moreover, it is necessary to check that sub-queries are properly nested, in order to avoid that a sub-query returns a result that doesn't meet the requirements of the operators combining them.

For example, suppose to have the query: *"Find the database tuples that are exceptions to the set of associations rules obtained by filtering the result of the Apriori algorithm on a database relation"*

Notice that there are three fundamental steps (sub-queries) in the previous query:

1. Extraction of the association rules by using the Apriori algorithm;
2. Filtering the extracted association rules;
3. Extraction of tuples that are exceptions to the filtered associations rules.

In order to control the nesting of sub-queries, we group the invocations of external Data Mining algorithms and operators that returns the same kind of knowledge in the same class, defined by means of a XML entities, as shown in the list below:

data_source invocations to operators that return databases objects; such operators can be also realized as predicates querying a Prolog expert system, and returning the set of ground answers as database tuples. In this way, during the overall KDD process we can refer both to extensional knowledge, i.e. data held in a database, and to intensional knowledge, i.e. data that can be **"inferred"** by using the deductive capabilities of a Prolog engine, or a deductive database in general. This last feature allows us a natural representation of background knowledge.

kdd_query_trees invocations to external Data Mining algorithms and operators that return classification trees;

kdd_query_rules invocations to external Data Mining algorithms and operators that returns association rules;

kdd_query_clusters invocations to external Data Mining algorithms and operators that returns sets of clusters.

In order to illustrate these concepts, we show a portion of Document Type Definition of the KDD_QUERY element. It must be composed by one element that denotes an invocation to an external Data Mining algorithm (contained in the *external_call* entity) or the invocation to operators returning the same kind of results (contained in the *learning_op* entity):

```
<!ELEMENT KDD_QUERY (%external_call;|learning_op;)>
```

As an example of invocation to an external Data Mining algorithm we show the element definition related to the MINE_RULE element:

```
<!ELEMENT MINE_RULE (APRIORI,%data_source;)>
<!ATTLIST MINE_RULE xml_dest CDATA #IMPLIED>
<!ELEMENT APRIORI EMPTY>
<!ATTLIST APRIORI number_of_rules CDATA #IMPLIED
                  min_support CDATA #IMPLIED
                  min_confidence CDATA #IMPLIED >
```

The element MINE_RULE is composed by an element that represents the specific algorithm, Apriori in this case , and an element of the data_source class, that represents the invocation to operators that returns tuples.

The element APRIORI is empty and contains some attributes representing the interface to the algorithm.

An example of MINE_RULE use is the following:

```
<MINE_RULE xml_dest="MineBasket.xml">
   <APRIORI min_confidence="0.6" min_support="0.4"
                                 number_of_rules="30"/>
   <FILE_ARFF nome_file="coop.arff"/>
</MINE_RULE>
```

In this example we choose the element FILE_ARFF from the class data_source as the input.

The entity external_call also contain the definition of two other elements:

MINE_TREE for the extraction of classification trees, using two algorithms: C4.5, ID3

MINE_CLUSTERS for the extraction of clusters, using the EM algorithm.

As an example of an invocation to an operator we show the element PRESERVED_RULES:

```
<!ELEMENT PRESERVED_RULES (%kdd_query_rules;,%kdd_query_rules;,
                                                   HIERARCHY_FILE)>
<!ATTLIST PRESERVED_RULES xml_dest CDATA #REQUIRED>
```

This element represents the invocation to the operator that determines the set of pairs of rules that hold at two different abstraction level of a hierarchy, and it is composed by two sets of rules (chosen from the kdd_query_rules class defined above), and by an element that represents the hierarchy.

An example of its use is the following:

```
<PRESERVED_RULES xml_dest="Coop.xml">
  <MINE_RULE>
    <APRIORI min_confidence="0.7" min_support="0.4"
             number_of_rules="10"/>
    <FILE_ARFF nome_file="coop.arff"/>
  </MINE_RULE>
  <FILE_RULES nome_file="FilterMineCoopGen.xml"/>
  <FILE_HIERARCHY nome_file="categoryCoop.xml"/>
</PRESERVED_RULES>
```

In this example the two sets of rules needed by PRESERVED_RULES are specified by means of the MINE_RULE element (defined above), and by FILE_RULES element that stands for the reference to an external file containing a set of rules.

In the entity *learning_op* we have also defined others operators on algorithm results, that are listed above:

FILTER_RULES given a set of rules returns the ones, that satisfy a filter specified by a boolean formula.

RULE_SUPPORT given a set of tuples returns the ones that support a specified set of rules.

RULE_EXCEPTION given a set of tuples returns the ones that are exceptions to a specified set of rules.

AND_TREE, OR_TREE allow us to query a set of trees according either to an *and* or to an *or* strategy.

COMMITTEE allows us to query a set of trees according to a majority voting strategy.

CLASSIFY classifies a given set of tuples, using a classification tree.

MAX_OF returns the set of tuples that belongs to the cluster of maximum cardinality.

N_CLUSTER_OF returns the set of tuples that belong to a cluster specified by its identification number.

PROLOG_QUERY executes a specified Prolog query (on specified sets of facts and rules), and returns the ground answers as a set of tuples; when no answers are found it returns an empty set of tuples.

All these operators can be **conditioned**, in the sense that a conditional operator can be used to combine two alternatives subqueries, that are selected according to the evaluation of a condition, which is evaluated on a particular relation (i.e. a set of tuples).

The operator that implements such a choice is called COND, and it has the following DTD:

```
<!ELEMENT COND_OP (CONDITION?, %data_source,
                   (%external_call|%meta_learning_op),
                   (%external_call|%meta_learning_op))>
<!ATTLIST COND_OP xml_dest CDATA #IMPLIED>
```

As showed above an instance of the COND_OP operator must contain:

1. An (optional) element representing a condition, i.e. a boolean formula regarding the values contained in a set of tuples; using XML we have defined a simple grammar that allows us to represent also quantified conditions, i.e. boolean conditions in the range of a standard quantifier (for each, exists). When no condition is specified, it means that we want to check the emptiness of a particular set of tuples; when this set of tuples is computed by querying the prolog system, this correspond to checking a condition implemented in an expert system style.

2. An element representing a set of tuples, on which the previous condition is evaluated; obviously the set of tuples can be computed by using any of the operators returning set of tuples, including queries to Prolog based deductive databases.

3. Two elements representing the alternatives sub_queries.

As an example suppose that we want to extraxt a set of associations rules, and that we want to choose the data set to use according to the evaluation of a condition on the results of a Prolog query. If the Prolog query is Prolog_Query(X,Y) (which facts and rules are specified in a file called Prolog_query.pl) and we want to check that exists a solution in which X values 10 and Y values 5, we have to write the following query:

```
<MINE_RULE xml_dest=''cond_rules''>
  <APRIORI min_confidence="0.7" min_support="0.4" number_of_rules="10"/>
  <COND_OP>
    <CONDITION quantifier=''exists''>
       <AND_C>
          <VARIABLE_CONDITION name=''X'' value=''10''>
          <VARIABLE_CONDITION name=''Y'' value=''5''>
       </AND_C>
    </CONDITION>
    <PROLOG_QUERY pred_name=''Prolog_Query'' facts_rules=''Prolog_Query.pl''>
      <VARIABLE name=''X''/>
      <VARIABLE name=''Y''/>
    </PROLOG_QUERY>
    <FILE_ARFF name=''coop.arff''/>
    <FILE_ARFF name=''basket.arff''/>
  </COND_OP>
</MINE_RULE>
```

The example above shows also that KDDML, like all others markup languages, is not an user friendly language, and specify a query using it may be (expecially for non XML experts) quite complex. However, as we will describe later, we have developed a GUI in wich the user can find a syntax driven editor by wich specify its queries.

The COND_OP operator can be specified at any nesting level in a query, and the syntax driven editor mentioned above will force us to specify the rigth type of knowledge as alternative subqueries.

3.6 System Architecture

KDDML is the basis on which we constructed our environment in support of Knowledge Extraction.

In defining the environment we chose to handle Data Mining algorithms as black boxes.

Fig. 7 depicts the whole architecture. As shown in the figure, the neutral representation provided by XML/DOM plays a central role.

To build our environment we have used four *"External components"*:

XML4J to support XML/DOM neutral representation [XML4J]

WEKA a Java based package of data mining tools [WEKA]

LOTUSXSL to transform XML document in HTML document visualizable on common browsers

SICSTUS PROLOG a Prolog engine easy to use in Java based applications [LotXSL]

Below we briefly describe the functionalities of the principal components:

3.6.1 Query Executor

This is the key component of the environment. Its aim is to resolve knowledge extraction problems expressed by KDD_QUERY in KDDML.

The Query Executor component gets the KDD_QUERY, in the form of a DOM tree, from the Graphic User Interface. In order to execute such a query, it performs a visit of the DOM tree and it performs the operations that it encounters.

When it encounters an external call to a Data Mining algorithm, it interacts with the WEKA component, in order to extract the required knowledge. The results are fed to the Query Executor component via the Wrappers, that take care of their translation in the KDDML format.

When it encounters an operator call, it cooperates with the Operators component, that executes the required operator and returns the result in KDDML format.

All results obtained in this process are translated, by using the LOTUSXSL component, into HTML format and presented to the user inside a browser.

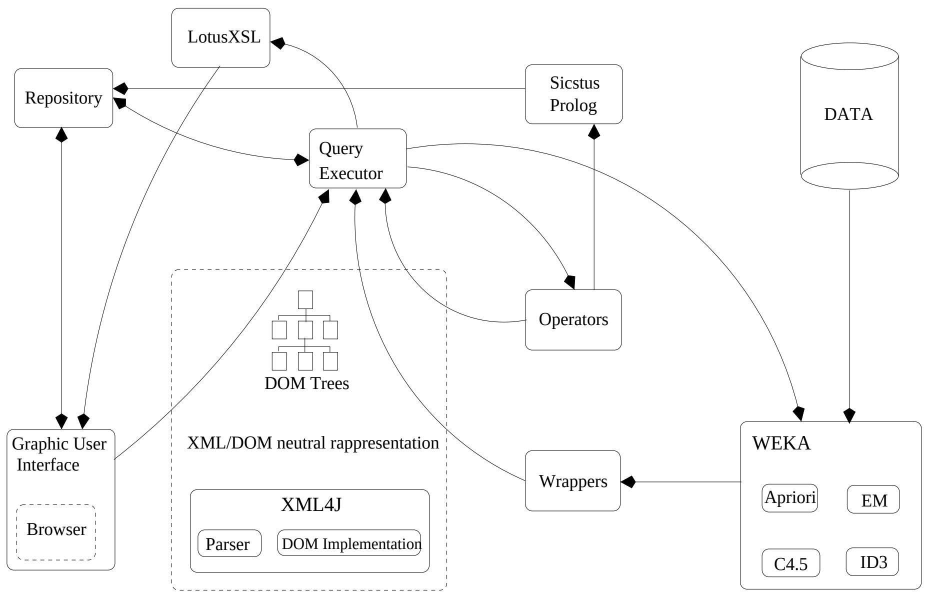


Figure 7: Environment Architecture

3.6.2 Graphic User Interface

This component allows the user to interact with the environment and to exploit its characteristics.

The GUI is divided in two parts:

- A generic component for handling knowledge extraction problems, and to retrieve previously extracted knowledge. The construction of a new query is supported by a syntax driven editor that guides the user in the process.
- A browser that allow the visualization of the HTML pages representing the results.

3.6.3 Wrappers

This component takes care of taking the results produced by the WEKA component and to transform them into DOM trees. After performing this transformation, the Wrapper component returns the result to the Query Executor component.

3.6.4 Operators

This component provides the implementation of the operators defined in the KDDML language. Every time it is invoked, it executes the required opera-

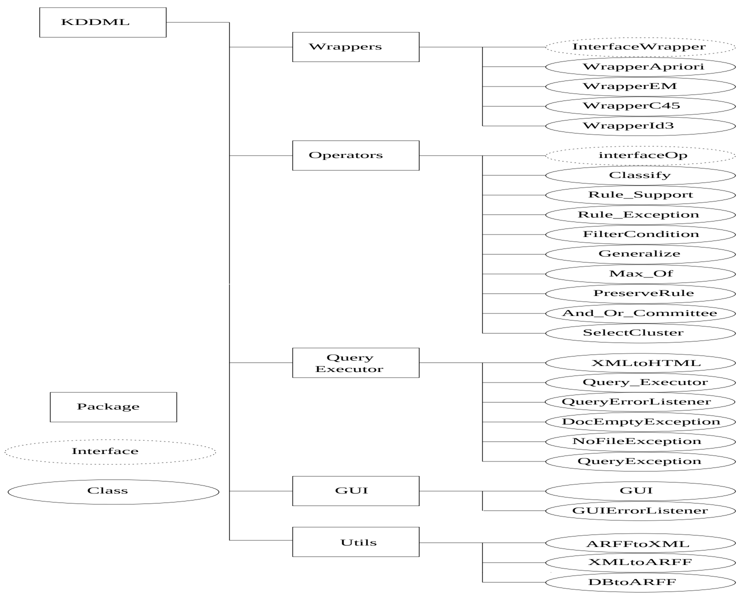


Figure 8: Packages

tor and returns the result, in the form of DOM tree, to the Query Executor component.

3.6.5 Repository

This component is used to store the queries created and the results obtained, in order to retrieve them later for further use.

The environment described above is entirely implemented in Java, and the code is organized in packages, as showed in fig. 8, each of them implements the functionalities of the corresponding component of the environment.

4 How to use KDDML

The aim of this section is to show, by meansof two examples, the potentialityof our markup language and of the overall environment.

In the first example we will show how our environment make easy to combine several types of knowledge, solving a knowledge extraction problem regarding clustering and classification.

The combination of several types of knowledge is fundamental for an environment in support of the KDD process, along with the possibility to exploit a certain type of background knowledge.

In the second example we will show how background knowledge can be exploited, querying an expert system in order to decide what to do with a certain type of extracted knowledge.

4.1 Example 1: combining several type of knowledge

As mentioned above, the aim of this example is to show how our environment supports the Data Mining step of the KDD process, and how it allows the combination of several knowledge extraction techniques.

We will consider a widely used dataset, available from the UCI repository [UCI], in which 214 types of glass are described by means of 9 numeric attributes, and grouped in 7 classes with reference to their commercial use.

The problem is to generate a classifier and to use it to classify new instances.

We follow the strategy illustrated in fig. 9:

1. a first clustering on the Glass dataset;
2. a second clustering;
3. the extraction of a set of classification trees from the clusters obtained in steps 1 and 2;
4. the combination of the classification trees obtained in step 3 in order to construct a new classifier, and classification of a new set of instances.

Is worth observing that to implement the above strategy it is necessary to combine clustering operations with the extraction of classification trees.

4.1.1 Step 1: Clustering on the Glass dataset

The graphic user interface provides a syntax driven editor that allows the user to specify only valid and well-formed KDD queries.

By using the editor the user can specify the query relative to the first clustering step, as showed in fig. 10.

This query specifies that the tuples contained in the file "glass.arff" must be partitioned in 3 clusters by using the EM algorithm with at most 100 iterations.

When the user executes the query, it is passed to the Query_Executor in the form of a DOM tree. The Query_Executor performs a depth-first search resolving all the sub-queries.

In this case no sub-query must be resolved, and the Query_Executor will invoke directly the EM algorithm using the WrapperEM class, in order to obtain the result as a KDDML document.

Finally the Query_Executor will use the XMLtoHTML class in order to present the user with the results in the shape of HTML pages, as showed in fig. 11.

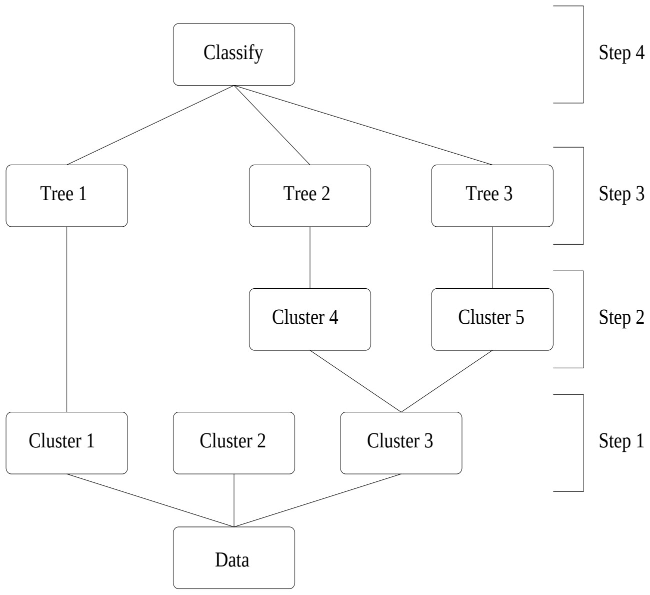
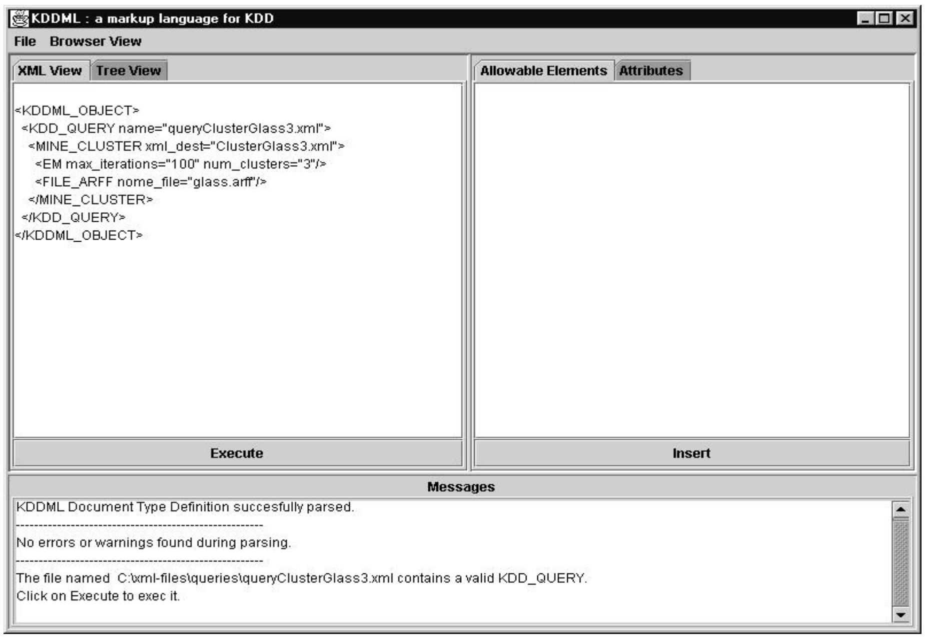


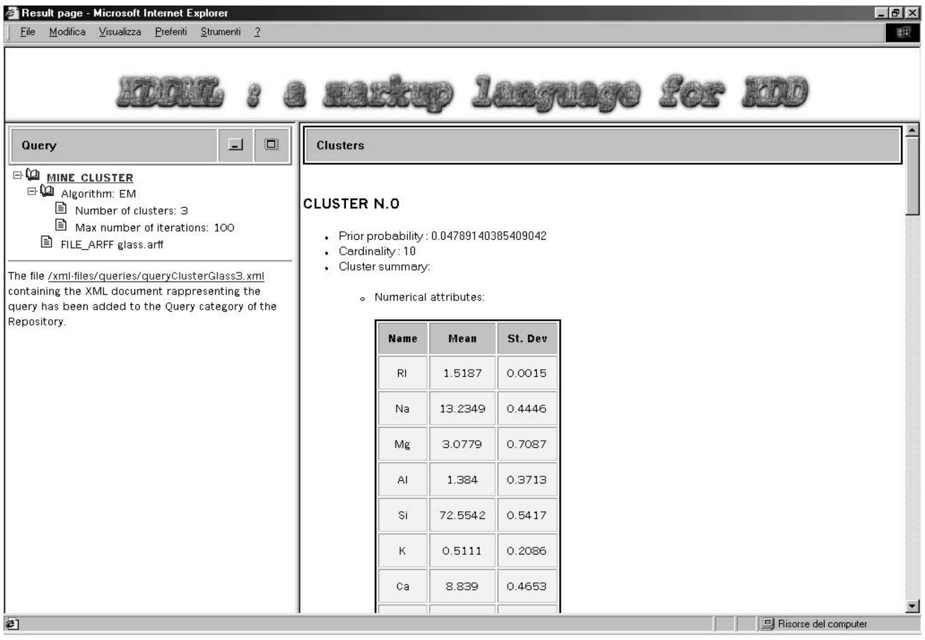
Figure 9: Strategy



```
<KDDML_OBJECT>
 <KDD_QUERY name="queryClusterGlass3.xml">
  <MINE_CLUSTER xml_dest="ClusterGlass3.xml">
   <EM max_iterations="100" num_clusters="3"/>
   <FILE_ARFF nome_file="glass.arff"/>
  </MINE_CLUSTER>
 </KDD_QUERY>
</KDDML_OBJECT>
```

```
KDDML Document Type Definition succesfully parsed.
----------------------------------------------------
No errors or warnings found during parsing.
----------------------------------------------------
The file named  C:\xml-files\queries\queryClusterGlass3.xml contains a valid KDD_QUERY.
Click on Execute to exec it.
```

Figure 10: Graphic user interface


Result page - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
KDDML : a markup language for KDD
Query
Clusters
Risorse del computer

MINE CLUSTER
Algorithm: EM
Number of clusters: 3
Max number of iterations: 100
FILE_ARFF glass.arff

The file /xml-files/queries/queryClusterGlass3.xml containing the XML document rappresenting the query has been added to the Query category of the Repository.

CLUSTER N.0

- Prior probability : 0.04789140385409042
- Cardinality : 10
- Cluster summary:
  - Numerical attributes:

Name	Mean	St. Dev
RI	1.5187	0.0015
Na	13.2349	0.4446
Mg	3.0779	0.7087
Al	1.384	0.3713
Si	72.5542	0.5417
K	0.5111	0.2086
Ca	8.839	0.4653

Figure 11: Step 1: Clustering on the Glass dataset

All the HTML pages designed to present the results of a data mining step are structured in the same way, that is the representation of the query in the left frame (with links to partial results), and the representation of the final result in the right frame.

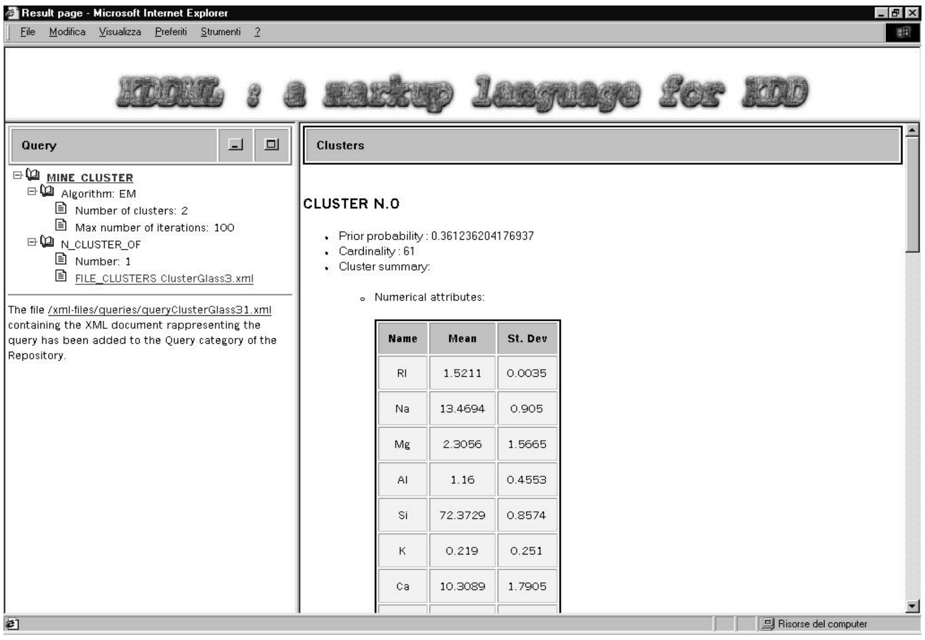
4.1.2 Step 2: Second clustering step

To obtain a better partitioning of the glass dataset, we have decided to perform a further clustering operation in order to partition the tuples of the third cluster obtained in the previous step.

The query that specifies this operation and its results are presented in fig 12.

This query is very similar to the one of the previous step; the only difference is that the dataset on which the EM algorithm must operate is obtained by using the N_CLUSTER_OF operator, that returns the tuples assigned to a particular cluster.

In this case, therefore, the Query_Executor finds a sub-query to resolve that produce the tuples that must be partitioned as partial results.


Result page - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
KDDML : a markup language for KDD
Query
Clusters
Risorse del computer

- MINE CLUSTER
  - Algorithm: EM
    - Number of clusters: 2
    - Max number of iterations: 100
  - N_CLUSTER_OF
    - Number: 1
    - FILE_CLUSTERS ClusterGlass3.xml

The file /xml-files/queries/queryClusterGlass31.xml containing the XML document rappresenting the query has been added to the Query category of the Repository.

CLUSTER N.0

- Prior probability : 0.361236204176937
- Cardinality : 61
- Cluster summary:
  - Numerical attributes:

Name	Mean	St. Dev
RI	1.5211	0.0035
Na	13.4694	0.905
Mg	2.3056	1.5665
Al	1.16	0.4553
Si	72.3729	0.8574
K	0.219	0.251
Ca	10.3089	1.7905

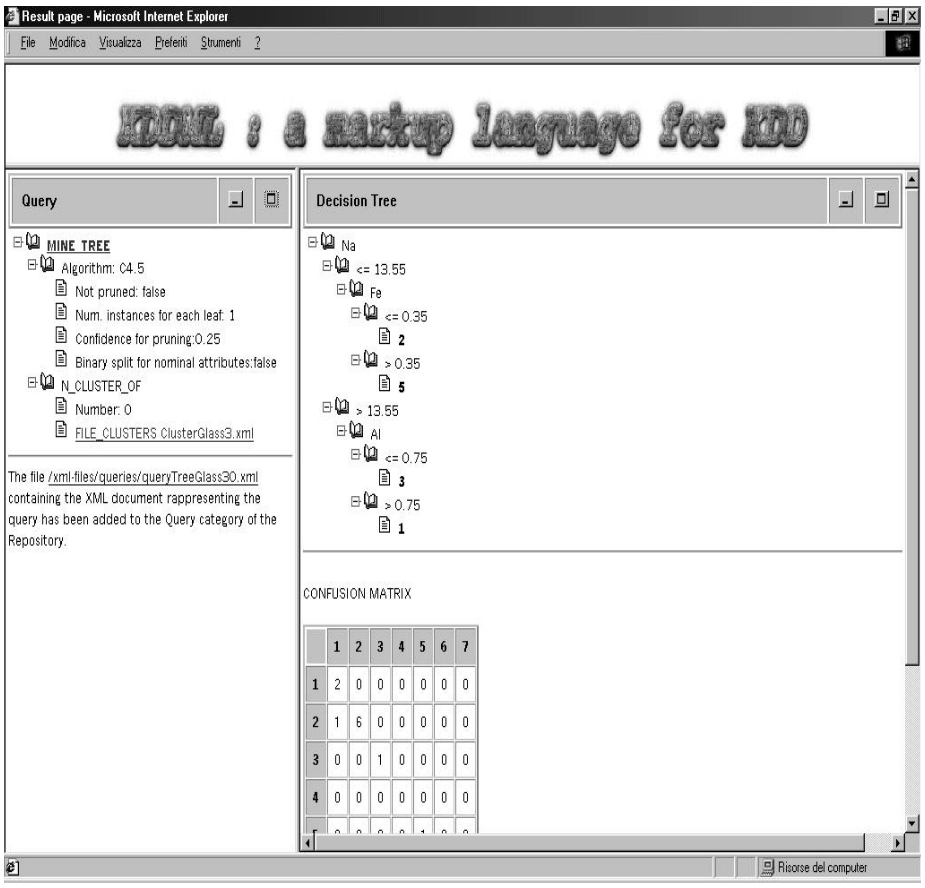
Figure 12: Step 2: Second clustering step

4.1.3 Step 3: Extraction of a set of classification trees

The aim of this step is to extract three classifications trees using the clusters obtained in the previous steps.

The three queries that specifies this operations are very similar, and each of them invokes the C4.5 algorithm on the tuples of the specified cluster.

An example of query and the relative result is showed in fig. 13; in this case too, the Query_Executor must retrieve the tuples belonging to the specified clusters in order to extract the valid classification tree.


Result page - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
KDDML : a markup language for KDD
Query
Decision Tree
Risorse del computer

MINE_TREE
- Algorithm: C4.5
  - Not pruned: false
  - Num. instances for each leaf: 1
  - Confidence for pruning:0.25
  - Binary split for nominal attributes:false
- N_CLUSTER_OF
  - Number: 0
  - FILE_CLUSTERS ClusterGlass3.xml

The file /xml-files/queries/queryTreeGlass30.xml containing the XML document rappresenting the query has been added to the Query category of the Repository.

```
Na
  <= 13.55
    Fe
      <= 0.35
        2
      > 0.35
        5
  > 13.55
    Al
      <= 0.75
        3
      > 0.75
        1
```

CONFUSION MATRIX

	1	2	3	4	5	6	7
1	2	0	0	0	0	0	0
2	1	6	0	0	0	0	0
3	0	0	1	0	0	0	0
4	0	0	0	0	0	0	0

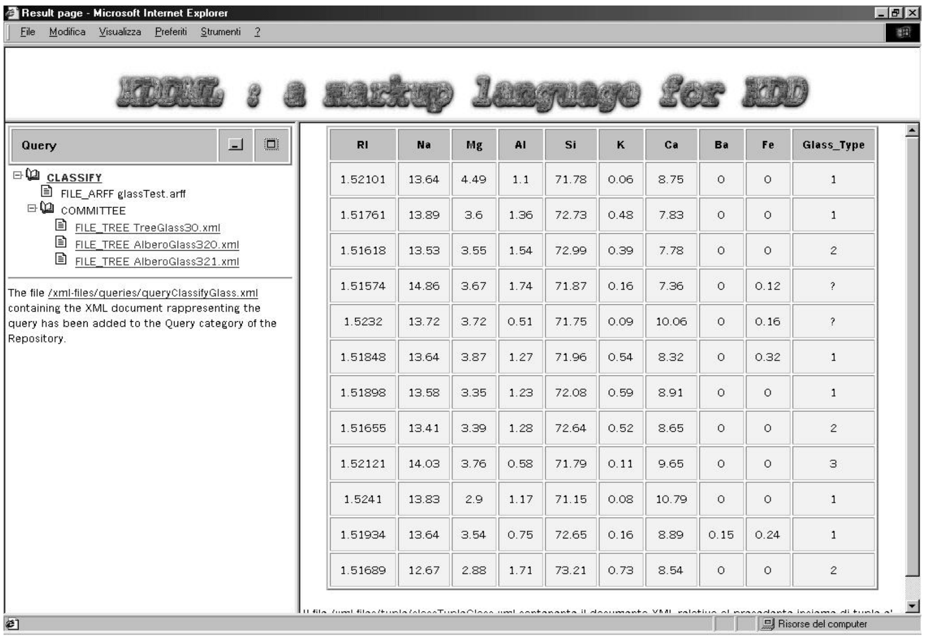
Figure 13: Step 3: Extraction of a set of classification trees

4.1.4 Step 4: Classification of a new tuple set

The aim of this step is to classify a new set of tuples by using the classification trees obtained in the previous step.

The query and its results are shown in fig. 14; in this case, the Query_Executor must retrieve the three classification trees referred by name in the element FILE_TREE and combine them.

The classification value for each instance is obtained by a majority voting strategy; it is important to observe that some tuples are not classified because a majority vote has not been reached.


Result page - Microsoft Internet Explorer
File Modifica Visualizza Preferiti Strumenti ?
KDDML : a markup language for KDD
Query
Risorse del computer

CLASSIFY
FILE_ARFF glassTest.arff
COMMITTEE
FILE_TREE TreeGlass30.xml
FILE_TREE AlberoGlass320.xml
FILE_TREE AlberoGlass321.xml

The file /xml-files/queries/queryClassifyGlass.xml containing the XML document rappresenting the query has been added to the Query category of the Repository.

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Glass_Type
1.52101	13.64	4.49	1.1	71.78	0.06	8.75	0	0	1
1.51761	13.89	3.6	1.36	72.73	0.48	7.83	0	0	1
1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0	0	2
1.51574	14.86	3.67	1.74	71.87	0.16	7.36	0	0.12	?
1.5232	13.72	3.72	0.51	71.75	0.09	10.06	0	0.16	?
1.51848	13.64	3.87	1.27	71.96	0.54	8.32	0	0.32	1
1.51898	13.58	3.35	1.23	72.08	0.59	8.91	0	0	1
1.51655	13.41	3.39	1.28	72.64	0.52	8.65	0	0	2
1.52121	14.03	3.76	0.58	71.79	0.11	9.65	0	0	3
1.5241	13.83	2.9	1.17	71.15	0.08	10.79	0	0	1
1.51934	13.64	3.54	0.75	72.65	0.16	8.89	0.15	0.24	1
1.51689	12.67	2.88	1.71	73.21	0.73	8.54	0	0	2

Figure 14: Step 4: Classification of a new tuple set

4.2 Example 2: using background knowledge

As mentioned above an important capability of our system is the possibility to exploit background knowledge, and now we will show an illustrative example on how to do it.

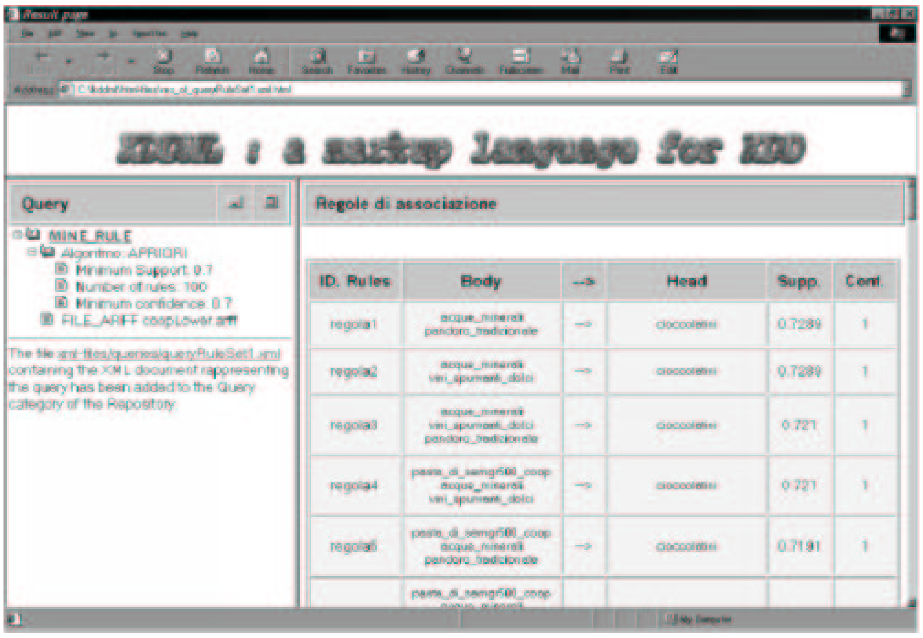
We will use a market basket dataset, that contains transactions of an Italian Supermarket done during Christmas period; the problem we want to solve is to find a set of associations rules and then if we find them "interesting", according to our background knowledge, determine which of them survive or decay upon a product hierarchy.

The role of our background knowledge is central in this process, and an expert system can do it easily and in the right way; in this example we will suppose to have a Prolog coded expert system that can be queried in order to know if a certain set of rules is interesting or not.

The interestingness of a set of rule may be determined by quantitative parameters, such as the amount of the transactions related to them, or by qualitative parameters, such as the presence of certain product inside them.

The expert system we suppose to use in this example is able to check if a certain set of rules, relative to a certain data set, matches those parameters, and so if it is "interesting" according to our background knowledge.

The expert system has an interface of the form "Check_Rules(X,Y)" where X is the name of the XML document containg the set of rules, and Y is the



KDDML : a markup language for KDD

Query

MINE_RULE
Algoritmo: APRIORI
Minimum Support: 0.7
Number of rules: 100
Minimum confidence: 0.7
FILE_ARFF coopLower.arff

The file xml-files/queries/queryRuleSet1.xml containing the XML document rappresenting the query has been added to the Query category of the Repository

Regole di associazione

ID. Rules	Body	-->	Head	Supp.	Conf.
regola1	acque_minerali pandoro_tradizionale	-->	cioccolatini	0.7289	1
regola2	acque_minerali vini_spumanti_dolci	-->	cioccolatini	0.7289	1
regola3	acque_minerali vini_spumanti_dolci pandoro_tradizionale	-->	cioccolatini	0.721	1
regola4	pasta_di_semg500_coop acque_minerali vini_spumanti_dolci	-->	cioccolatini	0.721	1
regola5	pasta_di_semg500_coop acque_minerali pandoro_tradizionale	-->	cioccolatini	0.7191	1
	pasta_di_semg500_coop				

Figure 15: Step 1: Extraction of associations rules

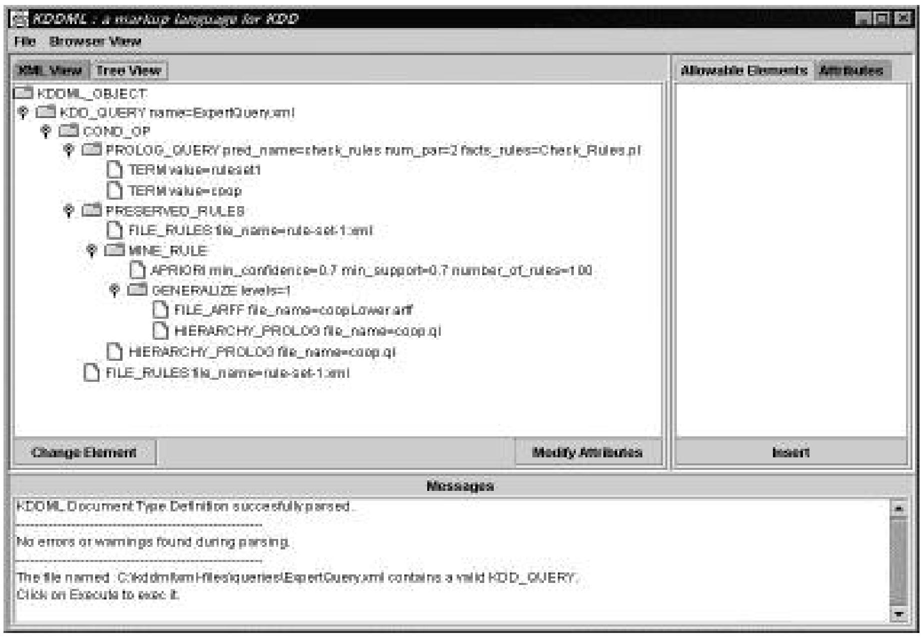
name of the dataset used to extract the rules.

To solve this knowledge extraction problem we have to do two fundamental steps:

1. Extract a set of association rules from the data set
2. Query the expert system, and if those rules are interesting determine which of them survive upon a particular product hierarchy

4.2.1 Step 1: Extraction of association rules

As mentioned above, in this step we want to extract from our transaction data a set of association rules; in our environment we can use the well known Apriori algorithm to do that, specifying the desired values for support and confidence. The query can easily be composed using the graphic user interface, and the related results are presented in fig. 15; in this query we specify that we want to extract a set of 100 rules from a dataset called “coopLower”, and those rules must have at least a confidence and a support of 0.7.



```
KDDML_OBJECT
  KDD_QUERY name=ExpertQuery.xml
    COND_OP
      PROLOG_QUERY pred_name=check_rules num_par=2 facts_rules=Check_Rules.pl
        TERM value=ruleset1
        TERM value=coop
      PRESERVED_RULES
        FILE_RULES file_name=rule-set-1.xml
        MINE_RULE
          APRIORI min_confidence=0.7 min_support=0.7 number_of_rules=100
          GENERALIZE levels=1
            FILE_ARFF file_name=coopLower.arff
            HIERARCHY_PROLOG file_name=coop.ql
        HIERARCHY_PROLOG file_name=coop.ql
      FILE_RULES file_name=rule-set-1.xml
```

```
KDDML Document Type Definition succesfully parsed.
--------------------------------------------------
No errors or warnings found during parsing.
--------------------------------------------------
The file named C:\kddml\xmlFiles\queries\ExpertQuery.xml contains a valid KDD_QUERY.
Click on Execute to exec it.
```

Figure 16: Step 2: Creating the query

4.2.2 Step 2: Querying the expert system

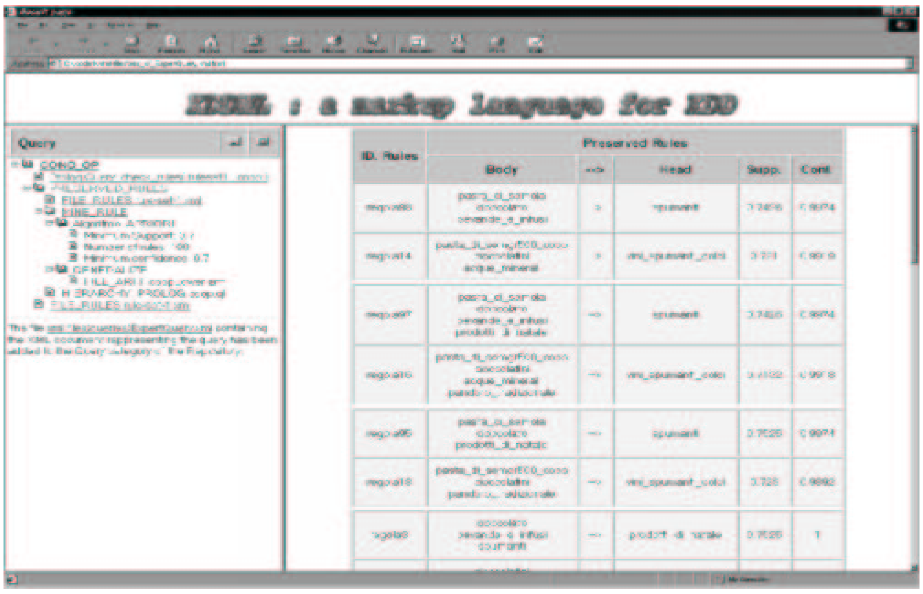
Now we want to check if the rules extracted in the previous step are interesting, and to do that we have to query the expert system mentioned above; as mentioned in section 3.5 in KDDML a Prolog query can be represented by an element called PROLOG_QUERY that is characterized by three attribute regarding the name of the predicate, its ariety, and the Prolog file where it is defined.

In section 3.5, we described the COND_OP element that is used to represent a choice; when the switching condition is expressed by means of a PROLOG_QUERY element the semantic of the COND_OP element is to check if exists a solution to the Prolog query specified.

To solve our problem we will use this feature of the COND_OP element, querying our expert system to know if the rules extracted in the previous step are interesting; if there is a solution, and so the rules are interesting, we will check with the PRESERVE_RULE operator wich of them survive upon a fixed product hierarchy, otherwise we will return the "non interesting" rule set.

In this case the two set of rules needed by the PRESERVE_RULE element are specified by means of a FILE_RULE element, to specify the previous extracted set of rules, and a MINE_RULE element, to specify the set of rules relative to the generalized transactions.

The overall query can be specified using the graphic user interface as showed



KDDML : a markup language for KDD

Query

- CONG_OP
  - [illegible]
  - PRESERVED_RULES
    - FILE_RULES [illegible]
    - MINE_RULE
      - Algorithm APRIORI
        - MinimumSupport 0.7
        - Number of rules 100
        - MinimumConfidence 0.7
      - GFHFT-ALCF
        - FILL_ART [illegible]
    - HIERARCHY PROLOG [illegible]
  - FILE_RULES [illegible]

This file [illegible] containing the XML document representing the query has been added to the Query category of the Repository.

ID. Rules	Preserved Rules				
	Body	-->	Head	Supp.	Conf.
[illegible]	pasta_di_semola cioccolato bevande_e_infusi	-->	spumanti	[illegible]	0.9874
regola14	pasta_di_semolFCO_cocco cioccolatini acque_minerali	-->	vini_spumanti_colci	[illegible]	[illegible]
[illegible]	pasta_di_semola cioccolato bevande_e_infusi prodotti_di_natale	-->	spumanti	[illegible]	0.9874
regola16	pasta_di_semolFCO_cocco cioccolatini acque_minerali pandoro_e_natalizie	-->	vini_spumanti_colci	[illegible]	[illegible]
regola85	pasta_di_semola cioccolato prodotti_di_natale	-->	spumanti	0.7528	0.9874
regola18	pasta_di_semolFCO_cocco cioccolatini pandoro_e_natalizie	-->	vini_spumanti_colci	0.728	0.9892
regola8	cioccolato bevande_e_infusi spumanti	-->	prodotti_di_natale	0.7528	1

Figure 17: Step 2: Query results

in fig. 16, and the results obtained are presented in fig. 17.

5 Conclusions

In this paper we have presented the design and the characteristics of an environment for supporting part of the process of Knowledge Discovery in Databases. More specifically, we have designed a query language that allows us to specify sophisticated combinations of data mining steps. One of the design requirements was flexibility and openness. We tried to satisfy the requirements by approaching the design of the support for a KDD process as the design of a query language on one side, and by using the standard XML for representing intermediate results. We are confident that the above choices will allow us to extend the environment to include support for the other steps of the KDD process.

One of our next objectives is to extend the query language with the possibility of exploiting background domain knowledge in order to drive the data mining steps. Rather than making our query language a full fledged knowledge representation language we are planning to integrate a knowledge representation language, like for example Prolog, as an external component. The interaction between our query executor and the knowledge representation language can be organized around the idea of defining a standard XML representation for goals and answers and the construction of appropriate wrappers.

References

[HF 95] J. Han and Y. Fu. Discovery of Multi-Level Association Rules from Large Databases. *Proceeding of The 21th International Conference on Very Large Databases*, pages 420-431, September 1995.

[IM 96] Imielinski, T.; and Mannila.H. 1996. A Database Perspective of Knowledge Discovery. In *Communications of the ACM*, 39(11):58-64.

[LotXSL] IBM's LOTUS XSL
http://www.alphaworks.ibm.com/formula/xsl

[UCI] UCI KDD Archive
http://kdd.ics.uci.edu/

[XML4J] IBM's Alphaworks XML4J
http://www.alphaworks.ibm.com/formula/xml

[WEKA] WEKA: Waikato Environment for Knowledge Analysis
http://www.cs.waikato.ac.nz/ml/weka

[W3CDOM] Document Object Model specification
http://www.w3c.org/DOM.

[W3CXML] eXtensible Markup Language specification
http://www.w3c.org/XML.

[W3CXSL] eXtensible StyleSheet Language specification
http://www.w3c.org/XSL.