KGL: A Language for Learning

Kenneth A. Kaufman
Ryszard S. Michalski

MLI 97-3

KGL: A LANGUAGE FOR LEARNING

Kenneth A. Kaufman and Ryszard S. Michalski*

Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA 22030-4444

MLI 97-3
P97-4

March 1997

* Also Institute of Computer Science, Polish Academy of Sciences

KGL: A LANGUAGE FOR LEARNING

Abstract

In real-life data mining endeavors, the extraction of important knowledge may require many trials and errors, and multiple executions of different sequences of data mining operations. Such applications may pose a variety of tasks, for example, determining a *characteristic description* or a *discriminant description* of given classes of entities, *optimizing* an initial hypothesis according to a cost function, determining the *most relevant attributes* for a given task, selecting the *most representative examples* from a large example set, *conceptually clustering* cases into classes, *predicting* the class membership of a new example, generating a *decision structure*, automatically *determining a learning curve*, etc. The application of these sequences of programs can be time-consuming, laborious, and error prone.

In response to these challenges, it is important for machine learning to develop a methodology for integrating diverse learning strategies so that a learning system can pursue different learning tasks and acquire different kinds of knowledge, depending on the problem at hand. We have developed a high-level language, called KGL, in which a data analyst can plan data-mining and knowledge discovery experiments using various operators for performing learning and discovery tasks. The presented approach implements a range of learning and knowledge processing programs as KGL operators. Using KGL, a user can specify a plan for applying these operators in a flexible and interdependent manner in pursuit of a desirable solution. The methodology and language are illustrated by a problem of detecting demographic and economic patterns from a database of 190 countries. The results show a great potential of the proposed approach for data mining.

Keywords: Multistrategy Learning, Integrated Learning, Knowledge Discovery

Acknowledgments

The authors thank Ibrahim Imam, Seok Won Lee and Jim Mitchell for their comments and criticism on earlier versions of this paper.

This research was conducted in the Machine Learning and Inference Laboratory at George Mason University. The Laboratory's activities are supported in part by the Defense Advanced Research Projects Agency under Grant No. F49620-95-1-0462 administered by the Air Force Office of Scientific Research, in part by the National Science Foundation under Grants No. DMI-9496192 and IRI-9020266, and in part by the Office of Naval Research under Grant No. N00014-91-J-1351.

1 INTRODUCTION

One of machine learning's modern challenges is to integrate many different strategies in a single system so that the system can pursue different learning tasks and acquire different types of knowledge depending on the problem at hand. This calls for the development of *multistrategy* systems that can call upon diverse learning methods and/or representation strategies in the process of learning (Michalski and Tecuci, 1991, 1993; Michalski and Wnek, 1996). The idea of multistrategy learning is particularly interesting for the task-oriented exploration of large data bases in real-world applications.

Among the tasks that may be called upon by these applications are such processes as determining a *characteristic description* of a single class of entities, determining a *discriminant description* of several classes of entities in the context of other classes of entities, *optimizing* an initial hypothesis according to an objective function (that may seek, for example, a maximally simple or minimum cost description), determining the *most relevant attributes* for a given task, selecting the *most representative examples* from a very large example set, *conceptually clustering* cases into classes, *predicting* the class membership of a new example, generating a *decision structure* (e.g., a tree), and automatically *determining a learning curve*.

The central problem in developing a multistrategy learning system is how to integrate and use the different strategies in the pursuit of given tasks. In the case of applying such a system to data exploration and knowledge discovery, the issue is what kind of strategies to apply to the data under which conditions, and in which order. One major difficulty in building such a multistrategy system is how to apply individual strategies and knowledge processing operators in a flexible and mutually dependent way, so that a desired solution can be obtained. Another difficulty is to determine which newly generated results should be regarded as useful.

In the last few years we have been developing the INLEN system for multistrategy learning and data exploration based primarily on the methods and techniques developed in machine learning. INLEN (Michalski et al, 1992) integrates a range of *knowledge generation operators*, many of which represent different programs originally developed for use in stand-alone machine learning applications. These operators can be viewed as different *knowledge transmutations*, as defined in the Inferential Theory of Learning (Michalski, 1994). The use of these operators allows a user to discover general patterns, trends or exceptions in data that may not be apparent through simple observation. The results of applying these operators may also suggest the selection of a subsequent set of experiments that may otherwise not have been noticed.

INLEN makes it easy for the user to apply various operators, but the application requires explicit decisions by the user at every step. The disadvantage of this approach is that the analyst may need to inspect the results of each step of the process in order to determine which operator to apply next. This process can be laborious and time-consuming, and the analyst will be very prone to errors.

There may exist rules of control suitably general that they can be applied to a large proportion of knowledge discovery domains, and hence embedded within a discovery system. Other control rules may be more domain- and task-dependent. Individual users may have their own collection of necessary repeating tasks to perform on the data. It is therefore important that a user can articulate desires and expertise to such a multistrategy system, beyond serial invocations of the basic operators, in such a way that the system can perform independently for lengthy sequences of actions.

In response, we have developed KGL-1, a prototype high-level language for knowledge discovery that has been implemented in the INLEN-3 version. The language allows the user to create plans for guiding the system through various contingencies. Such a language, if versatile enough, will allow the user to write simple programs to accomplish very complex tasks. These programs may be executed once, periodically or on the occurrence of some event, such as an infusion of new information into a database or the perception of some pattern in the data or discovered knowledge.

This research attempts to identify metaknowledge for operator control that can take a form such as "If a certain **condition** is satisfied by a dataset or the current associated hypotheses, apply the following sequence of **operators** with the following **parameters**." For example, in a domain in which the database is continuously being updated, a rule may be used such as, "If more than 3% of the records in the database have been modified or added since the last application of this rule, test the knowledge base for continued applicability to the new data. If its degree of match falls below a 90% threshold, apply incremental learning to improve the knowledge, and analyze the records that do not match the original rules to see if they share any interesting common bonds."

In general, the requirements of such a high-level language for learning include:

- Invoking different types of programs for learning and knowledge discovery as single operators.
- Looping and branching abilities similar to those found in programming languages.
- Discrimination among the different attributes in the database. The user should be able to classify the attributes into groups based on importance, type, number of legal values, etc. With such a feature, the user may specify "partial grand tours" of the database, such as "Use the Set Differentiation operator to generate a full set of classification rules using all nominal attributes with fewer than 5 legal values as decision variables."
- Discrimination among the different rules, rulesets, decision structures, etc. that make up a domain's knowledge base. For instance, the user should be able to select rules based on complexity, strength, typicality, etc.
- Data-driven control strategies, triggered by changes to a database beyond a given threshold level. Among the patterns that must be detectable are missing values and conflicts with the existing knowledge base.
- Knowledge-driven control strategies, triggered, for example, by the discovery of especially strong patterns or exceptions. The program must be able to examine a piece of discovered knowledge and identify some of the attributes of the knowledge itself.

2 KNOWLEDGE GENERATION OPERATORS IN INLEN

The general design of the INLEN-3 version of the system for multistrategy learning and discovery is shown in Figure 1. The system consists of a relational database for storing known facts about a domain, and a knowledge base for storing rules, constraints, hierarchies, decision trees, equations accompanied with preconditions, and enabling conditions for performing various actions on the database and/or knowledge base. To be able to perform operations that require both data and knowledge with ease, the concept of a *knowledge segment* has been introduced (Kaufman, Michalski and Kerschberg, 1991). Knowledge segments are used for passing results from one operator to another and to the user. One research issue addressed in INLEN's development has been how to implement and utilize such knowledge segments.

The structure of the knowledge base and its relationship with the KGL-1 language are discussed in the following section.

The purpose for integrating the above capabilities is to provide a user with a set of advanced tools to search for and extract useful knowledge from a database, to organize that knowledge from different viewpoints, to test this knowledge on a set of facts, and to facilitate its integration within the original knowledge base. These tools in INLEN are known as *knowledge generation operators* (KGOs), which are designed to complement one another, and to be capable of performing many types of learning. These operators allow a user to discover general patterns, trends or exceptions in data that may not be apparent through simple observation. The results of applying these operators may also suggest the selection of a subsequent set of experiments that may otherwise not have been noticed. Systems with similar integrated, multi-operator architectures include KEPLER (Wrobel et al, 1996) and DBMiner (Han et al, 1996).

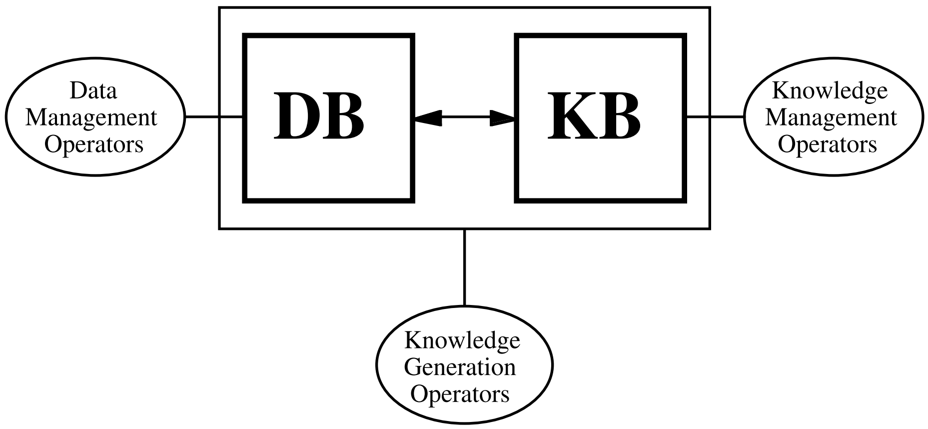

Figure 1. High-level architecture of INLEN

INLEN-3 is a prototype system, whose focus is on tool integration capabilities and testing of the design. As a result, some of the knowledge generation operators (Michalski et al, 1992) have yet to be implemented within INLEN. Below are listed the classes of operators which are part of INLEN-3, organized by category:

GENRULE: Generate Rules

Operators in the GENRULE class take some form of data and/or knowledge as an input, and return a ruleset consisting of facts induced from the input examples. They differ from one another in the type of rules generated (characteristic or discriminant).

GENSTRUCT: Generate Decision Structures

Operators in the GENSTRUCT class take some form of data and/or knowledge (i.e., either training examples or a ruleset) as an input, and return a decision structure (possibly, but not necessarily a decision tree; Michalski and Imam, 1994; 1997) outlining the procedure for discriminating between classes.

GENHIER: Generate Conceptual Hierarchies

The GENHIER operator creates hierarchies of objects through conceptual clustering.

GENEQ: Generate Equations

The GENEQ operator creates equations characterizing sets of qualitative and quantitative data.

TRANSFORM: Transform Knowledge

The TRANSFORM operators perform basic inferential transformations on knowledge, such as abstraction/concretization with respect to structured attributes (attributes whose values are explicitly organized into an abstraction hierarchy; see Kaufman and Michalski, 1996) and (real-valued) numeric attributes, and optimization of preliminary hypotheses.

GENATR: Generate Attributes

The GENATR operators map relational tables to relational tables whose rows are the same but whose columns have been changed, either by the addition of new attributes (through constructive induction) or by the removal of old ones (through selection).

GENEVE: Generate Events

The GENEVE class covers a wide variety of operators that generate a set of tuples, either as a subset of an existing relational table, or from the entire event space to which a table's tuples belong.

TEST: Test Knowledge

The TEST operator determines the performance of a ruleset on a set of examples by testing the input knowledge for consistency and completeness with regard to a set of classified input examples. The output analysis includes numerical weights indicative of the quality of the input knowledge. The primary output table is in the form of a *confusion matrix*, i.e. a matrix whose $(i,j)^{th}$ element shows how well the *i*th example matched the rules for class *j*.

Prior versions of INLEN made it easy for the user to apply the different operators, but their application required explicit decisions by the user at every step. Most of these operators are based on complex machine learning programs that may require the specification of a number of parameters to reflect accurately the nature and constraints of the problem at hand. For complex and large databases, their execution might take a considerable amount of time, and generate a considerable amount of output for analysis. Therefore, the risks of errors and inefficiency are augmented for such tasks.

To address this problem, the idea of developing a very high-level language for planning experiments in INLEN has been proposed. This paper briefly introduces to the language, called KGL, that has been implemented in INLEN, and presents examples of its application to the problem of demographic pattern discovery.

3 KNOWLEDGE BASE ORGANIZATION AND ACCESS IN INLEN AND KGL

Generated knowledge ideally serves many purposes. It may contain declarative or procedural knowledge on which decision-making can be based. It may contain equations for reference and deductive reasoning. It may contain organizational background knowledge to define relationships among different concepts. And it may contain references to examples of concepts, whether as representative, exceptional, or the basis for the generation of some other knowledge.

In order that the input and output information requirements of INLEN's knowledge base and the programs that make up its knowledge generation operators may be met, the design of the knowledge segment is critical. The knowledge base must be able to integrate knowledge represented in the different forms described above. Structures need to be designed that can represent the information stored by each of these knowledge types, support interactions between them, and allow the various operators, including the meta-operators taking the form of a KGL program, to extract what they need from the knowledge base.

As a result, the rules in INLEN's knowledge base have evolved the following syntactic structure:

Referent :	An attribute, or several attributes linked by (internal) conjunction or disjunction
*Relation Symbol (**Rel**):*	=, <>, >, <, >=, or <=
Reference:	A legal value of the attribute(s) comprising the associated Referent, an (internal) disjunction, or a range of such Values
Condition:	[<Referent> **<Rel>** <Reference>]
Condition Weights:	Numerical information indicating the number of datapoints (training examples) of the positive and negative classes satisfying a given relation
Weighted Condition:	<Condition> <Condition Weights>
(Weighted) Description:	a conjunction of (Weighted) Conditions
Rule Weights:	Numerical information characterizing a Description, e.g., number of training examples satisfying the Description
Example Keys:	Identifiers of individual training examples satisfying an associated Description
Rule:	An implication, <Description> => <Condition>, annotated by Rule Weights and Example Keys
Class Ruleset:	a set of Rules with the same consequent Condition
Ruleset:	a set of Class Rulesets, whose consequent Conditions share the same Referent attribute
Session:	A learning session, defined by a timestamp, input dataset, and output Ruleset

An example of a rule in this schema, is shown in Figure 2, extracted from a ruleset derived from the 1993 Central Intelligence Agency World Factbook. The countries of the world had been divided into classes based on their fertility rate, with the lowest class describing the 42 countries with fertility rates less than 2 per 1000 population. One of the discovered rules characterizing this class and also differentiating it from countries with higher fertility rates described 16 countries, including Andorra and Antigua. It consisted of eight conditions including one, "Birth Rate is between 10 and 20," that described all 42 low-fertility countries and only 20 others.

Fertility < 2 per 1000 population if: (42 examples)

		Pos	Neg
A.1.	[Birth Rate is 10..20]	42	20
2.	[Religion is R. Catholic or Orthodox or Anglican or Shinto]	24	31
3.	[Infant Mortality Rate <= 40]	41	54
4.	[Population Growth Rate is 3..4 or <= 1]	32	56
5.	[Literacy >= 70]	35	71
6.	[Life Expectancy = 60..80]	41	92
7	[Death Rate = 5..15]	42	102
8.	[Net Migration Rate >= -10]	42	140

16 Examples covered: Keys = Andorra, Antigua, Austria, Belgium, …

Figure 2. A rule in INLEN's knowledge base, as presented to the user

The presented *Rule* is made up of a consequent *Condition* [Fertility < 2], the *Rule Weights* indicating 16 examples satisfying the rule's *Description* out of 42 training examples in the Fertility < 2 class, the *Example Keys* of the 16 countries that satisfy the *Description*, and the *Description* consisting of eight *Weighted Conditions*. The first of the *Weighted Conditions*, consists of the *Referent* Birth Rate, the *Relational Symbol* =, the *Reference* range 10..20, positive *Weight* 42, and negative *Weight* 20.

The structure of the rules and rulesets described above can represent very complex functions and conditions without difficulty and, at the same time, can be easily interpreted conceptually. Consequently, this description language, which is based on the ideas of the variable-valued logic language VL1 (Michalski, 1973) and Annotated Predicate Calculus (Michalski, 1983), has a very high descriptive power as well as cognitive simplicity.

This schema leads to a natural division of a rule base into subunits, each describable by a set of pertinent statistics (Ribeiro, Kaufman and Kerschberg, 1995). An example of this structure is shown in Figure 3. The application of an operator (an individual learning *Session*) is associated with an input data set, the time at which it was run, and a set of output *Class Rulesets*. Each of these Class Rulesets can be associated with the class of objects it describes, statistics on the training sets associated with it, and the individual *Rules* that make up the Class Ruleset. The Rules are associated with the training examples that satisfied them through *Rule Weights* and *Example Keys*, and also with the individual *Weighted Conditions* within the Rules. The knowledge segments corresponding to these Weighted Conditions can be associated with the *Condition* statement and the various pertinent *Condition Weights,* such as the number of positive and negative training examples that satisfy it. In this format, INLEN's knowledge base stores a set of decision rules.

Figure 4 shows an instantiation of this architecture - a knowledge segment from which the rule shown in Figure 2 was derived.

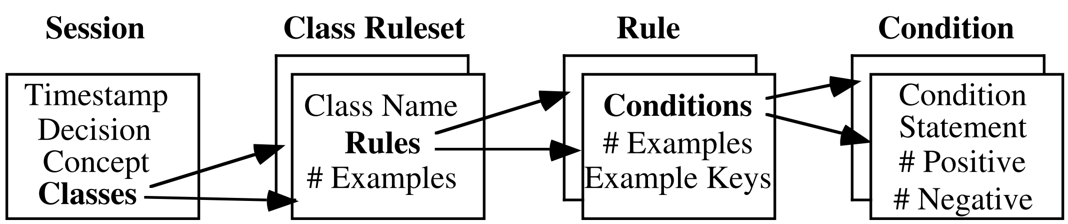

Figure 3. Knowledge organization for a decision ruleset.

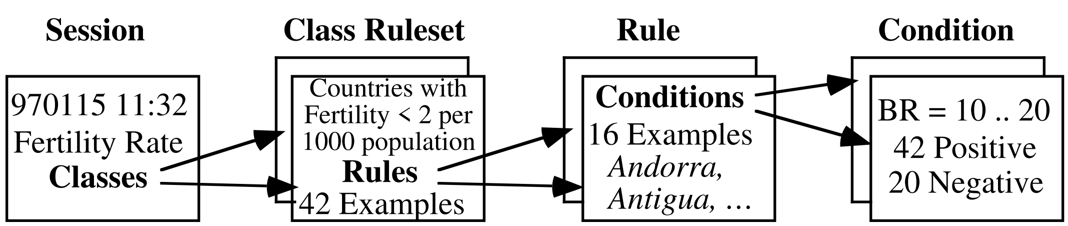

Figure 4. Exemplary instantiation of the schema presented in Figure 3

The KGL interpreter is able to extract details from the knowledge base as requested, so that it may respond to queries for elements in the knowledge base such as:

- The conditions in rules for the class Fertility < 2 whose positive example coverage outnumber their negative example coverage by at least a 2 to 1 ratio.
- Rules for any concept (not only Fertility) that are satisfied both by Andorra and Antigua.

- Rules that include the condition Birth Rate = 10 .. 20.

4 DESIGN OF THE KNOWLEDGE GENERATION LANGUAGE KGL

4.1 Basic Structure of KGL

The primary goal in the design of the knowledge generation language KGL is the ability to interact with INLEN data and knowledge bases to a degree at least equivalent to INLEN's capabilities in interactive mode. Simplicity of implementation was often chosen over elegance of the language. As such, the first version of KGL is a command-based language whose constructs take on the form <verb> <object> <parameters>.

The concepts upon which the language is based include the condition-rule structure described in the previous section. Other data structures utilized by KGL include:

Datatable: A table whose first row consists of a list of attributes and whose subsequent rows correspond to object descriptions in terms of these attributes, where each row describes one object.

Classified datatable: A datatable divided to three sets of columns: one set is associated with independent attributes, another with dependent attributes, and the third with object identifiers.

Object description: A set of attribute-value pairs characterizing a given object or situation; a row in the datatable containing values of independent attributes.

Example: An ordered triple consisting of a unique identifier (a key), an object description, and the name of the class to which example belongs; a row in a classified datatable.

Knowledge generation operator: A learning or other form of knowledge processing program that, given a datatable and a parameter set, generates an output that represents the desired knowledge (in the form of a ruleset, a new datatable, or a combination of both) with characteristics defined by the parameter set.

*Parameter set (**Params**) of an operator:* A set of parameters that specify how to run a knowledge generation operator. Each operator has a default set of parameters; when *Params* is not included in the argument list of the operator or a particular parameter is omitted from the parameter set, the operator will apply the default parameters.

Among the knowledge structures utilized by KGL are:

Characteristic description (of a class): A characterization of the examples of a given class in terms of the defined attributes and the values present in the class examples. The description takes the form of a Class Ruleset.

Discriminant description (of a class against a given set of examples or classes): A Class Ruleset such that each example of the target class satisfies at least one of its component Rules, while none of the examples to be discriminated against satisfy any of those Rules. Thus, it is a set of Rules to distinguish one set of examples from another.

Conceptual clustering (of a datatable): A transformation of an unclassified datatable into a classified datatable (through the creation of a new attribute representing the classification and a collection of Class Rulesets; each Class Ruleset is a characteristic description of the examples of one of the created classes.

4.2 Knowledge Processing Operators supported by KGL

This section describes the knowledge processing operators that may be invoked by KGL calls. It should be noted that these descriptions are functional ones; the KGL syntax will differ slightly. For example, in the operator CHAR(Datatable, Class, Params), Datatable does not need to be specified; KGL automatically uses the one currently in use.

CHAR(Datatable, Class, Params): Characterize a set of entities of a given class in the given datatable (i.e., build a characteristic description), using the given parameters to guide the characterization process.

DIFF(Datatable, Class, Negative Examples, Params): Differentiate entities of the given class in the given datatable from the set of negative examples (typically but not necessarily all examples of other classes in the datatable), using the given parameters to guide the differentiation process.

SELECT(Entities, Datatable, Params): Select components, specified by *entities* (an expression defining the entities to be selected, such as examples (rows), attributes (columns), or both), of the given datatable to be selected according to the criteria specified by Params, and thus form a smaller datatable that is a subset of the original one. The parameters may be deterministic (e.g., "select the first 50 rows") or based on the result of some data analysis (e.g., "select the 5 columns corresponding to the attributes that will be most useful in differentiating between the values of a given dependent attribute).

TEST(Datatable, Ruleset, Params): Test the knowledge contained in the given ruleset against a set of testing examples provided in the given datatable, using the given testing parameters. Report on the ruleset's consistency and completeness.

PREDICT(Ruleset, Example, Params): Predict the class of the given example based on the given rules and inference parameters.

Currently under development for KGL are the following operators for generating conceptual clusters and for generating decision structures.

STRUCT(Input, Params): Generate a decision structure for classification based on the input, which may be a classified datatable or a ruleset, and a set of parameters to guide the operator's search.

CLUSTER(Datatable, Params): Conceptually cluster the rows of the datatable into classes, guided by the parameter set. Return the classified datatable and characterizations of the created classes.

4.3 Commands Supported by KGL

The following commands are supported in KGL-1:

assign: Assignment operator - can take either of two forms: *assign <variable> <expression>* or *<variable> = <expression>*. In either form it assigns to the variable the result of the evaluation of the specified expression. The expression may be a string, a constant, the name of a variable, or an expression combining any of those forms with numerical, relational, logical, or functional operators.

begin/end: Delineators of a block of commands

display: Taking the form *display <expression list>*, this command writes the values specified by the expression list upon the screen.

do: Taking the form *do <operator>(<argument list>)*, this command invokes a knowledge generation operator, using the given set of arguments.

for: Performs a block of commands a certain number of times. It takes the form: *for <variable> = <min> to <max>* followed by the block to be repeated.

forall: Performs a block of commands upon each referenced object (e.g., a rule or condition) that satisfies the given criteria. It takes the form: *forall <object_type> (<criteria>)* followed by the block to be performed.

if: Tests the condition following the *if* for veracity. If it is true, perform the next block. Optionally, that block may be followed by an *else* command to indicate a block to be performed if the condition is not true.

open: Selects the active data table.

print: Identical to the display command, except output is written to an output file.

while: Tests the condition following the *while* for veracity, and repeatedly performs the next block as long as the condition remains true.

5 EXEMPLARY PROBLEM: LEARNING DESCRIPTIONS AND PATTERNS IN A LARGE DATABASE OF COUNTRIES

5.1. A KGL Program for Analyzing the PEOPLE Database

This problem is concerned with searching a database of 190 countries for patterns, regularities and exceptions. The dataset was extracted from the World Factbook's PEOPLE data table. This data table consists of characteristics of 190 countries in terms of nine attribute-value pairs per country. The attributes represent the country's population growth rate (PGR), birth rate (BR), death rate (DR), infant mortality rate (IMR), net migration rate (NMR), fertility rate (Fert), life expectancy (LE), literacy percentage (Lit), and predominant religion (Rel). Each attribute has between 5 and 7 values, with the exception of predominant religion, which has 31.

The following samples of KGL code illustrate some of the language's capabilities. The code describes a plan for determining the relationships between the different attributes. Specifically, it searches for strong relationships in the characterizations of the different concepts, tries to improve knowledge that does not meet desired quality standards, analyzes the makeup of a ruleset, and explores the relationship between two records in the database.

The first section of code (Sample 1) generates an initial rule base by characterizing the classes of each attribute in terms of the other attributes, and then counts how many Population Growth Rate characterizations are "strong" based on three metrics: rules that are satisfied by at least 60% of the training examples of their target class, rules that are satisfied by at least 25 training examples of their target class, and rules that contain at least three conditions that are satisfied by at least 10 positive examples of the target class, and by fewer negative examples than positive examples.

Sample 1

```
open PEOPLE                                   {Select the PEOPLE database}
do CHAR(decision=all, pfile=people1.lrn)      {Characterize all concepts
                                               using parameters specified
                                               in file people1.lrn}
strongPGrules1 = #rules(PGR, %covd >= 60)     {Count the strong PGR rules}
strongPGrules2 = #rules(PGR, num_covd >= 25)  {based on three metrics}
```

```
strongPGrules3 = #rules(PGR,
      num_conds(supp >= 50 and pos > 10) > 2)
print "Number of strong PGR rules: Type 1 = ",
      strongPGrules1, ", Type 2 = ",
      strongPGrules2, ", Type 3 = ", strongPGrules3
```

The learning stage of the above code's execution produced a set of characterizations of the relationships between each concept (attribute class) in the PEOPLE database and the remaining attributes. Each condition in the characterizations is associated with 4 weights: the numbers of positive and negative examples that satisfy the condition, the condition's support level (defined by Pos / (Pos + Neg)), and the condition's commonality level, defined as the percentage of examples of the target class that satisfy the rule. Two examples of the hundreds of generated characteristic descriptions are presented below:

(1) Countries with 1993 Population Growth Rate < 1%: (55 Examples)

		Pos	Neg	Supp	Comm
1.	Birth Rate <= 30 per 1000 population	53	52	50	96
2.	Death Rate = 10 .. 15 per 1000 population	21	21	50	38
3.	Life Expectancy = 60 .. 80 years	54	79	40	98

Number of Examples Covered: 21

Examples: Antigua, Austria, Barbados, Belgium, Croatia, Czech Republic, Denmark, Estonia, Germany, Hungary, Latvia, Lithuania, Moldova, Monaco, Norway, Romania, Russia, St. Kitts, Sweden, Ukraine, United Kingdom

(2) Countries with 1993 Population Growth Rate > 4%: (4 Examples)

		Pos	Neg	Supp	Comm
1.	Net Migration Rate >= 10 per 1000 population	4	5	44	100
2.	Predominant Religion = Muslim or Buddhist or Indigenous	4	34	10	100
3.	Life Expectancy = 60 .. 80 years	4	117	3	100

Number of Examples Covered: 4

Examples: Cambodia, Kuwait, Mozambique, United Arab Emirates

The first rule does not qualify as a strong Population Growth rule by any of the three criteria defined in the program, while the second rule qualifies under the first criterion— it describes all four of the countries with Population Growth Rate over 4 per 1000 population. The program reports to the user the number of strong Population Growth Rate rules it found of each type:

```
Number of Strong PGR rules: Type 1 = 1, Type 2 = 1, Type 3 = 7
```

Sample 2 determines whether the characterization of Fertility Rate is too complex, based on how many conditions are present. If it is too complex, the program will select the four best attributes for characterizing Fertility Rate (based on maximum discrimination), project the data table on these attributes into data table PEOPLE2, and generate new characterizations from only these attributes.

Sample 2

```
if #conditions(Fert) > 150                  {Is Fert ruleset too}
begin                                       {complex?}
  do SELECT(attributes, decision=Fert,      {find 4 best independent}
      thresh=4, out=PEOPLE2, criterion=max) {attributes}
do CHAR(pfile=people1.lrn, decision=Fert)   {recharacterize}
  end
```

The program determined that the ruleset for Fertility Rate was too complex based on the specified criterion (the total number of conditions present was 157 in the initial characterization), so the best 4 attributes for learning Fertility Rate concepts were selected and new characterizations were built. The system reported the results of this selection of attributes to the user:

```
Selection of best attributes from PEOPLE for concept Fert --
Attributes chosen:
      Birth Rate
      Predominant Religion
      Life Expectancy
      Death Rate
```

After the new ruleset was learned based only on these attributes, the number of conditions in the characterization of Fertility Rate had been reduced to 120.

In Sample 3 the KGL program inspects the Infant Mortality Rate ruleset for spurious rules (defined as rules covering fewer than 10% of the training examples in their class). If more than 10 such rules exist, relearn the ruleset in discriminant mode with a more stringent search (scope=5) and report the change in the number of values referenced in the IMR ruleset.

Sample 3

```
spuriousIMrules = #rules(IMR, %covd < 10)    {Check for spurious Infant
                                              Mortality rules}
if spuriousIMrules > 10                      {If too many exist}
  begin
  print "Relearning IMR in Discriminant
      mode"
  print "Old # of references:",              {note prior number of}
      #references(IMR)                       {references in the ruleset}
  do DIFF(decision=IMR, scope=5)             {learn discriminant rules}
  print "New # of references:",              {update number of}
      #references(IMR)                       {references}
  end
else print "Characteristic rules             {otherwise inform user}
      adequate for IMR"
```

The program found an unacceptable number of spurious rules for Infant Mortality Rate. It reported this fact and then relearned the rules as as requested:

```
Relearning Infant Mortality in Discriminant mode
Old number of references:  591
New Number of References:  326
```

In Sample 4, the KGL program is asked to analyze the Life Expectancy ruleset in two regards—the number of conditions in the individual rules and the number of conditions in the ruleset with positive/negative coverage ratios exceeding various given thresholds.

Sample 4

```
for i = 1 to 6                               {For each value of i from 1}
  begin                                      {to 6}
  print "Number of LE rules with", i,        {count and display number}
       "conditions = ",                      {of rules with that many}
      #rules(LE, num_conds() = i)            {conditions}
  print "Number of LE conditions with P/N    {and number of conditions}
      ratio of at least", i, ":1 =",         {with a pos/neg ratio}
      #conditions(LE, supp = 100 or          {exceeding i:1}
      pos / neg >= i)                        {The support=100 condition}
  end                                        {avoids divide-by-0 trouble}
```

The analysis of the Life Expectancy (LE) rules indicates that most have between 3 and 5 conditions, and that the drop in number of conditions exceeding a given positive/negative ratio appears to approximate a smooth curve:

```
Number of LE Rules with 1 conditions = 0
Number of LE Conditions with P/N ratio of at least 1:1 = 25
Number of LE Rules with 2 conditions = 1
Number of LE Conditions with P/N ratio of at least 2:1 = 10
Number of LE Rules with 3 conditions = 10
Number of LE Conditions with P/N ratio of at least 3:1 = 5
Number of LE Rules with 4 conditions = 5
Number of LE Conditions with P/N ratio of at least 4:1 = 1
Number of LE Rules with 5 conditions = 7
Number of LE Conditions with P/N ratio of at least 5:1 = 1
Number of LE Rules with 6 conditions = 2
Number of LE Conditions with P/N ratio of at least 6:1 = 1
```

Sample 5 is based on an earlier experiment with INLEN (Kaufman, 1994), in which rules were learned from countries selected as representative of their regions, and then other countries were tested against those rules to see how well they fit the patterns found for their region, and for other regions. One of the reported results was that Canada seemed to resemble more closely the developed countries of East Asia than it did the United States.

One effect of a ruleset for a class consisting of multiple rules is that two examples that satisfy a particular rule are likely to have more in common than two examples in the same class that require different rules to characterize them (the partition of the elements of a class by a set of rules can be seen as a form of clustering). Thus, one way of measuring whether the similarities between two countries such as Canada and Japan (who share the same class in 6 of the 9 PEOPLE rulesets, i.e., they share the same value for 6 of the 9 attributes) are superficial or very significant would be to count the rules that cover both countries, as opposed to the rules that only cover one of them. This is very easy to do in KGL; the sample below counts the rules satisfied by both Canada and Japan, and the rules satisfied by both Canada and the Republic of Korea (with whom Canada shares the same class in 8 of the 9 PEOPLE rulesets).

Sample 5

```
total_Canada = 0                             {initialize counters}
num_match_Japan = 0
num_match_Korea = 0
for i = 0 to 8                               {For each decision}
  begin                                      {attribute …}
  forall rules(i, covers("Canada"))          {… For each rule covering}
begin                                        {Canada}
    total_Canada = total_Canada + 1          {update total counter}
    if covers("Japan")                       {if Japan is also covered}
      begin
      num_match_Japan = num_match_Japan + 1  {update match counter}
      print var_name(i), "matches Japan"     {report to user}
      end
    if covers("Republic_Korea")              {if South Korea is also}
      begin                                  {covered}
      num_match_Korea = num_match_Korea + 1  {update match counter}
      print var_name(i), "matches Korea"     {report to user}
      end
    end
  end
print ("Total matching: Japan = ",
      num_match_Japan, ", Korea = ",
      num_match_Korea                        {summarize}
```

The output from Sample 5 is summarized by this line:

```
Total matching: Japan = 1, Korea = 7
```

It is surprising that out of all the generated rules satisfied by Canada, only one (a characterization of countries with birth rates between 10 and 20 per 1000 population) is also satisfied by Japan. Meanwhile, only once when Canada and the Republic of Korea were in the same class did the rule generation operator not classify them together under a single rule. The result indicates that the demographic commonalities between those two countries are very strong in spite of their geographic differences.

6 SUMMARY AND FUTURE WORK

The presented methodology supports the planning of learning experiments involving a variety of learning and knowledge processing programs. This is accomplished through the employment of a high-level language, KGL, which can access operators corresponding to these programs. KGL allows the user to write simple programs that can execute very complex learning and knowledge processing tasks.

KGL is modeled in part after common programming languages, and in part after database query languages such as SQL. A KGL user can access tools for such tasks as inductive learning from examples, feature selection, knowledge testing, and prediction of missing data based on provided or discovered knowledge. The language enables both the presentation of results to the user and the use of these results as a springboard to further discovery tasks. It allows the user easy interaction and communication with the program of the needs relating to the current application. It is not difficult to enhance the language by adding new operators or functions, and that is being done as new needs are recognized.

The KGL methodology takes a middle road in terms of level of system autonomy—it does not fully automate the process, nor does it require the individual invocation of every operator by the user (as is commonly the case in machine learning practice, including in earlier versions of INLEN). Instead, the high-level language allows the user to pass general guidelines to the program which can then execute the appropriate operators. Hence, this approach produces a real synergy between the user and the system in generating knowledge.

In its current form, KGL has several limitations. The presented version is a very new, and still unpolished solution, designed less for elegance than for the chance to prove its feasibility. It is planned that the implementation of this methodology will be refined and advanced in the future.

The greatest strength of the methodology is its ability to integrate and invoke diverse learning and knowledge processing operators and create a powerful environment for experiments in searching for solutions to practical learning and discovery tasks. It is designed to serve as a novel and useful solution for implementing multistrategy learning capabilities. The experiments done so far are very promising, and indicate that the proposed methodology can be of very high practical utility.

REFERENCES

Han, J., Fu, Y., Wang, W., Chiang, J., Gong, W., Koperski, K., Li, D., Lu, Y., Rajan, A., Stefanovic, N., Xia, B. and Zaiane, O.R., "DBMiner: A System for Mining Knowledge in Large Relational Databases," *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, Portland, OR, pp. 250-255, 1996.

Kaufman, K., "Comparing International Development Patterns Using Multi-Operator Learning and Discovery Tools," *Proceedings of AAAI-94 Workshop on Knowledge Discovery in Databases*, Seattle, WA, pp. 431-440, 1994.

Kaufman, K. and Michalski, R.S., "A Method for Reasoning with Structured and Continuous Attributes in the INLEN-2 Knowledge Discovery System," *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, OR, pp. 232-237, 1996.

Kaufman, K., Michalski, R.S. and Kerschberg, L., "Mining for Knowledge in Data: Goals and General Description of the INLEN System," In Piatetsky-Shapiro, G. and Frawley, W.J. (Eds.), *Knowledge Discovery in Databases*, Menlo Park, CA: AAAI Press, pp. 449-462, 1991.

Michalski, R.S., "Discovering Classification Rules Using Variable-Valued Logic System VL1," *Proceedings of the Third International Joint Conference on Artificial Intelligence*, Stanford, CA, pp. 162-172, 1973.

Michalski, R.S., "A Theory and Methodology of Inductive Learning," In Michalski, R.S. Carbonell, J.G. and Mitchell, T.M. (eds.), *Machine Learning: An Artificial Intelligence Approach*, Palo Alto: Tioga Publishing, pp. 83-129, 1983.

Michalski, R.S., "Inferential Theory of Learning: Developing Foundations for Multistrategy Learning," In Michalski, R.S. and Tecuci, G. (eds.), *Machine Learning: A Multistrategy Approach*, San Francisco: Morgan Kaufmann, pp. 3-61, 1994.

Michalski, R.S. and Imam, I.F., "Learning Problem-Oriented Decision Structures from Decision Rules: The AQDT-2 System," in Ras, Z.W. and Zemankova, M. (eds.), *Lecture Notes in Artificial Intelligence: Methodologies for Intelligent Systems — Proceedings of the 8th International Symposium on Methodologies for Intelligent Systems (ISMIS-94)*, Springer-Verlag, 1994.

Michalski, R.S. and Imam, I.F., "On Learning Decision Structures", *Fundamenta Matematicae*, 31:1, pp. 49-64, July, 1997.

Michalski, R.S., Kerschberg, L., Kaufman, K. and Ribeiro, J., "Mining for Knowledge in Databases: The INLEN Architecture, Initial Implementation and First Results," *Journal of Intelligent Information Systems: Integrating AI and Database Technologies*, 1:1, pp. 85-113, August, 1992.

Michalski, R.S. and Tecuci, G. (eds.), *Proceedings of the First International Workshop on Multistrategy Learning*, Harpers Ferry, WV, November, 1991.

Michalski, R.S. and Tecuci, G. (eds.), *Proceedings of the Second International Workshop on Multistrategy Learning*, Harpers Ferry, WV, May, 1993.

Michalski, R.S. and Wnek, J. (eds.), *Proceedings of the Third International Workshop on Multistrategy Learning*, Harpers Ferry, WV, May, 1996.

Ribeiro, J., Kaufman, K. and Kerschberg, L., “Knowledge Discovery from Multiple Databases,” *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Montreal PQ, pp. 240-245, 1995.

Wrobel, S., Wettschereck, D., Sommer, E. and Emde, W., “Extensibility in Data Mining Systems,” *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, Portland, OR, pp. 214-219, 1996.