STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LI**, Number 1, 2006

A PROGRAMMING INTERFACE FOR MEDICAL DIAGNOSIS PREDICTION

GABRIELA ŞERBAN, ISTVAN-GERGELY CZIBULA, AND ALINA CÂMPAN

Abstract. The aim of this paper is to present a programming interface that can be used for assisting physicians in medical diagnosis. The interface provides an original diagnosis technique based on relational association rules and a supervised learning method. Using the designed interface, we made an experiment for cancer diagnosis; the precision of the diagnosis on our testing data was 90%. The main advantage of the proposed interface is that can be use in diagnosis for every disease, and, much more, can be simply extended, by adding new symptom types and new relations between symptoms for the given disease.

Keywords: Relational association rules, Programming, Interface, Supervised Learning.

1. Introduction

The purpose of this paper is to propose a technique for assisting medical diagnosis using relational association rules and to present a programming interface for medical diagnosis, using the proposed technique. The interface is meant to facilitate the development of software for identifying the probability of illness in a certain disease.

Much more, this interface can be simply extended by adding new symptom types for the given disease, and by defining new relations between these symptoms.

In our model, we have a set of patients identified by a set of symptoms of a certain disease. The symptoms types and values are unimportant in our approach, the user of the interface can simply define concrete symptoms for the current diagnosis task.

For this issue, the patients, their symptoms types, and the relations between their symptoms can be designed and implemented separately and then interconnected relatively easily in a standard, uniform fashion.

Received by the editors: March, 1, 2006.

2000 *Mathematics Subject Classification.* 68P15, 68T05, 68N19.

1998 *CR Categories and Descriptors.* H.2.8[**Computing Methodologies**]: Database Applications – *Data Mining*; I.2.6[**Computing Methodologies**]: Artificial Intelligence – *Learning*; D.1.5[**Software**]: Programming Techniques – *Object-Oriented Programming*;

2. Relational Association Rules

We extend the definition of ordinal association rules ([3]) towards *relational association rules*.

Definition 1. *Let $R = \{r_1, r_2, \ldots, r_n\}$ be a set of entities (records in the relational model), where each record is a set of m attributes, $(a_1, \ldots, a_m)$. We denote by $\Phi(r_j, a_i)$ the value of attribute a_i for the entity r_j. Each attribute a_i takes values from a domain D_i, which contains ε (empty value, null). Between two domains D_i and D_j can be defined partial relations, such as: less or equal ($\leq$), equal ($=$), greater or equal ($\geq$), etc. We denote by $\mathcal{M}$ the set of all relations defined. An expression $(a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}) \Rightarrow (a_{i_1}\ \mu_1\ a_{i_2}\ \mu_2\ a_{i_3} \ldots \mu_{\ell-1}\ a_{i_\ell})$, where $\{a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}\} \subseteq \mathcal{A} = \{a_1, \ldots, a_m\}$, $a_{i_j} \neq a_{i_k}$, $j, k = 1..\ell$, $j \neq k$ and $\mu_i \in \mathcal{M}$ is a relation over $D_{i_j} \times D_{i_{j+1}}$, is an **relational association rule** if:*

a) *$a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ occur together (are non-empty) in s% of the n records ; we call s the **support** of the rule,*
and
b) *we denote by $R' \subseteq R$ the set of records where $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ occur together and $\Phi(r_j, a_{i_1})\ \mu_1\ \Phi(r_j, a_{i_2})\ \mu_2\ \Phi(r_j, a_{i_3})\ \ldots\ \mu_{\ell-1}\ \Phi(r_j, a_{i_\ell})$ is true for each record r_j din R'; then we call $c = |R'|/|R|$ the **confidence** of the rule.*

The users usually need to uncover interesting relational association rules that hold in a data set; they are interested in relational rules which hold between a minimum number of records, that is rules with support at least s_{min}, and confidence at least c_{min} (s_{min} and c_{min} are user-provided thresholds).

Definition 2. *We call a relational association rule in R* interesting *if its support s is greater than or equal to a user-specified minimum support, s_{min}, and its confidence c is greater than or equal to a user-specified minimum confidence, c_{min}.*

In [3] is given a discovery algorithm for binary ordinal association rules (rules between two attributes). We developed in [4] an algorithm, called *DOAR* (Discovery of Ordinal Association Rules), that efficiently finds all ordinal association rules, of any length, that hold over a data set. We have proved that the proposed algorithm is correct and complete. This algorithm can be used for finding relational association rules, as well. For implementing the main functionality of our interface we have used the *DOAR* algorithm.

3. Medical Diagnosis using Relational Association Rules

In this section we propose a supervised learning technique, based on finding relational association rules, called *MDRAR* (Medical Diagnosis using Relational

Association Rules). *MDRAR* determines the probability that a patient characterized by a set of symptoms suffers from a certain disease. The method works as follows.

Let us consider that our focus is a certain disease, denoted by D. For an appropriate diagnosis in the disease D, we consider a set of n relevant symptoms, S. Each symptom is an attribute a_i and has values from a domain D_i. In order to predict the probable diagnosis we perform a training step using two knowledge bases: a knowledge base containing the patients that suffer from the disease D (the ill patients), and a knowledge base containing the healthy patients. In both knowledge bases, each patient is characterized by a vector with components the values of all the symptoms from S. The knowledge bases of ill and healthy patients are the training data and are used in the training step of the algorithm (see Figure 1).

Let us assume that a n-dimensional vector describing the symptoms from S observed at a patient P is given as input to our algorithm. *MDRAR* determines the probability that the patient P suffers from the disease D, using the model learned in the training step. This is the prediction step of the algorithm (see Figure 1).

The main steps of our diagnosis technique are given in Figure 1:

FIGURE 1. The *MDRAR* technique.

(1) **TRAINING STEP**:
   (a) determine from the knowledge base with ill patients the set of association rules (R_1) having a minimum *support* and $confidence$;
   (b) determine from the knowledge base with healthy patients the set of association rules (R_2) having a minimum *support* and $confidence$;
(2) **PREDICTION STEP**: for each patient P for which we intend to predict the diagnosis, calculate the probability that P suffers from the disease D as the percentage of rules from R_1 verified by P and rules from R_2 not verified by P.

4. The programming interface

In this section we propose an API that allows a simple development of applications for medical diagnosis based on finding relational association rules. The API provides an uniform development for all these applications.

The main advantage of the interface is that the user can simply define, depending on the current disease, new types of symptoms and new types of relations between the symptoms, and the diagnosis prediction process remains unchanged. The interface is realized in JDK 1.5, and is meant to facilitate software development for assisting medical diagnosis.

There are seven basic entities (classes): `Patient` (defines a patient), `Symptom` (characterizes the patients), `SymptomType` (represents the type of a symptom), `Relation` (defines a partial relation between symptoms), `AssociationRule` (describes relational association rules between symptoms), `AssociationRuleGenerator` (responsible with generating relational association rules from the set of patients, based on their symptoms values and relations already defined) and `Diagnosis` (responsible with predicting the diagnosis for a given patient, using the *MDRAR* algorithm).

For designing the interface, we made an abstraction of the mechanisms for generating relational association rules, in order the interface to be useful for any kind of disease, symptoms and relations between symptoms. Much more, the patient entities are completely separated from the symptoms that characterize them (a `Patient` has to know nothing about its `Symptoms`, it has to know only about their behavior). Thus, we can easily change and add `Symptoms` characterizing the `Patients`, and `Relations` between `Symptoms`, without affecting the general diagnosis prediction process.

The `AssociationRuleGenerator` class is the main class of the interface and manages the process of finding relational association rules in the given set of `Patients` with respect to the given `Symptoms`. This class provides an operation that finds relational association rules in data, by implementing the *DOAR* algorithm (section 2).

The interface also provides:

- the class `Patients` that models a set of patients (the data set from which we want to extract relational association rules);
- the class `AssociationRules` that models a set of relational association rules;
- the class (`Relations`) that manages the set of relations between the symptoms (this class allows to manage dynamically the set of relations defined between symptoms).

For using the interface in a specific diagnosis prediction task, the user has only to:

- define specialized classes for the concrete symptom types (for example a class `SymptomInt` that extends the abstract class `SymptomType` if the symptoms are quantified by integer values);
- define specialized classes for the concrete relations between symptoms (for example a class `IntIntEqual` that extends the abstract class `Relation` if we want to describe the equality relation between two integer valued symptoms);
- construct the concrete set of patients.

All other mechanisms needed for generating the rules and predicting the diagnosis are provided by the classes from the interface.

In the following we present the skeleton of a diagnosis application. Let us assume that symptoms have integer values, and the only relation needed between Symptoms is "=".

- First, the user implements the class that defines the concrete symptom type.
  public class `SymptomInt` **extends** `Symptom`{...}.

- Second, the user implements the class that defines the concrete relation between the symptoms already defined.
  public class `IntIntEqual` **extends** `Relation`{...}.

In the same manner as above, the user can define as many symptom types and relations as are needed in the current diagnosis prediction task.

In the application class the user has to define a method that reads the data (patients) from an external device (file, database) and returns a set of patients (an instance of the `Patients` class) and to add (register) the concrete relations defined above to the set of relations `Relations`.

```
public class Application {
 public Application(){
  // The manager of relations adds a new concrete relation to its set of
  // relations
  RelationSet.addRelation(new IntIntEqual());
  // The application provides a method that constructs the set of ill patients
  Patients ill = readData();
  // An instance of an object AssociationRuleGenerator is created from the
  // Patient set created above
  AssociationRuleGenerator arg = new AssociationRuleGenerator(ill);
  // The association rule generator generates the set of association rules
  // having a minimum support and confidence
  double minimumSupport = 0.9;
  double minimumConfidence = 0.65;
  AssociationRuleSet illRules = arj.genAssociationRules(minimumConfidence);
  // The application provides a method that constructs the set of healthy
  // patients
  Patients healthy = readData();
  // An instance of an object AssociationRuleGenerator is created from the
  // Patient set created above
  arg = new AssociationRuleGenerator(healthy);
  // The association rule generator generates the set of association rules
  // having a minimum support and confidence
  AssociationRuleSet healthyRules=arj.genAssociationRules(minimumConfidence,
                                  minimumSupport);
  // An instance of a Diagnosis class is now created
```

```
  Diagnosis d = new Diagnosis(healthyRulles, illRules);
  // Now the object d can be used to predict the diagnosis a given patient
 }
}
```

Figure 2 shows a simplified UML diagram ([6]) of the interface, illustrating the hierarchy of classes. The figure illustrates the core of the interface and what is outside the core are the concrete classes that the user has to define, by extending the classes provided by the interface, in order to develop a diagnosis prediction application for a given disease. It is important to mention that all the core classes provided by the interface remain unchanged in all applications.

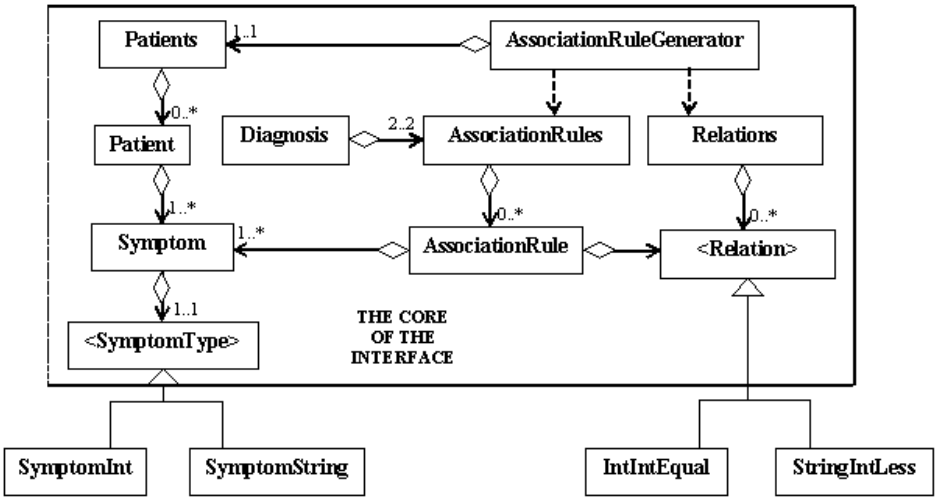

FIGURE 2. The diagram of the programming interface

5. The Design of the Interface

The classes used for realizing the interface are the following:

- `SymptomType` is `ABSTRACT`.

  Models an abstract symptom type, identified by the symptom type name. The class has operations for returning and modifying the symptom type name, a method for verifying the equality of two symptom types and an abstract method that creates a symptom value from a string. The concrete symptom types defined by the user of the interface, will extend the abstract SymptomType, managing their concrete type and overwriting the abstract methods from the abstract `SymptomType`;

- `Symptom`.

  Models a symptom characterizing the patients, and is characterized by a name, a symptom type (an instance of the `SymptomType` class) and a value (that is an object).
- `Patient`.

  Models a patient from the data set, which consists in a list of symptoms. The class has operations for managing the symptoms: adding, removing and returning symptoms from a given position, searching a symptom with a given name and symptom type.
- `Patients`.

  Models a set of patients, which consists in a list of *Patient* objects. The class has operations for managing the set of patients: adding, removing, searching patients and a method that returns an iterator on the set.
- `Relation` is `ABSTRACT`.

  Models an abstract relation between two symptom types $Type_1$ and $Type_2$. The class has abstract operations for: returning $Type_1$ and $Type_2$, returning the name of the relation, verifying if two `Symptom`s are in the given relation and for returning the converse of the relation.
- `AssociationRule`.

  Models a relational association rule, which consists in a set of abstract symptoms, a set of abstract relations, and characterized by its support and confidence. The main methods of this class are for: managing the symptoms and the relations from the association rule, setting and returning the support and the confidence of the rule.
- `AssociationRuleSet`.

  Models the structure of a set of relational association rules, which consists in a list of `AssociationRule` objects. The class has operations for managing the set of relational association rules: adding, removing, searching rules and a method that returns an iterator on the set.
- `Relations`.

  In our design this class models a repository (set) of relations, that allows the user to dynamically add relations between newly defined Symptom types. The user can dynamically add new defined relations in this list, using a method *addRelation*. This class has methods for obtaining the relations for a given `Symptom`, for verifying if there exists a given `Relation` between two `Symptom`s.
- `AssociationRuleGenerator`.

  Is the class that implements the process of finding relational association rules in a set of patients. The main method of the class is *generateAssociationRules*, that generates from the data set the relational

association rules having a minimum given support and confidence, and returns an instance of the `AssociationRuleSet` class.

- `Diagnosis`.

  Is the main class of the interface, that predicts a diagnosis for a given patient, based on the technique described in section 3. It represents the *heart* of the interface, the uniform usage that all patients, with their particular symptoms and relations, are meant to conform to. An instance of the `Diagnosis` class is associated with two instances of the `AssociationRuleSet` class.

As it can be seen on Figure 2, there is a dependency relationship between the `AssociationRuleGenerator` and `Relations`, that allows the association rule generator to dynamically manage the relations added by the user, without affecting the main process of detecting rules.

6. Experimental Evaluation

The file for this experiment was obtained from the website at "http://www.cormactech.com/neunet".

In order to test the above defined interface, we considered a HealthCare experiment for predicting the cancer disease.

The entities in this experiment are patients: each patient is identified by 9 Symptoms [1]. Each Symptom represents the value of a symptom in the cancer disease, and has integer values between 1 and 10. Each instance has one of 2 possible classes: benign or malignant. In this experiment are 457 patients (entities).

The attribute information used in the "cancer" experiment is shown in Table 1.

Table 1. Attribute information in the "cancer" experiment

	Attribute	Domain
1.	Clump Thickness	1 - 10
2.	Uniformity of Cell Size	1 - 10
3.	Uniformity of Cell Shape	1 - 10
4.	Marginal Adhesion	1 - 10
5.	Single Epithelial Cell Size	1 - 10
6.	Bare Nuclei	1 - 10
7.	Bland Chromatin	1 - 10
8.	Normal Nucleoli	1 - 10
9.	Mitoses	1 - 10

For this experiment, we have defined:

- `SymptomInt`, defining the integer Symptom representing a patient's symptom in the cancer disease;

- `IntIntEqual`, `IntIntLess` and `IntIntGreater`, defining the possible relations between two integer symptoms ($=$, $\leq$, $\geq$);
- a mechanism that reads the data (for each patient, it reads the values of the symptoms) and creates a `Patients` object.

We executed the medical diagnosis algorithm with minimum support threshold of 0.9 and minimum confidence threshold of 0.65. The training data consists in 147 ill patients and 260 healthy patients. The prediction step was made for 60 patients (30 ill patients and 30 healthy patients). We have obtained a precision of 90%.

As a conclusion of our experiments, we have to mention, from a programmer point of view, the advantages of using the interface proposed in this paper:

- is very simple to use;
- the effort for developing a diagnosis application based on relational association rule detection is reduced - we need to define only a few classes, the rest is provided by the interface;
- the user of the interface has to know nothing about the method of finding relational association rules or about the prediction method, because they are provided by the interface;
- we can add new symptom types and relations between symptoms, while the interface remains unchanged.

7. Conclusions and Further Work

As a conclusion, we have developed a small framework that will help programmers to build, dynamically, their own applications for diagnosis prediction in different kind of diseases, without dealing with the internal mechanism (that remains unchanged and is provided by the interface) and having the possibility to define their own types of symptoms and relations between symptoms. So, the programmer's effort for developing an application is small.

Further work can be done in the following directions:

- to test the accuracy of our technique on practical diagnosis. We think that increasing the number of data in the training step the accuracy of the prediction will grow;
- to study, for certain diseases, how can other symptoms and other relations between symptoms be added in order to assure better results in diagnosis.

References

[1] Wolberg, W., Mangasarian, O.L.: "Multisurface method of pattern separation for medical diagnosis applied to breast cytology", Proceedings of the National Academy of Sciences, U.S.A., Volume 87, December 1990, pp 9193–9196.

[2] http://www.cormactech.com/neunet, "Discover the Patterns in Your Data", CorMac Technologies Inc, Canada.

[3] Marcus, A., Maletic, J. I., Lin, K.-I., "Ordinal Association Rules for Error Identification in Data Sets", CIKM 2001, 2001, pp. 589–591.
[4] Campan, A., Serban, G., Truta, T. M., Marcus, A., "An Algorithm for the Discovery of Arbitrary Length Ordinal Association Rules", submitted to DMIN'06.
[5] Han, J., Kamber, M., "Data Mining: Concepts and Techniques", The Morgan Kaufmann Series in Data Management Systems, 2001.
[6] http://www.omg.org/technology/documents/formal/uml.htm.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: gabis@cs.ubbcluj.ro

"InfoWorld", Cluj-Napoca
E-mail address: czibula.istvan@infoworld.ro

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania
E-mail address: alina@cs.ubbcluj.ro