

Fig. 1 Architecture of a knowledge-mining system

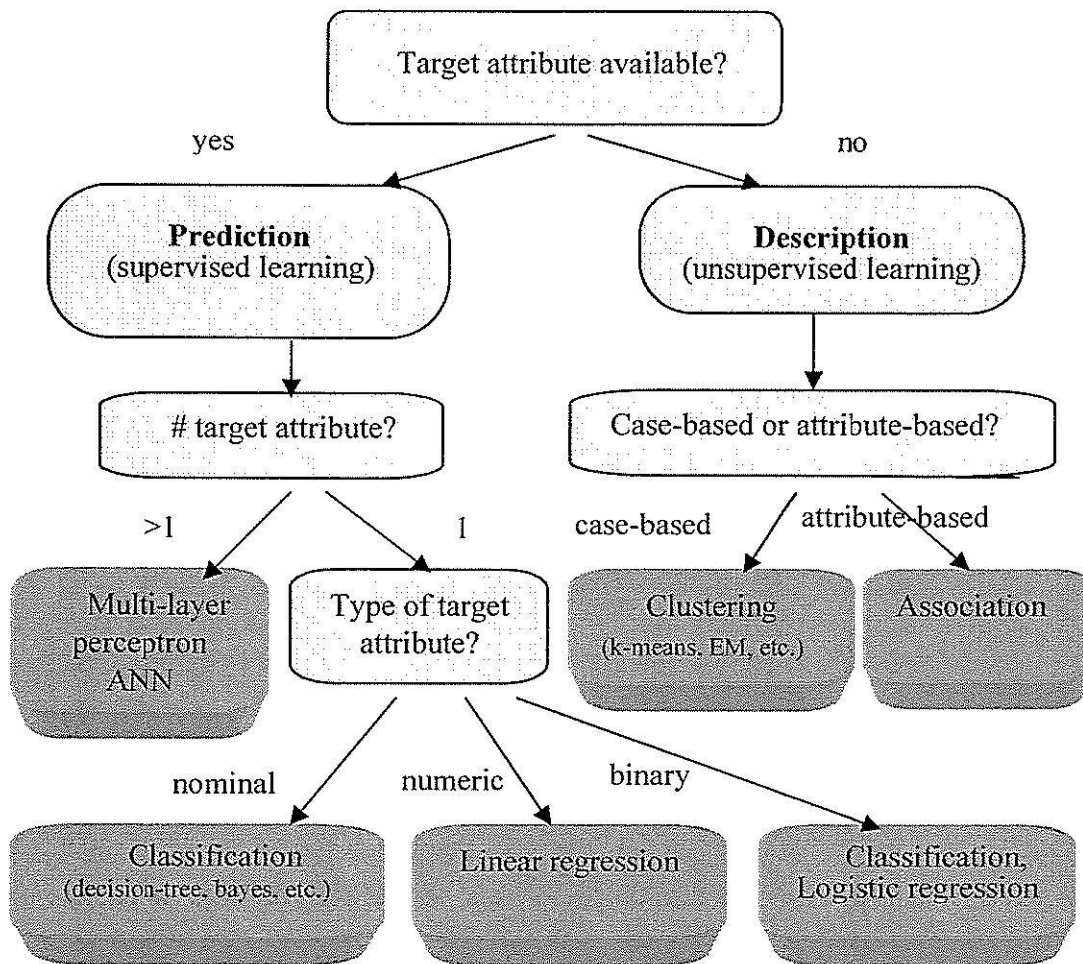


Fig. 2 Ontology for guiding mining method selection at the DM step

4 Implementation and Running Example

The SUT-Miner system has been implemented based on the concept of higher-order logic which is an extension of first-order logic to be more expressive and more powerful. First-order logic has been extensively used in intelligent systems [22] as an inference mechanism to deduce new facts. A classic example is that given a general rule $\forall x \text{ Man}(x) \Rightarrow \text{Mortal}(x)$ and a known fact $\text{Man}(\text{Socrates})$, we can deduce new fact that $\text{Mortal}(\text{Socrates})$.

Despite its successful application to many computational problems such as natural language processing, first-order logic poses a restriction on the type of variables appearing in quantifications to exclude predicates. Higher-order logic [12], on the other hand, allows variables to quantify over predicates. With such relaxation, higher-order logic facilitates the implementation of a knowledge-intensive system that takes other knowledge as its input in a closed form. An exemplar in figure 3 demonstrates a higher-order logic-based implementation of frequent pattern mining problem. The coding is based on the syntax of SWI prolog (www.swi-prolog.org).

```

frequent_pattern_mining:-
  min_support(V),                % set minimum support
  makeC1(C),                      % create candidate 1-itemset
  makeL(C,L),                    % compute large itemset
  apriori_loop(L,1).             % recursively run apriori

apriori_loop(L,N):- length(L) is 1,!. % base case of recursion
apriori_loop(L,N):-              % inductive step
  N1 is N+1,
  makeC(N1,L,C),  makeL(C,Res),
  apriori_loop(Res,N1).

makeC1(Ans):-
  input(D),                      % input data as a list, e.g. [[a], [a,b]]
  allComb(1,ItemList,Ans2),      % make combination of itemset
  maplist(countSS(D),Ans2,Ans).  % scan database
                                % countSS is predicate passing as argument
                                % to a higher-order predicate maplist

makeC(N,ItemSet,Ans):-
  input(D),
  allComb(2,ItemSet,Ans1),
  maplist(flatten,Ans1,Ans2),
  maplist(list_to_ord_set,Ans2,Ans3),
  list_to_set(Ans3,Ans4),
  include(len(N),Ans4,Ans5),    % include is also a higher-order predicate
  maplist(countSS(D),Ans5,Ans). % scan database find: List+N

makeL(C,Res):-                   %for all L creation
  include(filter,C,Ans),         % call higher predicates include
  maplist(head,Ans,Res).        % and maplist

filter(_+N):- input(D),length(D,I),min_support(V), N>=(V/100)*I.
head(H+_,H).                    % filter and head are for pattern matching

% an arbitrary subset of the set containing given number of elements.
comb(0,_,[]).
comb(N,[X|T],[X|Comb]):-N>0,N1 is N-1,comb(N1,T,Comb).
comb(N,[_|T],Comb):-N>0,comb(N,T,Comb).

allComb(N,I,Ans):-
  setof(L,comb(N,I,L),Ans).    % setof is a second-order predicate

countSubset(A,[],0).
countSubset(A,[B|X],N):-not(subset(A,B)),countSubset(A,X,N).
countSubset(A,[B|X],N):-subset(A,B),countSubset(A,X,N1),N is N1+1.

countSS(SL,S,S+N):-countSubset(S,SL,N).
len(N,X):-length(X,N).

```

Fig. 3 Frequent-pattern mining implemented with a higher-order logic approach

[tumor-size=20-24, inv-nodes=0-2] => [irradiate=no]	conf:(1)
[age=50-59, inv-nodes=0-2, node-caps=no] => [irradiate=no]	conf:(1)
[age=50-59, menopause=ge40, node-caps=no] => [irradiate=no]	conf:(1)
[menopause=ge40, inv-nodes=0-2, deg-malig=1] => [irradiate=no]	conf:(1)
...	
[node-caps=yes, breast=left, recurrence=yes] => [irradiate=yes]	conf:(0.63)
[breast-quad=left_low, recurrence=yes] => [irradiate=yes]	conf:(0.5)
[menopause=premeno, node-caps=yes] => [irradiate=yes]	conf:(0.5)
[node-caps=yes, breast=left] => [irradiate=yes]	conf:(0.5)
...	
[tumor-size=10-14, node-caps=no] => [recurrence=no]	conf:(1)
[tumor-size=10-14, irradiate=no] => [recurrence=no]	conf:(1)
[tumor-size=10-14, inv-nodes=0-2] => [recurrence=no]	conf:(1)
...	
[node-caps=yes, breast=left, irradiate=yes] => [recurrence=yes]	conf:(0.91)
[menopause=premeno, node-caps=yes, irradiate=no] => [recurrence=yes]	conf:(0.83)
[menopause=premeno, node-caps=yes, breast=left] => [recurrence=yes]	conf:(0.83)
[menopause=premeno, node-caps=yes, deg-malig=3] => [recurrence=yes]	conf:(0.79)
[breast=left, breast-quad=left_low, irradiate=yes] => [recurrence=yes]	conf:(0.77)

Fig. 4 Some part of a breast-cancer frequent pattern mining result

Frequent pattern mining is the discovery of relationships or correlations between items in a database. Let $I = \{i_1, i_2, i_3, \dots, i_m\}$ be a set of m items and $DB = \{C_1, C_2, C_3, \dots, C_n\}$ be a database of n cases or observations and each case contains items in I . A *pattern* is a set of items that occur in a case. The number of items in a pattern is called the length of the pattern. To search for all valid patterns of length 1 up to m in large database is computational expensive. For a set I of m different items, the search space for all distinct patterns can be as huge as $2^m - 1$. To reduce the size of the search space, the *support* measurement has been introduced [1]. The function $support(P)$ of a pattern P is defined as a number of cases in DB containing P . Thus, $support(P) = |\{T \mid T \in DB, P \subseteq T\}|$. A pattern P is called *frequent pattern* if the support value of P is not less than a predefined minimum support threshold $minS$. It is the $minS$ constraints that help reducing the computational complexity of frequent pattern generation. The $minS$ metric has an anti-monotone property and is applied as a basis for reducing search space of mining frequent patterns in algorithm Apriori [1].

We have tested our implementation with the breast cancer dataset taken from the UCI repository (<http://www.ics.uci.edu/~mllearn/MLRepository.html>). This medical data set is a collection of 191 observations on follow-up patients examining on a recurrence of breast cancer. Each patient's record contains ten attributes: age, menopause, tumor-size, inv-nodes, node-caps, deg-malignant, breast (left, right), breast-quad, irradiate (yes, no), recurrence (yes, no). We run frequent-pattern mining with minimum support 0.001 and show some of the discovered association rules in figure 4. Each rule is attached with the confidence value as a metric to evaluate accuracy of the induced rule. For the rule $A \Rightarrow B$, confidence is

computed from proportion of $support(A \& B)$ to $support(A)$. The confidence value 1.0 thus implies a 100% accurate association rule.

The mining results shown in figure 4 are in the form of implication. Rule interpretation is straightforward. Taking the last rule in the figure as an example, it can be interpreted as "a patient who has cancer at her left breast in left lower quadrant position and had been treated with radiation implies that she will have a 77-percent chance of cancer recurrence." During the process of searching for frequent patterns, the support values of frequently occurred patterns can also be out put to quantify supporting evidence of the induced association rules.

5 Conclusions and Discussion

We have proposed the design of SUT-Miner, a knowledge-mining system. The system is intended to support knowledge acquisition in medical data and other domains that require new knowledge to support better decisions. The proposed system is an environment for knowledge discovery composing of tools and methods suitable for various kinds of mining such as data classification, regression, clustering, association mining. The intelligence of the system is supported by ontology technology to provide semantics for mining technique selection as well as for knowledge integration at a post-data mining step.

The implementation of the proposed system has been done based on the concept of higher-order logic which is an extension of the well-known first-order logic. With the expressive power of higher-order logic, program coding of the designed system is very concise as demonstrated in the paper. Program conciseness directly contributes to program verification and validation which are important issues in software engineering. The closed form of higher-order logic also supports constraint mining and higher-order mining, i.e. mining from previously discovered knowledge. Our future research is to extend the design of SUT-Miner to facilitate constraint and higher-order mining. Implementing a mechanism for knowledge inferring is also one of our research plans.

Acknowledgments. This work has been fully supported by research fund from Suranaree University of Technology (SUT) granted to the Data Engineering and Knowledge Discovery research unit. This research is also partly supported by grants from the National Research Council of Thailand (NRCT) and the Thailand Research Fund (TRF) under grant number RMU 5080026.

References

- [1] Agrawal, R., Srikant, R.: Fast algorithm for mining association rules. In: Proc. VLDB, pp. 487–499 (1994)
- [2] Alavi, M., Leidner, D.E.: Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS Quarterly* 25(1), 107–136 (2001)
- [3] Bojarczuk, C.C., Lopes, H.S., Freitas, A.A., et al.: A constrained-syntax genetic programming system for discovering classification rules: Application to medical data sets. *Artificial Intelligence in Medicine* 30, 27–48 (2004)

- [4] Bratsas, C., Koutkias, V., Kaimakamis, E., et al.: KnowBaSIGs-M: An ontology-based system for semantic management of medical problems and computerised algorithmic solutions. *Computer Methods and Programs in Biomedicine* 83, 39–51 (2007)
- [5] Correia, R., Kon, F., Kon, R.: Borboleta: A mobile telehealth system for primary homecare. In: *Proc. ACM Symposium on Applied Computing*, pp. 1343–1347 (2008)
- [6] De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: *Proc. KDD*, pp. 204–212 (2008)
- [7] Ghazavi, S., Liao, T.W.: Medical data mining by fuzzy modeling with selected features. *Artificial Intelligence in Medicine* 43(3), 195–206 (2008)
- [8] Hristovski, D., Peterlin, B., Mitchell, J.A., et al.: Using literature-based discovery to identify disease candidate genes. *Int. J. Medical Informatics* 74, 289–298 (2005)
- [9] Huang, M.J., Chen, M.Y., Lee, S.C.: Integrating data mining with case-based reasoning for chronic diseases prognosis and diagnosis. *Expert Systems with Applications* 32, 856–867 (2007)
- [10] Hulse, N.C., Fiol, G.D., Bradshaw, R.L., et al.: Towards an on-demand peer feedback system for a clinical knowledge base: A case study with order sets. *J. Biomedical Informatics* 41, 152–164 (2008)
- [11] Kakabadse, N.K., Kouzmin, A., Kakabadse, A.: From tacit knowledge to knowledge management: Leveraging invisible assets. *Knowledge and Process Management* 8(3), 137–154 (2001)
- [12] Nadathur, G., Miller, D.: Higher-order Horn clauses. *J. ACM* 37, 777–814 (1990)
- [13] Nguyen, D., Ho, T., Kawasaki, S.: Knowledge visualization in hepatitis study. In: *Proc. Asia-Pacific Symposium on Information Visualization*, pp. 59–62 (2006)
- [14] Palaniappan, S., Ling, C.S.: Clinical decision support using OLAP with data mining. *Int. J. Computer Science and Network Security* 8(9), 290–296 (2008)
- [15] Roddick, J.F., Fule, P., Graco, W.J.: Exploratory medical knowledge discovery: experiences and issues. *ACM SIGKDD Explorations Newsletter* 5(1), 94–99 (2003)
- [16] Roddick, J.F., Spiliopoulou, M., Lister, D., et al.: Higher order mining. *ACM SIGKDD Explorations Newsletter* 10(1), 5–17 (2008)
- [17] Ruppel, C.P., Harrington, S.J.: Sharing knowledge through intranets: A study of organizational culture and intranet implementation. *IEEE Transactions on Professional Communication* 44(1), 37–51 (2001)
- [18] Sahama, T.R., Croll, P.R.: A data warehouse architecture for clinical data warehousing. In: *Proc. 12th Australasian Symposium on ACSW Frontiers*, pp. 227–232 (2007)
- [19] Satyadas, A., Harigopal, U., Cassaigne, N.P.: Knowledge management tutorial: An editorial overview. *IEEE Transactions on Systems, Man and Cybernetics, Part C* 31(4), 429–437 (2001)
- [20] Shillabeer, A., Roddick, J.F.: Establishing a lineage for medical knowledge discovery. In: *Proc. 6th Australasian Conf. on Data Mining and Analytics*, pp. 29–37 (2007)
- [21] Thongkam, J., Xu, G., Zhang, Y., et al.: Breast cancer survivability via AdaBoost algorithms. In: *Proc. 2nd Australasian Workshop on Health Data and Knowledge Management*, pp. 55–64 (2008)
- [22] Truemper, K.: *Design of logic-based intelligent systems*. John Wiley & Sons, New Jersey (2004)
- [23] Uramoto, N., Matsuzawa, H., Nagano, T., et al.: A text-mining system for knowledge discovery from biomedical documents. *IBM Systems J.* 43(3), 516–533 (2004)
- [24] Zhou, X., Liu, B., Wu, Z.: Text mining for clinical Chinese herbal medical knowledge discovery. In: *Discovery Science 8th Int. Conf.*, pp. 396–398 (2005)
- [25] Zhuang, Z.Y., Churilov, L., Burstein, F.: Combining data mining and case-based reasoning for intelligent decision support for pathology ordering by general practitioners. *European J. Operational Research* 195(3), 662–675 (2009) doi: 10.1016/j.ejor.2007.11.003

Knowledge Induction from Medical Databases with Higher-Order Programming

Nittaya Kerdprasop and Kittisak Kerdprasop
Data Engineering and Knowledge Discovery (DEKD) Research Unit
School of Computer Engineering, Suranaree University of Technology
111 University Avenue, Nakhon Ratchasima 30000
THAILAND
nittaya@sut.ac.th, kittisakThailand@gmail.com

Abstract: - Medical data mining is an emerging area of computational intelligence applied to automatically analyze patients' records aiming at the discovery of new knowledge potentially useful for medical decision making. Induced knowledge is anticipated not only to increase accurate diagnosis and successful disease treatment, but also to enhance safety by reducing medication-related errors. Modern healthcare organizations regularly generate huge amount of electronic data that could be used as a valuable resource for knowledge induction to support decision-making of medical practitioners. Unfortunately, a domain-specific decision support system that provides a suite of customized and flexible tools to efficiently induce knowledge from medical databases with representational heterogeneity does not currently exist. We, thus, design and develop a medical decision support system based on a powerful logic programming framework. The proposed system includes a knowledge induction component to induce knowledge from clinical data repositories and the induced knowledge can also be deployed to pre-treatment data from other sources. The implementation of knowledge induction engine has been presented to express the power of higher-order programming of logic-based language. The flexibility of our mining engine is obtained through the pattern matching and meta-programming facilities provided by logic-based language.

Key-Words: - Medical decision making, Medical informatics, Logic-based knowledge induction, Higher-order programming

1 Introduction

Knowledge is a valuable asset to most organizations as a substantial source to enhance understanding of data relationships and support better decisions to increase organizational competency. Automatic knowledge acquisition can be achieved through the availability of the knowledge induction component. The induced knowledge can facilitate various knowledge-related activities ranging from expert decision support, data exploration and explanation, estimation of future trends, and prediction of future outcomes based on present data.

In this paper, we present the knowledge induction system specifically designed to facilitate knowledge discovery from medical data. Various data mining and machine learning methods had been proposed to learn useful knowledge from medical data [5], [6], [7], [8], [17], [19]. Major techniques adopted by many researchers are rule induction and classification tree generation with the main purpose to support medical diagnosis [3], [9], [13]. Some researchers had even extended the knowledge discovery aspect to the larger scale of medical

decision support system and data warehouse [2], [4], [10], [11], [20].

Our work is also in the main stream of medical decision support system development, but our methodology is different from those appeared in the literature. The system proposed in this paper is based on logic and higher-order programming paradigms. The justification of our logic-based system is that the closed form of Horn clauses that treats program in the same way as data facilitates fusion of knowledge learned from different sources; this situation is a normal setting in medical domain. Knowledge reuse can also easily practice in this framework. We design the system as an integrated environment storing a repertoire of tools for discovering various kinds of knowledge.

The outline of this paper is as follows. Section 2 briefly discusses knowledge induction methods implemented in our system. Section 3 reviews the basics of logic and higher-order programming. Sections 4 and 5 present the conceptual design and implementation, respectively, of our system. Section 6 concludes the paper.

2 Knowledge Induction Methods

This section briefly reviews the three main data mining methods extensively applied to induce knowledge from varieties of data domains. These methods are implemented in our medical decision support system.

2.1 Tree-Based Knowledge Induction

Decision tree induction [18] is a popular method for inducing knowledge from data. Popularity is due to the fact that mining result in a form of decision tree is interpretability, which is more concern among medical practitioners than a sophisticated method but lack of understandability. A decision tree is a hierarchical structure with each node contains decision attribute and node branches corresponding to different attribute values of the decision node. The goal of building decision tree is to partition data with mixing classes down the tree until the leaf nodes contain pure class.

In order to build a decision tree, we need to choose the best attribute that contributes the most towards partitioning data to the purity groups. The metric to measure attribute's ability to partition data into pure class is *Info*, which is the number of bits required to encode a data mixture. To choose the best attribute, we have to calculate information gain, which is the yield we obtained from choosing that attribute. The information gain calculates yield on data set before splitting and after choosing attribute with two or more splits. The gain value of each candidate attribute is calculated. Then choose the maximum one to be the decision node. The process of data partitioning continues until the data subset has the same class label.

2.2 Association Mining

Association mining is the discovery of relationships or correlations between items in a database. Let $I = \{i_1, i_2, i_3, \dots, i_m\}$ be a set of m items and $DB = \{C_1, C_2, C_3, \dots, C_n\}$ be a database of n cases or observations and each case contains items in I . A *pattern* is a set of items that occur in a case. The number of items in a pattern is called the length of the pattern. To search for all valid patterns of length 1 up to m in large database is computational expensive. For a set I of m different items, the search space for all distinct patterns can be as huge as $2^m - 1$. To reduce the size of the search space, the *support* measurement has been introduced [1]. The function $support(P)$ of a pattern P is defined as a number of cases in DB containing P . Thus,

$support(P) = |\{T \mid T \in DB, P \subseteq T\}|$. A pattern P is called *frequent pattern* if the support value of P is not less than a predefined minimum support threshold $minS$. It is the $minS$ constraints that help reducing the computational complexity of frequent pattern generation. The $minS$ metric has an anti-monotone property and is applied as a basis for reducing search space of mining frequent patterns in algorithm Apriori [1].

2.3 Data Clustering

Clustering refers to the iterative process of automatic grouping of data based on their similarity. There exist a large number of clustering techniques, but the most classical and popular one is the k-means algorithm [12]. Given a data set containing n objects, k-means partitions these objects into k groups. Each group is represented by the centroid, or central point, of the cluster. Once cluster means or representatives are selected, data objects are assigned to the nearest centers. The algorithm iteratively selects new better representatives and reassigns data objects until the stable condition has been reached. The stable condition can be observed from cluster assigning that each data object does not change its cluster.

3 Programming Based on Logic

In logic programming, a clause is a disjunction of literals (atomic symbols or their negations) such as $p \vee q$ and $\neg p \vee r$. A statement is in clausal form if it is a conjunction of clauses such as $(p \vee q) \wedge (\neg p \vee r)$. Logic programming is a subset of first order logic in which clauses are restricted to Horn clauses.

A Horn clause, named after the logician Alfred Horn [16], is a clause that contains at most one positive literal such as $\neg p \vee \neg q \vee r$. Horn clauses are widely used in logic programming because their satisfiability property can be solved by resolution algorithm (an inference method for checking whether the formula can be evaluated to true).

A Horn clause with no positive literal, such as $\neg p \vee \neg q$, which is equivalent to $\neg(p \wedge q)$, is called *query* in Prolog and can be interpreted as ' $:- p, q$ ' in which its value (true/false) to be proven by resolution method. A clause that contains exactly one positive literal such as r is called a *fact* representing a true statement, written in clausal form as ' $r :-$ ' in which the condition part is empty and that means r is unconditionally true. Therefore, facts are used to represent data. A Horn clause that contains one positive literal and one or more

negative literals such as $\neg p \vee \neg q \vee r$ is called a *definite clause* and such clause can equivalently written as $(p \wedge q) \rightarrow r$ which in turn can be represented as a Prolog rule as $r :- p, q$. The symbol $' :- '$ is intended to mean $' \leftarrow '$, which is implication in first-order logic (it stands for 'if'), and the symbol $' , '$ represents the operator \wedge (or 'AND').

In Prolog, rules are used to define procedures and a Prolog program is normally composed of facts and rules. Running a Prolog program is nothing more than posing queries to obtain true/false answers. The advantages of using logic programming are the flexible form of query posing and the additional information regarding variable instantiation obtained from the Prolog system once the query is evaluated to be true.

The symbols p, q, r are called *predicates* in first-order logic programming and they can be quantified over variables such as $r(X) :- p(X, Y), q(Y)$. This clause has the same meaning as $\forall X (p(X, Y) \wedge q(Y) \rightarrow r(X))$. The scope of variables is within a clause (delimit the end of clause with a period). Horn clauses are thus the fundamental concept of logic programming.

Higher-order predicate is a predicate in a clause that can quantify over other predicate symbols [14], [15]. As an example, besides the rule $r(X) :- p(X, Y), q(Y)$, if we are also given the following five Horn clauses (or facts): $p(1, 2), p(1, 3), p(5, 4), q(2), q(4)$.

By asking the query: $?- r(X)$, we will get the response as *'true'* and also the first instantiation information as $X=1$. If we want to know all instantiations that make $r(X)$ to be true, we may ask the query: $?- findall(X, r(X), Answer)$. We will get the response: $Answer = [1, 5]$, which is a set of all answers obtained from the predicate $r(X)$ according to the given facts. The predicate symbol *findall* quantifies over the variables $X, Answer$, and the predicate r . The predicate *findall* is thus called a higher-order predicate.

Meta-level programming is also another powerful feature of Prolog. Meta-programs treat other programs as their input data. Data and program in Prolog take the same representational format, i.e. clausal form. Therefore, it is very natural to write meta-program in Prolog.

The following example illustrates the procedure *map* that takes a list of integers [1,2,3,4,5] and another procedure *square* as its input arguments and produce a list of square values as its output. If we pose the query: $?- map(square, [1,2,3,4,5], L)$, then we will get the answer: $L = [1,4,9,16,25]$.

$square(X, Y) :- Y \text{ is } X*X.$

```
map(ProcedureName, [H|T], [NewH|NewT]) :-
    Procedure=.. [ProcedureName,H,NewH],
    call(Procedure),
    map(ProcedureName, T, NewT).

map(_, [], []).
```

4 Medical Decision Support System

Health information is normally distributive and heterogeneous. Hence, we design the medical decision support system (Figure 1) to include data integration component at the top level to collect data from distributed databases and also from documents in text format. Data at this stage are to be stored in a warehouse to support direct querying as well as analysis with knowledge induction engine.

Knowledge base in our design stores both induced knowledge, in which its significance has to be evaluated by the domain expert, and background knowledge encoded from consultation with human experts. Knowledge inferring and reasoning is the module interfacing with medical practitioners and physicians at the front-end and accessing knowledge base at the back-end.

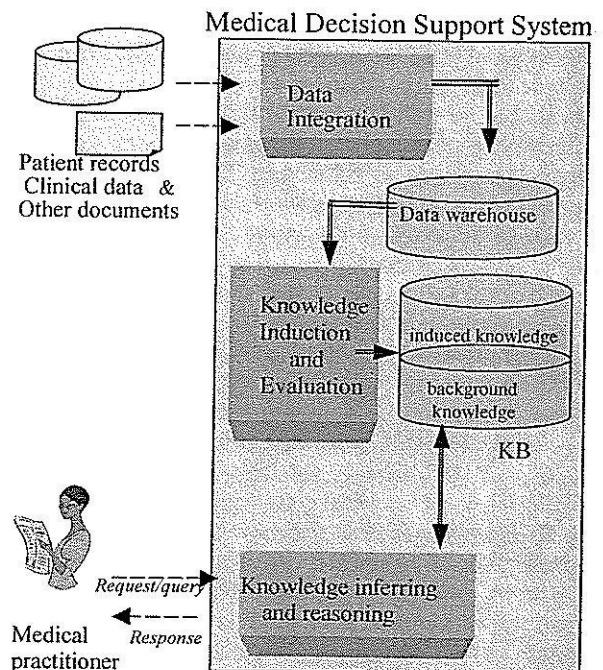


Figure 1. Knowledge induction component in the medical decision support system

The process of knowledge discovery is complex and iterative in its nature. We design the system to be composed of two phases: knowledge induction and knowledge inferring.

Knowledge induction is the back-end of the system responsible for acquiring and discovering new and useful knowledge. Usefulness is to be validated at the final step by human experts. Discovered knowledge is stored in the knowledge base to be applied to solve new cases or create new knowledge in the knowledge inferring phase, which is the front-end of the proposed system.

The proposed system obtains input from heterogeneous data sources. Such data can be redundant, incomplete, and noisy. Therefore, the knowledge-induction-and-evaluation component (Figure 2) has been designed to clean, transform, and select only relevant data sample.

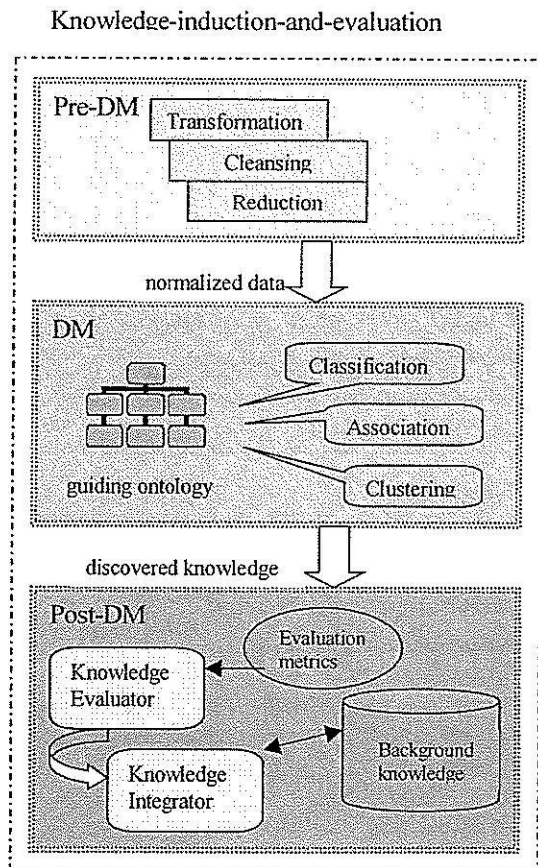


Figure 2. Architecture of the knowledge-induction-and-evaluation component

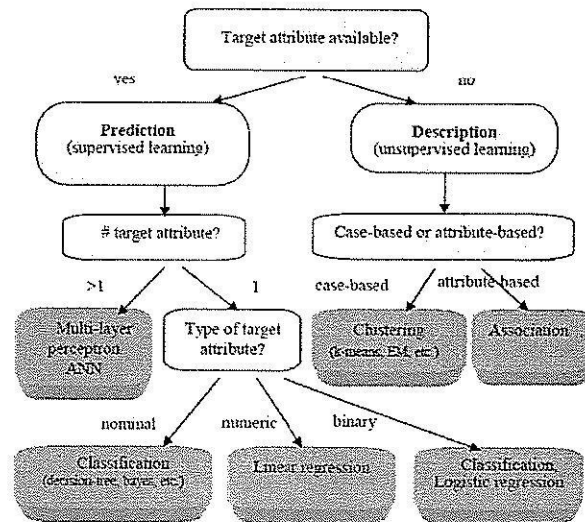


Figure 3. Ontology for guiding mining-method selection at the DM step

The DM component is for performing various mining tasks. Currently, we design and implement three different mining modules, i.e. classification, association, and clustering. We adopt the ontology concept at this step to guide the mining methodology selection. A simple form of ontology to select appropriate mining method is shown in Figure 3.

The Post-DM component composed of two main features: knowledge evaluator and knowledge integrator. These features perform functionalities aiming at a feasible knowledge deployment. Knowledge evaluator involves evaluation, based on corresponding measurement metrics, of the mining results. Knowledge integrator examines the induced patterns to remove redundant knowledge.

5 System Implementation

In this section, we present the Prolog coding of DM, a major module for mining different kinds of knowledge in the knowledge-induction-and-evaluation component. Prolog code is based on the syntax of SWI Prolog (www.swi-prolog.org).

Data format. The data to be used by any mining method of the DM module take the same format, that is, as a Prolog file. As an illustration, we use the allergy data of ten patients. The following data show information of ten patients suffering from allergy (class = yes). The possible indicative symptoms are sore throat, fever, swollen glands, congestion, and

headache. Some patients had some of these symptoms but are not suffering from allergy (class = no). To induce the common symptoms (or model) of allergy patients from this data, we have to save this data set as a Prolog file (data.pl).

```
%% Data: Allergy diagnosis
% Patients' symptoms and their possible values
attribute( soreThroat, [yes, no]).
attribute( fever, [yes, no]).
attribute( swollenGlands, [yes, no]).
attribute( congestion, [yes, no]).
attribute( headache, [yes, no]).
attribute( class, [yes, no]).

% data instances
instance(1, class=no, [soreThroat=yes, fever=yes,
    swollenGlands=yes, congestion=yes,
    headache=yes]).
instance(2, class=yes, [soreThroat=no, fever=no,
    swollenGlands=no, congestion=yes,
    headache=yes]).
instance(3, class=no, [soreThroat=yes, fever=yes,
    swollenGlands=no, congestion=yes,
    headache=no]).
instance(4, class=no, [soreThroat=yes, fever=no,
    swollenGlands=yes, congestion=no,
    headache=no]).
instance(5, class=no, [soreThroat=no,
    fever=yes, swollenGlands=no,
    congestion=yes, headache=no]).
instance(6, class=yes, [soreThroat=no, fever=no,
    swollenGlands=no, congestion=yes,
    headache=no]).
instance(7, class=no, [soreThroat=no,
    fever=no, swollenGlands=yes,
    congestion=no, headache=no]).
instance(8, class=yes, [soreThroat=yes, fever=no,
    swollenGlands=no, congestion=yes, headache=yes]).
instance(9, class=no, [soreThroat=no,
    fever=yes, swollenGlands=no,
    congestion=yes, headache=yes]).
instance(10, class=no, [soreThroat=yes,
    fever=yes, swollenGlands=no,
    congestion=yes, headache=yes]).
```

Classification. The objective of classification is to induce data model of two classes: positive (class = yes) and negative (class=no). Binary classification is a typical task in medical data mining. The code, however, can be easily modified to classify data with more than two classes. To induce the common symptoms (or model) of patients suffering from allergy, we use the decision-tree induction method [18]. The process starts when the following main module is invoked. Note that clauses containing higher-order predicates are highlighted throughout the given program code.

```
:-include('data.pl').
:-dynamic current_node/1,node/2,edge/3.

main :-
    init(AllAttr,EdgeList),
    getNode(N), % get node number
    create_edge(N,AllAttr,EdgeList),
    print_model.

init(AllAttr,[root-nil/PB-NB]) :-
    retractall(node(_,_)),
    retractall(current_node(_)),
    retractall(edge(_,_)),
    assert(current_node(0)),
    findall(X, attribute(X,_), AllAttr1),
    delete(AllAttr1, class, AllAttr),
    findall(X2,instance(X2,class=yes,_),PB),
    findall(X3,instance(X3,class=no,_),NB).

getNode(X) :-
    current_node(X),
    X1 is X+1,
    retractall( current_node(_)),
    assert( current_node(X1)).
```

The main module calls the *init* procedure (or predicate) to initialize the temporary knowledge base by removing all information that might be remained in the knowledge base and asserting the root node of the tree. The node and edge structures of our decision tree have the following formats:

```
node(nodeID, [PositiveCase]-[NegativeCase])
edge(ParentNode, EdgeLabel, ChildNode)
```

The node structure is composed of two parts: node-id and the mixture of positive and negative cases in that node. The edge is a link from parent node to child node. Each edge contains three pieces of information; that is, id of parent node, the edge label, and id of child node. Node id 0 is a special node representing a root node and it links to node number 1. The tree building starts with the *create_edge* and *create_nodes* procedures.

```

create_edge( _,_ ) :- !.
create_edge( _,_ ) :- !.
create_edge(N, AllAttr, EdgeList) :-
    create_nodes(N, AllAttr, EdgeList).

create_nodes( _,_ ) :- !.
create_nodes( _,_ ) :- !.
create_nodes(N, AllAttr, [H1-H2/PB-NB | T]) :-
    getNode(N1), % get node number N1
    assert(edge(N,H1-H2,N1)),
    assert(node(N1,PB-NB)),
    append(PB, NB, AllInst),
    ( (PB \== [], NB \== []) ->
        (cand_node(AllAttr, AllInst, AllSplit),
         best_attribute(AllSplit,
                        [V, MinAttr, Split]),
         delete(AllAttr, MinAttr, Attr2),
         create_edge( N1, Attr2, Split))
      ; true ),
    create_nodes(N, AllAttr, T).

best_attribute([], Min, Min).
best_attribute([H | T], Min) :-
    best_attribute(T, H, Min).
best_attribute([H | T], Min0, Min) :-
    H = [V, _ , _ ],
    Min0 = [V0, _ , _ ],
    ( V < V0 -> Min1 = H;
      Min1 = Min0),
    best_attribute(T, Min1, Min).

% generate candidate decision node
cand_node([],_ ) :- !.
cand_node( _,_ ) :- !.

```

```

cand_node([H | T], Ins, [[Val,H,SplitL] | Att]) :-
    info(H, Ins, Val, SplitL),
    cand_node(T, Ins, Att).

% compute Info of each candidate node
concat3(A,B,C,R) :-
    atom_concat(A,B,R1),
    atom_concat(R1,C,R).

info(A, CurInstL, R, Split) :-
    attribute(A,L),
    maplist( concat3(A,=), L, L1),
    suminfo(L1, CurInstL, R, Split).

suminfo([],_ ,0,[]).
suminfo([H | T], CurInstL, R, [Split | ST]) :-
    AllBag=CurInstL,
    term_to_atom(H1,H),
    findall(X1, (instance(X1,_ ,L1),
                member(X1, CurInstL),
                member(H1,L1)), BagGro),
    findall(X2, (instance(X2,class=yes, L2),
                member(X2, CurInstL),
                member(H1,L2)), BagPos),
    findall(X3, (instance(X3,class=no, L3),
                member(X3, CurInstL),
                member(H1,L3)), BagNeg),
    (H11=H22) =H1,
    length(AllBag, Nall),
    length(BagGro, NGro),
    length(BagPos, NPos),
    length(BagNeg, NNeg),
    Split = H11-H22/BagPos-BagNeg,
    suminfo(T, CurInstL, R1,ST),
    ( NPos is 0 ->L1 = 0;
      L1 is (log(NPos/NGro)/log(2)) ),
    ( 0 is NNeg ->L2 = 0;
      L2 is (log(NNeg/NGro)/log(2)) ),
    ( NGro is 0 -> R= 999;
      R is (NGro/Nall)*(-(NPos/NGro)*L1-
                (NNeg/NGro)*L2)+R1 ).

```

The given source code does not provide detail for *print_model* procedure. Interested readers are suggested to simply add a rule *print_model* :- true.

Then run the program by calling predicate *main*. Prolog will respond *true* with no other information because we simply add the always-true condition in the *print_model* predicate. At this moment we can view the tree model by calling *listing(node)* and *listing(edge)* predicates. The results will be as follows:

```
1 ?- main.
true.

2 ?- listing(node).
:- dynamic user:node/2.
user:node(1, [2, 6, 8]-[1, 3, 4, 5, 7, 9, 10]).
user:node(2, []-[1, 3, 5, 9, 10]).
user:node(3, [2, 6, 8]-[4, 7]).
user:node(4, []-[4, 7]).
user:node(5, [2, 6, 8]-[]).
true.

3 ?- listing(edge).
:- dynamic user:edge/3.
user:edge(0, root-nil, 1).
user:edge(1, fever-yes, 2).
user:edge(1, fever-no, 3).
user:edge(3, swollenGlands=yes, 4).
user:edge(3, swollenGlands=no, 5).
true.
```

The running results convey the following information. From node number 1, the edge with label fever-yes (representing attribute fever with a value yes) links to node number 2. Node 1 contains all ten cases of patients suffering and not suffering from allergy, whereas node 2 contains the information []-[1,3,5,9,10] to infer none of positive cases and five negative cases. Therefore, the results in the above node and edge structures represent the following data model:

```
class(allergy) :- fever=no,
                swollenGlands=no.
```

Association mining. We implement the association mining module based on the algorithm APRIORI [1]. The implementation shows only the first pass of the algorithm; that is, the generation of frequent itemsets. The second pass, which is the generation of association rules from frequent

itemsets, can be easily extended from the given code.

Main predicate of this module is *association_mining*. Upon invocation, this predicate obtains input data from the predicate *input(Data)*, and get the minimum support value through the predicate *min_support(V)*. Then the main predicate starts the process by making candidate and large itemsets of length one, two, three, and so on (through the predicates *makeC1*, *makeL*, and *apriori_loop*, respectively). All highlighted terms are higher-order predicates. These predicates are *maplist*, *include*, and *setof*.

The predicate *maplist* takes three arguments; therefore, it may be written as *maplist/3*. This predicate applies its first argument, which is also a predicate, to each element of a list appeared in the second argument. The result is a list in the third argument.

The predicate *include/3* takes another predicate as its first argument and adds the result obtained from the first argument to the list in second argument. The result appears as a list in the third argument. The predicate *setof/3* also works with other predicate to collect each answer as a list in its third argument.

```
association_mining :-
    input(Data),
    min_support(V),
    makeC1(C),
    makeL(C,L),
    apriori_loop(L,1).

apriori_loop(L,N) :-
    length(L) is 1,!.
apriori_loop(L,N) :- N1 is N+1,
    makeC(N1,L,C),
    makeL(C, Res),
    apriori_loop(Res, N1).

makeC1(Ans) :- input(D),
    allComb(1, ItemSet, Ans2),
    maplist(countSS(D), Ans2, Ans).

makeC(N,ItemSet,Ans) :- input(D),
    allComb(2,ItemSet, Ans1),
    maplist(flatten, Ans1, Ans2),
```

```

maplist(list_to_ord_set, Ans2, Ans3) ,
list_to_set(Ans3,Ans4),
include(len(N), Ans4, Ans5),
maplist(countSS(D), Ans5, Ans).

%scan database to find: List+N
makeL(C,Res) :- include(filter, C, Ans),
maplist(head, Ans, Res).

filter(_+N) :- input(A),
length(A, I),
min_support(V),
N>=(V/100)*I.

head(H+_,H).

% arbitrary subset of the set containing
% given number of elements
comb(0, _, []).
comb(N, [X|T], [X|Comb]) :-
N > 0,
N1 is N-1,
comb(N1,T,Comb).
comb(N,[_|T],Comb) :-
N > 0,
comb(N,T,Comb).

allComb(N,I,Ans) :-
setof(L, comb(N, I, L), Ans).

countSubset(A,[],0).
countSubset(A,[B|X],N) :-
not(subset(A,B)),
countSubset(A,X,N).

countSubset(A,[B|X],N) :-
subset(A,B),
countSubset(A,X,N1),
N is N1+1.

countSS(SL,S,S+N) :-
countSubset(S,SL,N).

len(N,X) :- length(X,N1), N is N1.

```

Clustering. We implement the data clustering based on k-means algorithm [12]. The main predicate is *clustering* in which the number of clusters (k) has to be specified and data are to be included. The predicate *makeInitCluster* creates initial k clusters with randomized k centroids, then assign each data to the closest centroid through the predicate *assignPoint*.

Note that the symbol ‘*’, such as those appear in the predicate *cmax(Res, A*V)* and *freq(X, N*Y, N*F)*, refers to the data format to represent *Attribute*Value*; it does not mean multiplication. In Prolog, numerical computation will occur in a clause with the predicate ‘is’, such as *SI is S + I* in the *reComputeCenter* procedure.

The iteration step, *repeatCompute* predicate, re-computes the new k centroids and then re-assign each data point to the new closest centroid. Iteration stops when all data do not change their clusters. The source code presented in the following works with categorical data. For numerical or data with mixing types, the distance measurement has to be modified.

```

clustering(K) :-
makeInitCluster(K, AllClust),
assignPoint(AllClust, Data, Start, AllPt),
OldClust=AllClust,
repeatCompute(K, AllPt, OldClust).

makeInitCluster(K, AllClust):-
initClust(K, 1, AllClust).

initClust(K, L0, []) :-
L0 > K, !.
initClust(K, L0, [L0*L|T]) :-
instance(L0,_,L),
L1 is L0+1,
initClust(K, L1, T).

assignPoint(_, U, M, []) :-
M > U, !.
assignPoint(AllClust, U, M, [M-V-A|T]) :-
maplist(freq(M), AllClust, Res),
cmax(Res, A*V),
M1 is M+1,
assignPoint(AllClust, U, M1, T).

```

```

freq(X, N*Y, N*F) :-
    instance(X, _, L1),
    intersection(L1, Y, I),
    length(I, F).

cmax(L, A*V) :-
    maplist(cvalue, L, L2),
    max_list(L2, V),
    member(A*V, L), !.

cvalue(_*V, V).

reComputeCenter(K, S, AllPoint, []) :-
    S > K, !.

reComputeCenter(K, S, AllPoint, [S*NewC | T]) :-
    findall(P, member(P, _S, AllPoint), Z),
    allPointAtAllAttr(Z, NewC),
    S1 is S+1,
    reComputeCenter(K, S1, AllPoint, T).

allPointAtAllAttr(AllP, NewClusters) :-
    findall(AttName, (attribute(AttName, _,
        AttName\==class), AttNameL),
    maplist(allPoint(AllP), AttNameL,
        NewClusters).

allPoint(AllP, Att, A) :-
    findall(Att=V, (instance(X, _, K),
        member(X, AllP),
        member(Att=V, K)), Z),
    maxFreq(Z, A*V).

maxFreq(L, A*V) :-
    findall(X*C, (member(X,L), count(X,L,C)), Z),
    cmax(Z,A*V).

repeatCompute(K, AllPt, OldClust) :-
    reComputeCenter(K,Start,AllPt,NewClus),
    ( OldClust==NewClus ->
        writeln('No-cluster-changes***End*');
    ( writeln(newClust-NewClus),
        assignPoint(NewClus,Data,Start,AllPt2),
        writeln(allNewPoint-AllPt2),
        repeatCompute(K, AllPt2, NewClus) ) ).

```

6 Conclusion

Huge amount of data collected by hospitals and clinics are not yet turned into useful knowledge due to the lack of efficient analysis tools. We thus propose a rapid prototyping of an automatic data-mining tool to induce knowledge from medical data. The induced knowledge is to be evaluated and integrated into the knowledge base of a medical decision support system. Discovered knowledge facilitates the reuse of knowledge base among decision-support applications within organizations that own heterogeneous clinical and health databases. One obvious application of such knowledge is to pre-process other data sets by grouping it into focused subset containing only relevant data instances.

Our implementation of knowledge induction engines is based on the concept of higher-order Horn clauses using the logic-programming paradigm. Higher-order programming has been originally appeared in functional languages and soon be ubiquitous in several modern programming languages such as Java. Higher order style of programming has shown the outstanding benefits of code reuse and high level of abstraction.

This paper illustrates higher order programming techniques in Prolog by means of higher-order predicates such as *maplist*, *findall*, *setoff*, and *include*. These predicates take other predicates as its argument. With such expressive power of higher-order predicates, program coding of the designed system is very concise as demonstrated in the paper. Program conciseness contributes directly to program verification and validation, which are important issues in software engineering.

The powerful feature of meta-level programming in Prolog facilitates the reuse of data-mining results represented as rules to be flexibly applied as conditional clauses in other applications. The plausible extension of our current work is to add constraints into the knowledge induction method in order to limit the search space and therefore yield useful and timely knowledge. We also plan to extend our system to work with stream data that normally occur in modern medical institutions.

Acknowledgements

This research has been funded by grants from the National Research Council and the Thailand Research Fund (TRF, grant number RMU5080026). Data Engineering and Knowledge Discovery

(DEKD) Research Unit has been fully supported by Suranaree University of Technology.

References:

- [1] R. Agrawal et al., Fast discovery of association rules, In U. Fayyad, G.Piatetsky-Shapiro, P. Smyth, and R.Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, AAAI Press, pp.307-328.
- [2] Y. Bedard et al., Integrating GIS components with knowledge discovery technology for environmental health decision support, *Int. J. Medical Informatics*, Vol.70, 2003, pp.79-94.
- [3] C. Bojarczuk et al., A constrained-syntax genetic programming system for discovering classification rules: Application to medical data sets, *Artificial Intelligence in Medicine*, Vol.30, 2004, pp.27-48.
- [4] E. German, A. Leibowitz, and Y. Shahar, An architecture for linking medical decision-support applications to clinical databases and its evaluation, *J. Biomedical Informatics*, Vol.42, 2009, pp.203-218.
- [5] S. Ghazavi and T. Liao, Medical data mining by fuzzy modeling with selected features, *Artificial Intelligence in Medicine*, Vol.43, No.3, 2008, pp.195-206.
- [6] M. Huang, M. Chen, and S. Lee, Integrating data mining with case-based reasoning for chronic diseases prognosis and diagnosis, *Expert Systems with Applications*, Vol.32, 2007, pp.856-867.
- [7] N. Hulse et al., Towards an on-demand peer feedback system for a clinical knowledge base: A case study with order sets, *J. Biomedical Informatics*, Vol.41, 2008, pp.152-164.
- [8] C.-P. Hung, H.-J. Su, and S.-L. Yang, Melancholia diagnosis based on GDS evaluation and meridian energy measurement using CMAC neural network approach, *WSEAS Transactions on Information Science and Applications*, 6(3), March 2009, pp.500-509.
- [9] E. Kretschmann, W. Fleischmann, and R. Apweiler, Automatic rule generation for protein annotation with the C4.5 data mining algorithm applied on SWISS-PROT, *Bioinformatics*, Vol.17, No.10, 2001, pp.920-926.
- [10] P.-J. Kwon, H. Kim, and U. Kim, A study on the web-based intelligent self-diagnosis medical system, *Advances in Engineering Software*, Vol.40, 2009, pp.402-406.
- [11] C. Lin et al., A decision support system for improving doctors' prescribing behavior, *Expert Systems with Applications*, Vol.36, 2009, pp.7975-7984.
- [12] J. MacQueen, Some methods for classification and analysis of multivariate observations, *Proceedings of the 5th Berkeley Symp. on Mathematical Statistics and Probability*, vol.1, pp.281-297.
- [13] E. Mugambi et al., Polynomial-fuzzy decision tree structures for classifying medical data, *Knowledge-Based System*, Vol.17, No.2-4, 2004, pp.81-87.
- [14] G. Nadathur and D. Miller, Higher-order Horn clauses, *J. ACM*, Vol.37, 1990, pp.777-814.
- [15] L. Naish, Higher-order logic programming in Prolog, *Technical Report 96/2*, Dept. Computer Science, Univ. Melbourne, Australia, 1996.
- [16] S.-H. Nienhuys-Cheng and R.D. Wolf, *Foundations of Inductive Logic Programming*, Springer, 1997.
- [17] B. Pandey and R.B. Mishra, Knowledge and intelligent computing system in medicine, *Computers in Biology and Medicine*, Vol.39, 2009, pp.215-230.
- [18] J.R. Quinlan, Induction of decision trees, *Machine Learning*, Vol.1, 1986, pp.81-106.
- [19] O. Rijal et al., A relook at logistic regression methods for the initial detection of lung ailments using clinical data and chest radiography, *WSEAS Transactions on Information Science and Applications*, 6(9), September 2009, pp.1503-1512.
- [20] T. Wah and O. Sim, Development of a data warehouse for lymphoma cancer diagnosis and treatment decision support, *WSEAS Transactions on Information Science and Applications*, 6(3), March 2009, pp.530-543.

SUT-Miner: A Knowledge Mining and Managing System for Medical Databases

Kittisak Kerdprasop and Nittaya Kerdprasop

Data Engineering and Knowledge Discovery (DEKD) Research Unit
School of Computer Engineering, Suranaree University of Technology
Nakhon Ratchasima, Thailand
{kerdpras, nittaya}@sut.ac.th

Abstract—Knowledge is a valuable asset to most organizations as a substantial source to support better decisions. Recently there has been an increasing interest in devising database and data mining technologies to automatically induce knowledge from biomedicine, clinical and health data. Most work had adopted a single technique in the knowledge induction process. We propose a knowledge mining system as an integrated environment storing a repertoire of tools for discovering strong and useful knowledge. We demonstrate the usefulness aspect through the semi-automatic trigger creation for the medical database. A rapid prototyping of association mining engine is also presented in the paper.

Keywords—knowledge mining; medical databases; triggers; association rules

I. INTRODUCTION

The term *knowledge assets* refer to any organizational intangible possessions related to knowledge such as know-how, expertise, intellectual property. In clinical companies and computerized healthcare applications, knowledge assets include order sets, drug-drug interaction rules, guidelines for practitioners, and clinical protocols [11]. Knowledge assets can be stored in data repositories either in implicit or explicit form. Explicit knowledge can be managed through the existing tools available in the current database technology. Implicit knowledge, on the contrary, is harder to achieve and retrieve.

Implicit knowledge acquisition can be achieved through the availability of the knowledge-mining system. *Knowledge mining* is the discovery of hidden knowledge stored possibly in various forms and places in large data repositories. In health and medical domains, knowledge has been discovered in different forms such as association, classification rules, clustering, trend or temporal pattern analysis [22]. The discovered knowledge facilitates expert decision support, diagnosis and prediction.

In this paper we present the design and implementation of a knowledge-mining system named SUT-Miner to support a high-level decision in medical domains. The system is also applicable to any domain that requires a knowledge-based decision support. A rapid prototyping of the proposed system is provided in a declarative style using second-order Horn clauses [15]. The intuitive idea of our design and implementation is that for such a complicated knowledge-based system program coding should be done declaratively at a high level to alleviate the burden of programmers.

The proposed system has been tested with diabetes database containing 768 cases to induce association rules from the underlying data instances. Application of the induced knowledge on creating database triggers is also demonstrated.

The outline of this paper is as follows. Section 2 reviews related work. Section 3 is the architecture of SUT-Miner. Section 4 shows the implementation of association mining engine. Section 5 demonstrates running example on diabetes database. Section 6 illustrates the application of the induced association rules. Section 7 concludes the paper and discusses our future research directions.

II. RELATED WORK

In recent years we have witnessed increasing number of applications mining knowledge from biomedicine, clinical and health data. Roddick et al [19] discussed the two categories of mining techniques applied over medical data: explanatory and exploratory. Explanatory mining refers to techniques that are used for the purpose of confirmation or making decisions. Exploratory mining is data investigation normally done at an early stage of data analysis in which an exact mining objective has not yet been set.

Explanatory mining in medical data has been extensively studied in the past decade employing various learning techniques. Bojarczuk et al [4] applied genetic programming method to discover classification rules from medical data sets. Thongkam et al [24] studied breast cancer survivability using AdaBoost algorithms. Ghazavi and Liao [9] proposed the idea of fuzzy modeling on selected features medical data. Huang et al [10] introduced a system to apply mining techniques to discover rules from health examination data. Then they employed a case-based reasoning to support the chronic disease diagnosis and treatments. The recent work of Zhuang et al [26] also combined mining with case-based reasoning, but applied a different mining method. Biomedical discovery support systems are recently proposed by a number of researchers [5, 7]. Some work [21] extended medical databases to the level of data warehouses.

Exploratory, as oppose to explanatory, is rarely applied to medical domains. Among the rare cases, Nguyen and Kawasaki [16] introduced knowledge visualization in the study of hepatitis patients. Palaniappan and Ling [18] applied the functionality of OLAP tools to improve visualization.

It can be seen from the literature that most medical knowledge discovery systems have applied only some mining techniques such as classification rules mining,

association mining to discover hidden knowledge. We, on the contrary, design a knowledge-mining system aiming at providing a suite of tools to facilitate users and medical practitioners on discovering different kinds of knowledge from their data and background knowledge repositories. The deployment of induced knowledge has also been demonstrated through the creation of database triggers.

Triggers are a major concept of active databases, which extend traditional database systems with the mechanism to respond automatically to some specific events. Upon the occurrence of the specified event, the rule condition is evaluated. If the condition is satisfied, then some actions are performed. Although triggers are important database feature on consistency monitoring, their deployment is still limited due to the fact that creating complex trigger rules is not an easy task [6, 13]. Tools and environments to aid users and database programmers are certainly needed.

In medical domain the employment of triggers to achieve active behavior is quite rare. Most of the proposed methods are for detecting static events such as the discovery of relationships that suggest risks of adverse events in patient records [17, 23], detection of dependency patterns of process sequences for curing brain stroke patients [14], the generation of rules to annotated protein data in medical database [12], or the exploration of environmental health data [3]. Our work differs from those appeared in the literature in that we propose to employ knowledge discovery techniques to semi-automatically create trigger rules. The utilization of our proposed method is to increase consistency in medical database. Any database modification events violating constraints will be alerted and undone. The system designed by Agrawal and Johnson [1] is also to support medical database but in a different aspect; they concentrate on security and privacy preservation of patients and other sensitive health data.

III. SYSTEM ARCHITECTURE

The process of knowledge mining is complex and iterative in its nature. We design the system (Fig. 1) to be composed of two phases: knowledge induction and knowledge inferring.

Knowledge induction is the back-end of the system responsible for acquiring and discovering new and useful knowledge. Usefulness is to be validated at the final step by human experts. Discovered knowledge is stored in the knowledge base to be applied to solve new cases or create new knowledge in the knowledge-inferring phase which is the front-end of the proposed system.

The system obtains input from heterogeneous sources. Therefore, redundancy, incompleteness, noise can be expected from the input data. The Pre-DM component has been designed to clean, transform the data format, and select only relevant data. The DM component is for performing various mining tasks. Currently, we design and implement three different mining engines, i.e. classification, association, and clustering.

The Post-DM component composed of two main features: knowledge evaluator and knowledge integrator. These features perform functionality aiming at a feasible

knowledge deployment, which is important for the applications in medical diagnosis and predicting. Knowledge evaluator involves evaluation, based on corresponding measurement metrics, of the mining results. Knowledge integrator examines the induced patterns to remove redundant knowledge. Ontology has also been applied at this step to provide essential semantics regarding the domain problems.

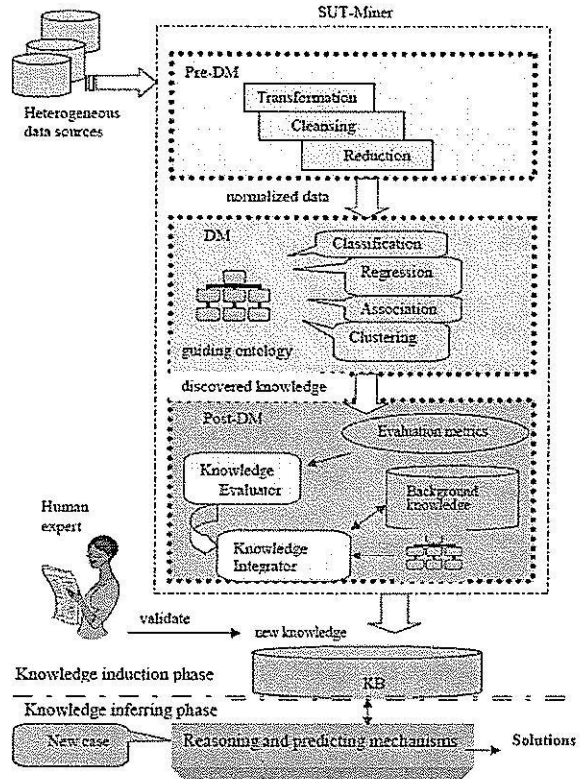


Figure 1. An architecture of SUT-Miner.

IV. IMPLEMENTATION

Our implementation is based on the concept of second-order Horn clauses, which is an extension of first-order logic (FOL) to be more expressive. FOL has been extensively used in intelligent systems [25] as an inference mechanism to deduce new facts. A classic example is that given a general rule that all men are mortal $\forall x \text{ Man}(x) \Rightarrow \text{Mortal}(x)$ and a known fact that Socrates is a man $\text{Man}(\text{Socrates})$, we can deduce new fact that $\text{Mortal}(\text{Socrates})$.

Despite its successful application to many computational problems such as natural language processing, FOL poses a restriction on the type of variables appearing in quantifications to exclude predicates. Second-order logic [15], on the other hand, allows variables to quantify over predicates. With such relaxation, second-order logic facilitates the implementation of a knowledge-intensive system that takes other knowledge as its input in a closed form.

```

association_mining :-
    input(Data),
    min_support(V),
    makeC1(C, makeL(C, L),
    apriori_loop(L,1).
apriori_loop(L, N) :-
    length(L) is 1, !.
apriori_loop(L, N) :-
    N1 is N+1,
    makeC(N1, L, C),
    makeL(C, Res),
    apriori_loop(Res, N1).
makeC1(Ans) :-
    input(D),
    allComb(1, ItemSet, Ans2),
    maplist(countSS(D), Ans2, Ans).
makeC(N, ItemSet, Ans) :-
    input(D),
    allComb(2, ItemSet, Ans1),
    maplist(flatten, Ans1, Ans2), maplist(list_to_ord_set,
    Ans2, Ans3), list_to_set(Ans3, Ans4),
    include(len(N), Ans4, Ans5), maplist(countSS(D),
    Ans5, Ans).

                %scan database to find: List+N
makeL(C, Res) :- include(filter, C, Ans),
                maplist(head, Ans, Res).
filter(_+N) :- input(A),
                length(A, I),
                min_support(V),
                N>=(V/100)*I.
head(H+_, H). % The sign + represents pattern, not addition
% arbitrary subset of the set containing given no. of elements
comb(0, _, []).
comb(N, [X | T], [X | Comb]) :- N > 0, N1 is N-1,
                                comb(N1, T, Comb).
comb(N, [_ | T], Comb) :- N > 0,
                            comb(N, T, Comb).
allComb(N, I, Ans) :- setof(L, comb(N, I, L), Ans).
countSubset(A, [], 0).
countSubset(A, [B | X], N) :- not(subset(A, B)),
                                countSubset(A, X, N).
countSubset(A, [B|X], N) :- subset(A, B),
                                countSubset(A, X, N1),
                                N is N1+1.
countSS(SL, S, S+N) :- countSubset(S, SL, N).
len(N, X) :- length(X, N1), N is N1.

```

Figure 2. Association mining in SUT-Miner implemented with second-order logic using Prolog.

In Fig. 2, a second-order logic-based implementation of association rule mining [2] has been demonstrated. The coding is based on the syntax of SWI prolog (www.swi-prolog.org). The main module of this program is the predicate *association_mining*. The predicate *apriori_loop* implements the Apriori algorithm [2] with the specified minimum support value *V*. The first candidate set of association is to be found through the predicate *makeC1*, in which the frequently occurred ones are discovered by the predicate *makeL*. The next candidate sets are to be generated by the predicate *makeC*.

From our coding, the SOL concept has been utilized through the predicates *maplist*, *include*, and *setof*. The use of these second-order Horn clauses contributes significantly to program conciseness and the ease of program verification.

V. EXPERIMENTATION

We have tested our association mining implementation with the diabetes dataset taken from the UCI database (<http://www.ics.uci.edu/~mllearn/MLRepository.html>). This medical data set is a collection of 768 observations on female patients investigating whether the patient shows signs of diabetes (class=1) or not (class=0) according to World Health Organization criteria. Each patient's record contains eight attributes: number of times pregnant, plasma glucose concentration (a two hours in an oral glucose tolerance test), diastolic blood pressure, triceps skin fold thickness (mm.), 2-hour serum insulin, body mass index, diabetes pedigree function, and age. The best five accurate association rules are shown in Fig. 3. Each rule is annotated with the number of cases supporting the induced association.

1. IF *triceps-thickness*='{0-9.9}'
AND *diabetes-pedigree-fn*='{0-0.3122}'
THEN *2Hr-serum-insulin*='{0-84.6}'
(support= 128 cases out of 768)
2. IF *triceps-thickness*='{0-9.9}'
AND *diabetes-pedigree-fn*='{0-0.3122}' AND *class*=0
THEN *2Hr-serum-insulin*='{0-84.6}'
(support= 83 cases out of 768)
3. IF *diastolic-pressure*='{73.2-85.4}'
AND *triceps-thickness*='{0-9.9}'
THEN *2Hr-serum-insulin*='{0-84.6}'
(support= 75 cases out of 768)
4. IF *times-pregnant*='{3-5}'
AND *triceps-thickness*='{0-9.9}'
THEN *2Hr-serum-insulin*='{0-84.6}'
(support= 52 cases out of 768)
5. IF *diastolic-pressure*='{73.2-85.4}'
AND *triceps-thickness*='{0-9.9}' AND *class*=0
THEN *2Hr-serum-insulin*='{0-84.6}'
(support= 48 cases out of 768)

Figure 3. Association rules that are mined from the diabetes database.

VI. KNOWLEDGE DEPLOYMENT

We design a framework (Fig. 4) to add active behavior to the medical database through the induced trigger rules. There are three major components in our model: mining, trigger generation and conflict resolution components. Mining component induces knowledge from the database contents and presents as rules: association and classification rules. The data repository contains both base data and trigger rules. Trigger generation component is responsible for converting induced classification/association rules into trigger format then stores generated triggers in the repository. In case of trigger rule application and rule conflict occurs, conflict resolution component will handle the situation.

We demonstrate the application of our proposed framework towards database consistency monitoring through the following example.

Example: Trigger generation on diabetes database.

From the first association rule in Fig. 3,

```
IF triceps-thickness='{0-9.9}'
  AND diabetes-pedigree-fn='{0-0.3122}'
  THEN 2Hr-serum-insulin='{0-84.6}'
```

the database trigger can be created as follows:

```
CREATE TRIGGER rule_1 ON diabetes FOR UPDATE,
INSERT
AS IF (SELECT COUNT(*) FROM diabetes
WHERE (triceps-thickness = '{0-9.9}')
and (diabetes-pedigree-fn= '{0-0.3122}' )
and (2Hr-serum-insulin <> '{0-84.6}') ) > 0
BEGIN
RAISERROR ('soft constraint violation, please verify');
END
```

This trigger will raise a warning message upon any database updates that violates the rule "any female patient with triceps thickness in the range 0-9.9 mm and diabetes pedigree function in the range 0-0.3122, 2-hour serum insulin has to be in the range 0-84.6". Any attempt to insert violating data will fire this trigger to draw attention from database administrator. Such trigger rule is thus deployed as a tool to support database integrity checking.

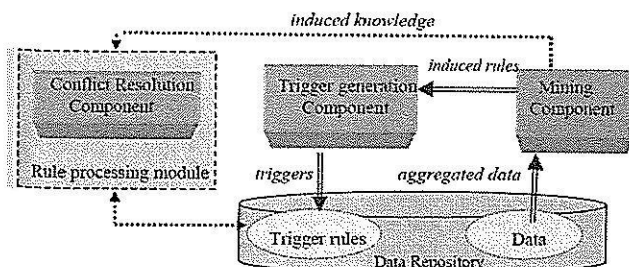


Figure 4. Incorporating induced knowledge as a set of trigger rules in medical databases.

VII. CONCLUSION AND DISCUSSION

We have proposed the design and implementation of SUT-Miner, a knowledge-mining system. The system is intended to support knowledge acquisition in medical data and other domains that require new knowledge to support better decisions. The proposed knowledge discovery environment is composed of tools and methods suitable for various kinds of mining such as data classification, clustering, association-rule mining.

The implementation of the proposed system has been done based on the concept of second-order logic, which is an extension of the well-known first-order logic. With the expressive power of higher-order logic, program coding of the designed system is very concise as demonstrated in the paper. Program conciseness contributes directly to program verification and validation, which are important issues in software engineering. The declarative style of programming also eases the future extension of our system to cover the concepts of higher-order mining [20] and constraint programming [8] that should naturally be applied to the task of knowledge mining.

We also provide a framework to semi-automatically generate trigger rules from current database contents by means of association mining technique. Induced trigger rules, in addition to predefined triggers, can be viewed as supplementary constraints to help increasing database consistency. Our proposed framework is thus a preliminary design of active medical databases.

This work is a rapid prototype of our big project. We plan to further our implementation on the knowledge inferring part and also test the knowledge induction component on various medical data sets. The induced knowledge is also to be verified by the doctors.

ACKNOWLEDGMENT

This research has been funded by grant from the National Research Council. The first author has been supported by a grant from the Thailand Research Fund (TRF, grant number RMU5080026). The Data Engineering and Knowledge Discovery (DEKD) Research Unit is fully supported by the research grants from Suranaree University of Technology (SUT), in which our system has been named after.

REFERENCES

- [1] R. Agrawal and C. Johnson, "Securing electronic health records without impeding the flow of information", *Int. J. Medical Informatics*, 76, 2007, pp.471-479.
- [2] R. Agrawal and R. Srikant, "Fast algorithm for mining association rules", *Proc. 20th VLDB*, 1994, pp.487-499.
- [3] Y. Bedard et al., "Integrating GIS components with knowledge discovery technology for environmental health decision support", *Int. J. Medical Informatics*, 70, 2003, pp.79-94.
- [4] C. Bojarczuk, H. Lopez, A. Freitas, and E. Michalkiewicz, "A constrained-syntax genetic programming system for discovering classification rules: Application to medical data sets", *Artificial Intelligence in Medicine*, 30, 2004, pp.27-48.
- [5] C. Bratsas, V. Koutkias, E. Kaimakamis, P. Bamidis, G. Pangalos, and N. Maglaveras, "KnowBaSICS-M: An ontology-based system for semantic management of medical problems and computerised