

3-2002

An intelligent middleware for linear correlation discovery

Cecil CHUA


Roger Hsiang-Li CHIANG

Ee Peng LIM

Singapore Management University, eplim@smu.edu.sg

DOI: [https://doi.org/10.1016/S0167-9236\(01\)00127-0](https://doi.org/10.1016/S0167-9236(01)00127-0)

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

Citation

CHUA, Cecil; CHIANG, Roger Hsiang-Li; and LIM, Ee Peng. An intelligent middleware for linear correlation discovery. (2002). *Decision Support Systems*. 32, (4), 313-326. Research Collection School Of Information Systems.

Available at: https://ink.library.smu.edu.sg/sis_research/57

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

An intelligent middleware for linear correlation discovery

Cecil Eng Huang Chua^a, Roger H.L. Chiang^{b,*}, Ee-Peng Lim^c

^a*J. Mack Robinson College of Business, Georgia State University, Atlanta, GA 30303, USA*

^b*College of Business Administration, University of Cincinnati, Cincinnati, OH 45221, USA*

^c*Center for Advanced Information Systems, School of Applied Science, Nanyang Technological University, Singapore 639798, Singapore*

Received 2 August 2000; accepted 1 June 2001

Abstract

Although it is widely accepted that research from data mining, knowledge discovery, and data warehousing should be synthesized, little research addresses the integration of existing data management and analysis software. We develop an intelligent middleware that facilitates linear correlation discovery, the discovery of associations between attributes and attribute groups. This middleware integrates data management and data analysis tools to improve traditional data analysis in three perspectives: (1) identify appropriate linear correlation functions to perform based on the semantics of a data set; (2) execute appropriate functions contained in the data analysis packages; and (3) derive useful knowledge from data analysis. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Software integration; Middleware; Data mining; Knowledge discovery

1. Introduction

Much recent research has focused on database integration [21], data warehousing [13], and data mining and knowledge discovery [6]. All of these research areas have attempted to address an emerging business need: the exploitation of large amounts of data to derive useful information (i.e. obtain business intelligence).

Many businesses own separate software for data definition, data manipulation, and data analysis. For

example, while business data may be stored in a Microsoft Access database, necessary data analysis functions are contained in heterogeneous data analysis packages such as SPSS/Base [10] or SAS [14]. These businesses face three major problems in leveraging on their existing software for knowledge discovery:

- **Scarce data analysis expertise.** Few users have formal training with advanced data analysis methods such as data mining and on-line analytical processing (OLAP) [4], and data management tools such as data warehouses. Experienced analysts continue to be in short supply, especially since the growth in data to be analyzed continues to outpace the number of new trained data analysts entering the market [9].

- **Affordability of integrated tools.** While integrated prototype and commercial database/data analysis systems do exist (e.g. Refs. [2,8,13,25]), many companies are either unable or unwilling to adapt

* Corresponding author. Tel.: +1-513-556-7086; fax: +1-513-556-4891.

E-mail addresses: cchua@gsu.edu (C.E.H. Chua), Roger.Chiang@uc.edu (R.H.L. Chiang), aseplim@ntu.edu.sg (E.-P. Lim).

these products due to technique feasibility, economic feasibility, operational feasibility, and many other reasons.

- **Lack of a well accepted data analysis communication standard.** To transfer data from a database to a data analysis package, it is necessary to create an export file in a format the data analysis package understands, and manually import it to the data analysis package. Results obtained from a data analysis package must also be manually keyed into the database. Furthermore, standard interfaces for data analysis tools do not exist. For example, statistical packages adopt different languages (e.g. SAS and SPSS both employ different commands to perform a linear regression), and generate output in different formats. Data analyst often do not have the time, knowledge or ability to integrate their databases with their disparate data analysis packages.

The transfer of information between database and data analysis packages is not only a tedious and task intensive process, but also an error prone one. Data analysis is fundamentally iterative. Knowledge obtained from one analysis is used to guide a second, and then a third analysis. Each time an analyst must export data to an analysis package, or enter results into a database manually, there is a chance that the analyst commits a mistake. As the analyst performs the same tasks repeatedly, the likelihood that the analyst commits an error increases. To reduce the amount of inaccuracy, we propose to simplify or automate the integration between databases and data analysis systems.

Therefore, there is an emerging and urgent need to develop an intelligent system to seamlessly integrate existing data management and data analysis tools to allow the business to maximize the use of information.

In our research, we aim to develop an intelligent middleware between databases and data analysis packages. Because data analysis is a very broad topic, we restrict the scope of our research to linear correlation discovery. Linear correlation discovery refers to the discovery of associations between attributes and attribute groups (sets of attributes). For example, a store manager wants to know whether alcohol sales are directly related to temperature and consumer profile (e.g. gender, age). While our work concentrates on linear correlation discovery, it can be generalized to other forms of data analysis such as market basket

analysis [1], comparisons of groups, or prediction. Our research does not attempt to discover non-linear associations, or associations between data with a time-dependent component. Such analyses often require techniques more sophisticated than that incorporated in this research.

1.1. Research objectives

The middleware is developed to accomplish the following objectives:

- **Automatic identification of appropriate functions.** In data analysis, the appropriate function to apply is determined based on knowledge about the kind of analysis to perform, and the characteristics of the data to analyze. For example, when one measures associations between nominal (i.e. unordered) attributes, it is best to use a contingency table. However, associations between ordinal (i.e. ranked) attributes are measured using Spearman's Rho, or Kendall's Tau.

Most existing data analysis package require that the users determine the data set and the function for the analysis. This adds a cognitive burden to the user, because the user must (in a single step) identify not only the kind of analysis to perform, but also the function that best performs the task. Novice users are often unable to perform this task correctly. Our middleware identifies the appropriate correlation function to apply based on the characteristics of data to be analysed, thereby relieving the user from this task.

- **Standardized access to data analysis packages.** No standard language currently exists among data analysis packages. For example, both SPSS and SAS use different commands to execute a linear regression. As migration to more sophisticated data analysis packages requires expensive retraining, users are often locked into one particular package.

Our middleware is developed to translate users' data analysis requests into the commands of the target data analysis package. Thus, users do not have to learn the appropriate command syntax to express their data analysis requirements. Furthermore, users are not constrained by any one package and can apply different packages for their analysis task.

- **Automatic extraction and interpretation of data analysis results.** The functions of most data analysis packages produce voluminous amounts of information, most of it irrelevant to the specific data

analysis task performed. Furthermore, different data analysis packages report the same information in different ways. This results in additional learning and human information processing costs, as the user must learn how to extract and interpret results from different packages. Our middleware scans the data analysis output, and extracts only the relevant information.

1.2. Definitions

Special terms used in this paper are defined as follows:

- **Function:** A function refers to a theoretic construct used to perform data analysis. For example, linear regression and classification trees are functions.
- **Algorithm:** An algorithm is a specific implementation of the function. For example, the CART [3], and QUEST [16] algorithms are two implementations of classification trees. Similarly, a linear regression can be implemented using stochastic approximation, or through a matrix minimization approach.
- **Package:** A package is a software that is widely available and adaptable to many situations. Microsoft Access and SPSS are packages. Customized systems specific to a business are not packages.

2. Intelligent middleware development

The intelligent middleware performs the following tasks to achieve the research objective:

- **Store expert knowledge concerning data analysis.** The middleware stores knowledge concerning statistical function selection, data analysis package execution, and data analysis output interpretation as production rules. This enables users to perform effective data analysis with only minimal training. These rules can be easily adapted and revised to suit an organization's specific data analysis requirements.
- **Derive schema and instance characteristics from the data.** The middleware employs schema information, such as the data type, and instance information, such as the variation in instance length, to determine the kind of data analysis functions that is appropriate for the data set. This enables the user to focus on what to analyze (e.g. measure the association between Race and Occupation) instead of how to

analyze it (e.g. Goodman and Kruskal's Lambda is an appropriate function for this analysis).

- **Analyze data and report results.** The middleware automatically exports data from the database system to the data analysis package and executes the relevant function in the data analysis package. It then reads output from the data analysis package and extracts relevant information from it. Thus the user does not have to become familiar with the output from the data analysis package.

Thus, all interactions and tasks between the database and data analysis packages are transparent to the user. The user need not know the implementation details of the target packages, nor the name of the function being used. The user simply specifies the analysis requirements of the target data (attributes).

The middleware was developed primarily to support novice users in low-budget knowledge discovery. As a result, it does not incorporate sophisticated database management, or data analysis algorithms, which are assumed to be available through other softwares (e.g. databases and data analysis software). The current implementation uses the Visual Basic programming language. It currently couples a Microsoft Access database with the SPSS statistical package [10], the Nevprop3 neural network package¹, and the Quest classification tree package [16].

The middleware implements the discovery process presented in Fig. 1.

2.1. System architecture

Fig. 2 presents the four components of the middleware, which are:

- **Central Control Unit:** The Central Control Unit (CCU) integrates the other components, and serves as the primary interface between the user and the integrated software packages. It also evaluates results produced by the data analysis package and determines which results are valid and interesting.

The Central Control Unit calls the appropriate component of the middleware to perform specified tasks. For example, the Central Control Unit repeat-

¹ NevProp, developed by Phil H. Goodman, is freely distributed under the GNU public license and can be downloaded from <http://www.scs.unr.edu/nevprop/>.

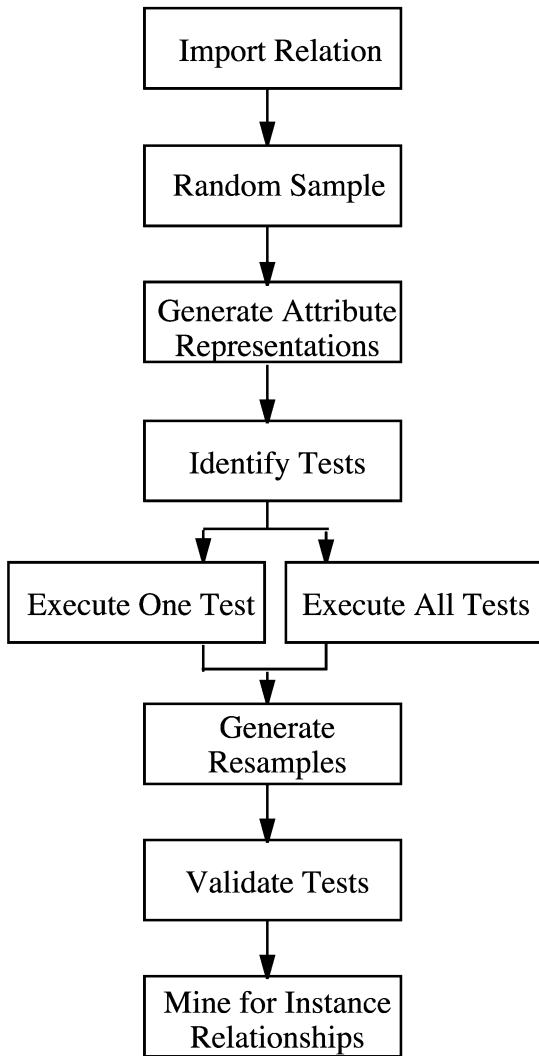


Fig. 1. The linear correlation discovery process.

edly polls the data analysis package to determine when output has been generated. Upon generation, the Central Control Unit calls the function coupler to read the output data and extract appropriate results.

In addition, the Central Control Unit uses information generated by the other components to determine whether a result is interesting and valid. For example, when the user executes a linear regression, the Central Control Unit compares the resultant R^2 and p -values against user-specified thresholds to determine whether the regression was both statistically significant and practically important. The process sequence of the

middleware (which captures the essence of the Central Control Unit) is shown in Fig. 1.

- **Selection assistant:** The selection assistant is responsible for selecting the appropriate data analysis functions. To achieve this goal, the selection assistant performs three principal tasks. First, it uses the database schema and instance information to classify attributes and attribute groups. Second, it determines the classification of attribute groups (i.e. sets of attributes) from the classification of the individual attributes. Third, it uses the classifications to identify functions in the data analysis packages that are appropriate for analyzing the attributes groups. These tasks are described more fully in Section 3.

- **Data analysis interface:** The data analysis interface determines the appropriate commands and data format for invoking the data analysis package and interprets results produced by the package. It is described in detail in Section 4.

- **Database interface:** The database interface enables the import of data from a database for data analysis. Data is imported in one of two ways: (1) through ODBC, or (2) through a well-accepted intermediate database format (e.g. DBase IV). Since interfaces between databases such as SQL, RDA, ODBC, and JDBC are well known and accepted, we do not discuss the database interface in this paper.

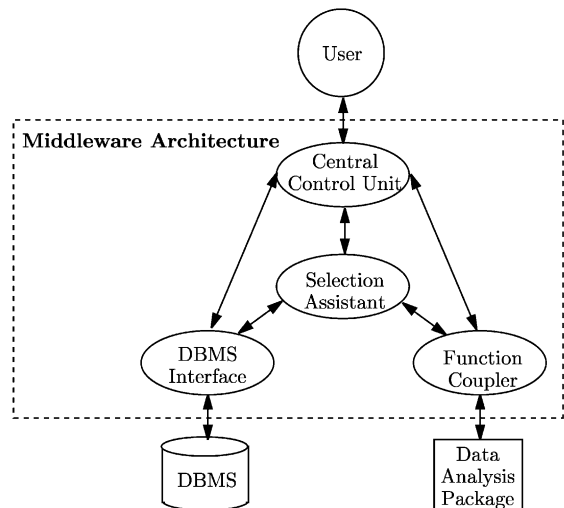


Fig. 2. Middleware system architecture.

3. Attribute classification

The selection assistant adopts a three-stage selection process. In the first stage, the attributes are classified according to the analysis functions that can be appropriately applied. Schema and instance information are used to obtain these classes. In many cases, functions on attributes are not applicable to their attribute groups (i.e. sets of attributes). For example, while a Pearson's coefficient of determination can be applied to determine the association between Salary and Years_of_Service, it cannot be applied to the association between {Salary, No_of_Awards} and {Years_of_Service, Performance}. A canonical correlation would be more appropriate in this case. Thus, in the second stage, the attributes are combined into attribute groups. Each group is then classified based on the combined classes of all its component attributes. In the third stage, the appropriate analysis function for the attributes is automatically selected, based not only on the analysis purpose, and attribute semantics but also on the availability of the function in the data analysis package. For example, while the Box–Cox function is preferred for discovering relationships of the form $\sum_{i=1}^n \beta_i x_i^{\gamma_i} + C + \epsilon = y$, it is not available in the base SPSS statistical package. When coupled with SPSS, the middleware substitutes the SPSS curve-fit function in its stead.

In this section, we describe how attribute classification is implemented in the middleware. We first discuss the possible classification schemes. We then discuss the mechanisms by which the middleware obtains schema, domain, and instance information. Third, we describe how heuristic rules can be embedded in the middleware to perform attribute classification. Finally we describe how attribute groups (i.e. sets of attributes) are classified.

3.1. Attribute classification schemes

The selection assistant performs classification of attributes using its heuristic rules. The heuristic rules are adapted from the classification schemes proposed in [5,12,20]. We briefly describe the adopted schemes below. Attributes were categorized into two groups, STRING, or NUMBER in Ref. [12]. STRING attributes were those which could not be analyzed using algebraic functions such as addition, subtraction, multiplication and division.

Attributes were categorized based on their measurement scales (i.e. NOMINAL, ORDINAL, INTERVAL) in Ref. [20]. NOMINAL attributes have distinct values. For example in Religion, a 'Muslim' is distinctly different from a 'Christian'. ORDINAL attributes have ranked values. For example, Military_Rank is ORDINAL data, since 'Lieutenant' < 'Captain' < 'Mayor'. The values of INTERVAL attributes have a distance. For example, Height is INTERVAL, since the difference between 3.5 and 4.0 in. is the same as the difference between 4.0 and 4.5 in.

The measurement scales in Ref. [20] were further subdivided into DICHOTOMOUS, CATEGORICAL, ORDINAL, DATE, and NUMBER in Ref. [5]. NOMINAL attributes were segregated into DICHOTOMOUS and CATEGORICAL attributes, since DICHOTOMOUS attributes (e.g. Gender) are always bi-valued. Some functions can exploit bi-valued NOMINAL attributes. For example, while it is possible to determine the mean and standard deviation on Yes/No opinion polls (DICHOTOMOUS attribute), it is not possible to determine the mean and standard deviation of Race. INTERVAL attributes were subdivided into DATE and NUMBER, as some operations on NUMBER (e.g. multiplication) were not valid on DATES. This is the default classification scheme used in the selection assistant.

3.2. Derivation of schema information

Three kinds of data are used by the selection assistant to classify the attributes, schema information, domain knowledge, and the attribute instances. Schema information is obtained through the database data dictionary. A set of functions to obtain schema information (e.g. data type, maximum attribute length, record count, etc.) have been defined in the selection assistant. Most relational databases have data dictionaries structured in a relational format. Thus, all meta-schema information can be retrieved by some standard relational queries.

Furthermore, domain information can be obtained from the user. For example, a user may be asked to identify foreign keys if those are not stored as part of the data dictionary.

3.3. Derivation of instance information

Instance information is obtained through *interface subroutines*. The interface subroutines serve as a

bridge between the selection assistant and the database manipulation language. While the names of the interface subroutines are standardized, their implementation on database platforms will differ depending on the native procedural data manipulation language (e.g. Visual Basic, XBase, Java). Some examples of interface subroutines include CountDistinct, which counts the number of distinct instances in an attribute, LengthVary, which measures the variance (i.e. spread) in the length of the instance, and InDict, which calculates the percentage of distinct instances found in a dictionary. An extended version of the WinEdt English spellcheck dictionary² is used as the reference dictionary for InDict.

All interface subroutines accept three kinds of arguments (data types).

- **A constant.** Constants are identified to the interface functions either as values (e.g. '26'), or as the name of an attribute in the data dictionary (e.g. ('CONSTANTS', 'MAX_DISTINCT_ORDINAL')). The first kind of constant is always passed to the interface function as a single value. The second is passed as a table name/attribute name pair. Thus, the interface subroutine has no difficulty differentiating them.

- **The name of an attribute:** This is useful for accessing schema information that the interface subroutine may require. For example, the InDict subroutine performs an SQL query to match instances in the attribute with dictionary words, and requires the attribute name to perform the query.

- **The instances of the attribute:** Instances are passed to the subroutine in one of three different forms. In the first form, called *distinct instance passing*, only the distinct instances are passed to the interface subroutine. This form is used if the interface subroutine does not require knowledge of the variation in instances, and reduces processing time. For example, if the attribute Gender has 20 instances of 'Male', and 24 instances of 'Female', only one instance of 'Male' and one instance of 'Female' is passed to the interface subroutine using distinct instance passing. In the second form, *count instance passing*, instances are passed with a count of their occurrence frequency. For example, under count instance passing, the pair ('Male', 20),

('Female', 24) would be passed to the interface function. This is useful for interface subroutines where the frequency of the occurrences is important, but not their ordering. For example, an interface subroutine that calculates the variance in length between the instances would be passed instance information in this form. Finally, in *full instance passing*, the instances are passed to the interface subroutine in their original state. For example, all 20 instances of 'Male', and all 24 instances of 'Female' are passed to the subroutine. This kind of instance passing is only used for subroutines that must identify patterns based on instance order. An example of such an interface subroutine is one that analyzes the database update logs.

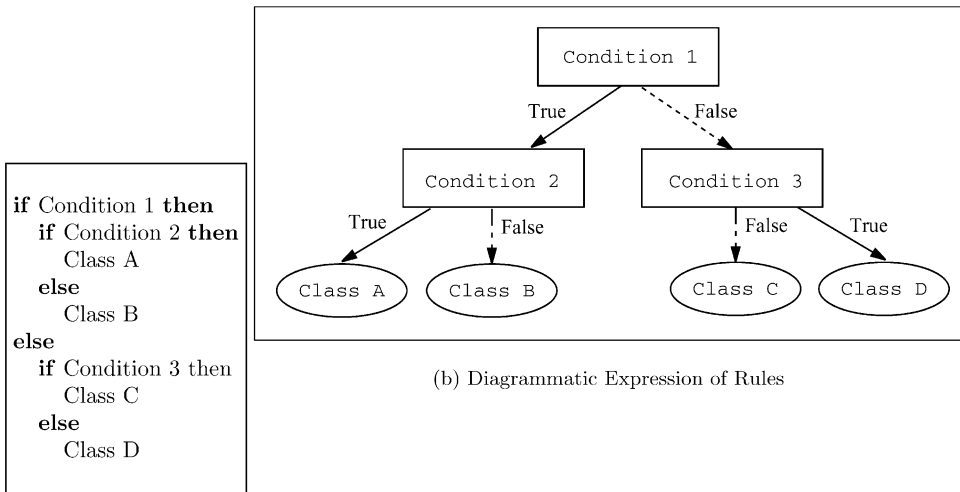
3.4. Rule embedding

We developed a user customizable rule base to classify the attributes based on schema and instance information. These statements (i.e. rules) are represented as a set of negative statements as **if–then is not**. The antecedent (i.e. **if** component) evaluates the schema, domain and instance information. The consequent (**then is not** component) identifies classes which are inappropriate for that attribute. Our representational choice (i.e. negative instead of positive statements) [18] is based on performance (i.e. speed) considerations [11].

Each rule contributes through a divide-and-conquer strategy as illustrated in Fig. 3a and b. Each condition refers to a query on the domain, database schema, or attribute instances. In the example, Condition 1 partitions the solution space into {Class A, Class B} and {Class C, Class D}, Condition 2 partitions the remaining solution space into {Class A} and {Class B}, etc. Note that Fig. 3b is a dichotomy because an **if–then–else** rule is by its nature dichotomous.

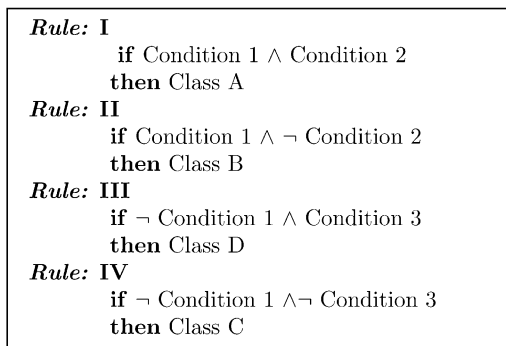
It was not feasible to include both positive and negative statements in the rule base, because of the different ways these two kinds of statements are interpreted. Each positive statement that evaluates to true would add to the set of possible classes. Negative statements subtract from it. If both positive and negative statements were included in the rule base, it would be difficult to clearly and unambiguously determine the appropriate class for an attribute group, as a class removed by one rule could possibly be reinserted by a following rule.

² WinEdt, created by Alexander Simonic, is a shareware text editor for \LaTeX . It is available from <http://www.winedt.com>.

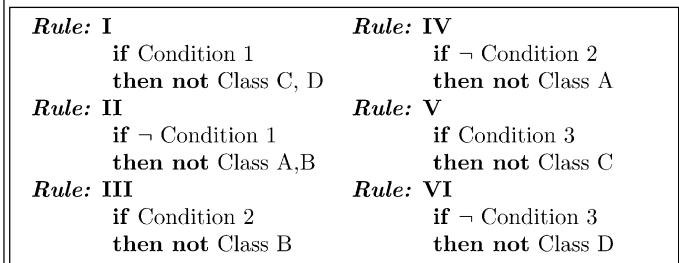


(a) Syntactic Expression of Rules

(b) Diagrammatic Expression of Rules



(c) Rules using Assertions



(d) Rules using Negations

Fig. 3. Divide and conquer through rules.

As queries performed on instances are computationally expensive, a rule representation based on negative statements (e.g. Fig. 3d) was selected as it requires less time to process than one based on positive statements [11] (e.g. Fig. 3c). For example, in Fig. 3c, under the worst case, Condition 1 is called five times. In Fig. 3d, it is evaluated at most twice. Later in this paper, we show how to further reduce the number of statement evaluations using a **converse flag**. A converse flag is similar to the ‘else’ statement in traditional *if-then* clauses as it identifies inappropriate classes when the antecedent is false.

Problems with negative statements: When antecedents are expressed using only negative statements, the inference engine is only able to reduce the classification search space. It may not be able to conclusively classify an attribute. For example, assume that Conditions 1 and 2 are true. When all the rules in Fig. 3d are evaluated, the system can only determine that classes B, C, and D are inappropriate. It cannot conclude that class A is appropriate.

To resolve this problem, an ordering is established on the rules [24]. Some rules are assigned a **termination clause**. The rule with the least priority is also assumed to

have a termination clause (regardless of whether it was given one), as the inference engine must classify the attribute after all rules have been exhausted. The termination clause is only tested for when the antecedent evaluates to true, and the inference engine reduces the number of possible classes in the consequent. If there is only one remaining possible class, that class is considered as appropriate for that attribute. Thus, the inference engine evaluates each rule in sequence, reducing the number of potential classes for the attribute every time a rule's antecedent evaluates to true. Whenever the inference engine encounters a rule with a true antecedent and a termination clause, the attribute is classified (if it has only one remaining potential class).

Other considerations: Different DBMSes contain different kinds of database meta-schemas. For example, some databases such as Microsoft Access explicitly identify the primary key, while others such as DBase do not. Thus, schema information required by a rule may not be available in the implementation DBMS. The kind of meta-schema information required for rule evaluation is stored along with the rule. Before a rule is invoked, the inference engine tests if the DBMS schema contains the necessary information. A failed test means that the rule is not invoked.

The performance of the inference engine is further improved by combining rules that are converses (i.e. exact opposites). Such rules (e.g. Rules I and II of Fig. 3d) are combined using a **converse flag**. For example, in a classification scheme with five classes {NUMBER, ORDINAL, CATEGORICAL, DATE, and DICHOTOMOUS}, rules such as 'If an attribute has two or less distinct instances, then it cannot be classified as NUMBER, ORDINAL, CATEGORICAL, DATE' and 'If an attribute has three or more distinct instances, then it cannot be classified as DICHOTOMOUS' are combined and evaluated as a single rule.

Example rules: Some of the rules in the selection assistant's rule base are presented below as examples of how the rule-base is structured. Sample output from the attribute classifier is presented in Fig. 4.

Rule 1: ID_Date

Required Information: Data Type

IF The data type of the attribute is of type 'Date'
THEN the attribute is not classified as NUMBER, DICHOTOMOUS, ORDINAL OF CATEGORICAL

Termination Clause: Terminate

Converse Flag: No converse flag

Field	presentation
AGE	Number(1)
CAPITALGAI	Number(1)
CAPITALLOS	Number(1)
CLASS	Dichotomou
COUNTRY	Categorical
EDUC	Ordinal
EDUCYRS	Categorical
FNLWEIGHT	Number(1)
HRS_WEEK	Number(1)
ID	Not Applicat
MARITAL	Categorical
OCCUP	Ordinal
RACE	Categorical

Record: 5 of

Fig. 4. Sample output from the attribute classifier.

Rule 2: ID_OrdCat

Required Information: Attribute Length, Number of Instances

IF The maximum length of the attribute is less than or equal to the user-defined constant 'LENGTH' and the maximum number of distinct instances is less than or equal to the user-defined constant 'INSTANCE'

THEN the attribute is not classified as NUMBER OF DATE

Termination Clause: Do not terminate

Converse Flag: Has a converse flag

Rule 3: ID_StringNotNumber

Required Information: Data Type, Instance

IF The date type of the attribute is of type 'String' and at least one instance of the attribute contains

the letters ‘A’...‘Z’ or ‘a’...‘z’ and at least one instance has a length different from the length of all other instances

THEN the attribute is not classified as NUMBER or DATE

Termination Clause: Do not terminate

Converse Flag: No converse flag

Rule 4: ID_Key

Required Information: Candidate Key

IF The attribute is a candidate key

THEN the attribute cannot have the classes {NUMBER, DATE, ORDINAL, CATEGORICAL, DICHOTOMOUS}, i.e. it cannot have a class.

Termination Clause: Terminate

Converse Flag: No converse flag

3.5. Attribute group classification

For multivariate data analysis (i.e. data analyses involving more than two attributes), attribute classes are not sufficient for determining analysis functions, as attribute semantics are not generalizable to attribute groups. For example, while the point biserial correlation coefficient [15] is appropriate for measuring the correlation between Salary and Gender, it cannot be used to measure the correlation between Salary and {Gender, Religion}, since Gender’s distance property is not applicable when it is combined with a CATEGORICAL attribute.

While many multivariate data analysis functions consider the order of attribute groups for analysis (e.g. it is possible to regress {Salary, Age} on Job Performance, but not possible to regress Job Performance on {Salary, Age}), most do not consider the order of the attributes within the attribute groups. For example, the linear regression of (Salary, Age) to Job Performance produces the same result as the linear regression of (Age, Salary) to Job Performance. Thus, attribute group classification is generally both commutative and associative, i.e. the class of the group (A, B) must be the same as the class of the group (B, A). Similarly, the class of the group {A, B, C} is the same irrespective of the order of the attributes.

For classification purposes, these properties reduce the attribute group classification matrix to one where it is possible to derive all attribute group classes by classifying attributes one pair at a time. Attribute groups

Table 1
Sample mapping table

Group 1	Group 2	Resultant group
Number	Number	Number
Number	Date	Date
Date	Date	Date
Number	Ordinal	Number
Ordinal	Ordinal	Categorical
⋮	⋮	⋮

with more than two attributes are considered as a special pair, where each member of the pair captures the ‘class’ of multiple attributes. For example, the attribute group {Salary, Age, Occupation} can be treated as the group {Group 1, Occupation}, where Group 1 is an attribute with the class of {Salary, Age}. As classification is associative, the class of {Group 1, Occupation} is the same as that of {Salary, Group 2}, where Group 2 has the same class as {Age, Occupation}.

Since all classes can be treated in this way, we implement attribute group classification as a three-column mapping table. The first two columns identify the classes of the attributes in the group. The third column identifies the class of the attribute group. The classes of attribute groups that are composed of more than two attributes are discovered by recursively querying the table. A sample of the mapping table (using the classification scheme proposed in Ref. [5]) is presented as Table 1.

4. Function coupler

Once the appropriate functions to apply to the data are known, the middleware must call the data analysis package to execute the functions, and read and interpret the package’s output results. The function coupler provides facilities to perform these tasks.

Most of the data analysis packages (e.g. SPSS, SAS, Minitab) allow batched requests to be submitted following the process illustrated in Fig. 5. In this process, a sequence of data analysis commands, and a data file to be analyzed are inputted into the data analysis package. The package generates an output file with the results obtained through the commands.

Thus, in order to effectively interface with the data analysis package, the function coupler performs the following tasks.

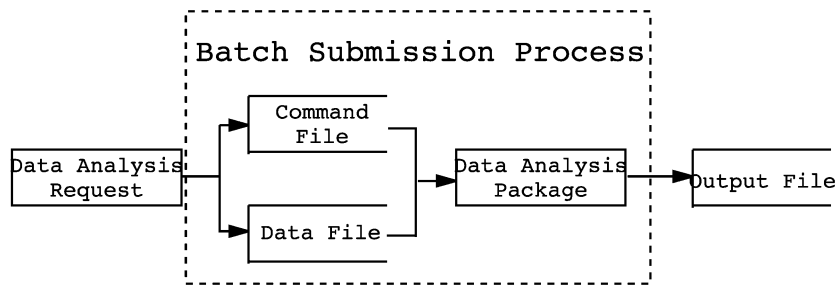


Fig. 5. Batch job processing in statistics packages.

- **Generate a command file.** The command file (typically presented as ASCII text) contains instructions that the data analysis package understands.

- **Prepare a data file.** The data file must be in a format that is readable by the data analysis package. The preparation of the data file not only involves exporting the data in a compatible file format (a trivial problem, since many well accepted interchangeable file formats exist), but also ensuring that data is in a format conducive to data analysis. For example, most data analysis packages are unable to perform analysis on String data, even if all instances are numeric.

- **Call the data analysis package.** Based on the appropriate files to execute, and the locations of these files in the file directory the function coupler calls the data analysis package.

- **Extract relevant results from the data analysis output file.** The data analysis output file is typically semi-structured. The function coupler must have enough knowledge about the statistics package to extract pertinent results from the output data.

The information required for performing each task differs between data analysis packages. For example, the command file for invoking a linear regression in SPSS would differ substantially from that of SAS. However, the required information to perform these tasks are common across the data analysis packages. By identifying this required information, we are able to categorize and organize them to facilitate integration of data analysis packages with database management systems for knowledge discovery. For example, since SPSS and SAS provide linear regression analysis functions, it is critical to obtain and synthesize the syntax of the command for linear regression from the data analysis packages. Currently, we manually enter this information. However, it is our vision that future

versions of data analysis packages will be packaged with an electronic manual for reference by other software.

We propose six kinds of information that must be stored in this electronic manual. These kinds of information are: (1) function identification, (2) function package command, (3) function input, (4) function package format, (5) result package output, and (6) package layout information. We explain each kind of these kinds of information below.

Function identification: Function identification includes a unique code identifying each data analysis function, the common name of that function, and the function's purpose. For example, a "Linear Regression" might have unique ID '001', and is used for 'Correlation', and 'Function estimation' (i.e. determining the functional relationship between data).

Function package command: Function package command information describes the syntax of the function in a particular data analysis package. Each function package command has four parts:

- **Pre-command information:** This describes preliminary instructions to prepare the data analysis package to receive the command. For example, before performing a linear regression, it may be useful to measure the skew of the data, or to randomly sample the data.
- **Static command:** This is the part of the command which remains constant. For example, in SPSS, a linear regression always begins with the command 'Regression'.
- **Variable markers:** These mark locations in the data analysis command which should be substituted for the names of the attributes being analyzed.

- **User preferences:** The implementations of some functions in the data analysis package give users some optional control over how the function is executed. The function coupler assigns defaults to these preferences, but allows users to modify them.

Fig. 6 illustrates the information associated with a command. This command produces a linear regression between {Job Performance, Age} and Salary for SPSS.

Function input: Function input describes the mapping from the attribute group classes to the data analysis functions. For example, given the scheme in Ref. [20], the appropriate function for measuring the association between an attribute group with an INTERVAL class and a single attribute with an INTERVAL class is a linear regression. Likewise, given the scheme in Ref. [12], both attribute groups must have the NUMBER class before a linear regression is performed. Function input information is used by the selection assistant in determining appropriate data analysis functions.

	Sum of Square	df	Mean Square	F	Sig.
Between Groups	1881.508	67	28.082	3.381	.000

However, at other times it might look like:

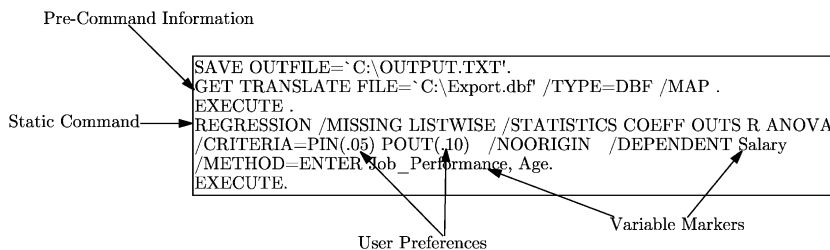
	Sum of Squares	df	Mean Square	F	Sig.
Between Groups (Estimated)	1881.508	67	28.082	3.381	.000

Function package format: Function package format identifies whether attribute groups need to be recorded for the data analysis package. For example, most data analysis packages cannot perform a linear regression on an attribute with a String

data type, even if all the instances have a numeric representation. Function package format allows the function coupler to identify such limitations in the data analysis package and appropriately recode the attribute groups to avoid problems during data analysis.

Result package output: Result package output identifies the location of key results (e.g. *p*, degrees of freedom, etc.) in the output generated by a data analysis package. To facilitate this task, the function coupler includes a primitive scripting language. The scripting language searches ASCII text files for key words, and then extracts other words based on their position relative to those words. For example, one statement in the Result Package Output information on ANOVA for the SPSS package is ‘Degrees of Freedom is the second word after Between Groups’.

The scripting language also includes a selection statement (if then else). This enables the user to specify that a result could appear in multiple locations. For example, a line of statistical text obtained from SPSS might look like:



The function coupler scans the previous output and determines where the degrees of freedom value is. It performs this task by noting if the word ‘(Estimated)’ follows the words ‘Between Groups’.

Fig. 6. Information required of a linear regression in SPSS.

The function coupler also performs simple calculations. This allows the user to derive additional results from those presented. For example, if R^2 is not presented in the data analysis output, the middleware derives it from output produced by a logit function using the likelihood ratio test statistic of the best fit model and the independent best guess model [17]. Standardized data analysis measures are thus obtained irrespective of whether the data analysis package presents those measures in the output.

The function coupler also performs limited decision making. For example, a conflict resolution function is included that extracts R^2 only if the sample size of an analysis is below a particular value. The adjusted R^2 value is otherwise extracted.

4.1. Coupling to data mining packages

While the described function coupler is able to couple the database to traditional statistical packages, additional issues have to be considered to couple it to data mining packages. These issues include:

- **Multiple data sets:** Data mining functions such as neural networks [19], and classification trees [3] use hold-out validation to validate their results. In hold-out validation, the data set is partitioned into a *training set* used for data analysis, and a *test set* used to validate the patterns discovered in the training set. The method for partitioning a data set differs between data analysis packages. While some data analysis packages provide features to perform this partitioning, others (e.g. the neural network Nevprop3) do not. A flag in the Package Layout Information identifies whether the data analysis package performs its own partitioning. If partitioning cannot be performed by the data analysis package, the middleware performs it according to the following well accepted heuristic [23]:

1. If the data set contains more than 3000 tuples, then 1000 tuples are randomly allocated to the test set. The others are allocated to the training set.
2. Otherwise, 1/3 of the tuples are randomly allocated to the test set.

- **Function complexity:** Many data mining function have numerous input parameters. These param-

eters can often be determined by the analyst. A change in any one of these parameters often dramatically changes the nature of the analysis. For example, a fully connected neural network model with three hidden nodes is different from one with four hidden nodes. It is currently not possible to determine the optimal number of nodes or hidden layers for most problem domains [22].

Furthermore, the implementation of many of these data mining functions is not yet standardized. The methods for invoking these data mining functions differ dramatically between software packages. For example, to invoke the classification tree package QUEST [16], 12 different parameters must first be defined (e.g. node size for constructed tree, alpha value for splitting). On the other hand, CART [3] only requires the user to specify the independent and dependent attribute group, if the setting ‘Minimize cost tree regardless of size’ is specified.

The function coupler handles this issue by first generating the data analysis command file with some defaults. The command file is then presented to the user for modification prior to executing the data analysis package.

5. Conclusion and future research

In this paper, we have presented the development of an intelligent middleware that facilitates knowledge discovery. It integrates the data manipulation power of available databases with the data analysis capability of available data analysis packages. The middleware benefits the user in the following way:

- **The user does not need to learn about disparate systems:** Since the middleware ‘knows’ how to operate the database and data analysis packages and read the data analysis output, the user does not need to learn about the packages.

- **Migration costs are lowered:** Since the user does not need to learn the data analysis package, the user can upgrade to a competing product without considering migration cost.

- **It separates conceptual data analysis from implementation:** The user only needs to consider the kind of data analysis (e.g. correlation) that needs to be performed. The middleware selects the appropriate function (e.g. linear regression, Spearman’s

rank order, canonical correlation, Goodman and Kruskal's lambda) for the given task.

While the middleware described demonstrates that it is possible to couple existing databases with data analysis packages, there are many other avenues for future research. Such research includes the development of more sophisticated schemes for classifying attributes for data analysis, and implementation of a standard data analysis language.

While current classification schemes (e.g. Refs. [5,12,20]) enable automated systems to select data analysis functions, much can be done to improve the classification accuracy to these schemes. As one example, classification schemes should consider the distribution of the data as this information is often important for function selection. Thus, measures such as the Pearson's coefficient of skew [15] should be incorporated as they are more useful than the median and mean separately for describing the average value of skewed (i.e. non-normally distributed) data. More sophisticated methods of selecting appropriate functions are necessary to develop sophisticated data analysis packages.

Also, the research community must establish a standard language for data analysis akin SQL for data definition/manipulation. Most current data analysis research focuses on developing new data analysis function [7]. These functions see limited use, as they are implemented in packages with limited functionality. Standardized interface design of data analysis packages will provide many holistic benefits.

Acknowledgements

The authors wish to thank the Editor-in-chief and the anonymous reviewers for their supportive comments on earlier versions of this manuscript.

References

- [1] R. Agrawal, T. Imielinski, A. Swami, Mining association rules between sets of items in large databases, *Proceedings of the 1993 ACM SIGMOD Conference on Management of Data*, 1993, pp. 207–216.
- [2] S.S. Anand, B.W. Scotney, M.G. Tan, S.I. McClean, et al., Designing a kernel for data mining, *IEEE Expert/Intelligent Systems and Their Applications* 12 (2) (March–April 1997) 65–74 [online] <http://inchinn.infnj.ulst.ac.uk/htdocs/mks.html>.
- [3] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, *Classification and Regression Trees*, Wadsworth and Brooks, Pacific Grove, CA, 1984.
- [4] S. Chaudhuri, U. Dayal, An overview of data warehousing and olap technology, *ACM SIGMOD Record* 26 (1) (March 1997) 65–74.
- [5] C. Chua, R.H.L. Chiang, E.-P. Lim, A heuristic method for correlating attribute group pairs in data mining, *International Workshop on Data Warehousing and Data Mining (DWDW'98)*, 1998, pp. 29–40.
- [6] U. Fayyad, R. Uthurusamy, Data mining and knowledge discovery in databases, *Communications of the ACM* 39 (11) (November 1996) 24–26.
- [7] C. Glymour, D. Madigan, D. Pregibon, P. Smyth, Statistical themes and lessons for data mining, *Data Mining and Knowledge Discovery* 1 (1997) 11–28.
- [8] J. Han, J.Y. Chiang, S. Chee, et al., DBMINER: a system for data mining in relational databases and data warehouses, *Proceedings CASCON '97: Meeting of Minds, 1997*, pp. 249–260.
- [9] D.J. Hand, Intelligent data analysis: issues and opportunities, *Proceedings of the Second International Symposium, IDA-97*, 1997, pp. 1–14.
- [10] J. Hedderson, *SPSS/PC+ Made Simple*, Thomson Information/Publishing Group, Belmont, CA, 1991.
- [11] G.J. Holzmann, The model checker SPIN, *IEEE Transactions on Software Engineering* 23 (5) (May 1997) 279–290.
- [12] W. Hou, Extraction and application of statistical relationships in relational databases, *IEEE Transactions on Knowledge and Data Engineering* 8 (6) (December 1996) 939–945.
- [13] W.H. Inmon, *Building the Data Warehouse*, 2nd edn., Wiley, New York, NY, 1996.
- [14] J.A. Jaffe, *Mastering the SAS System*, Van Nostrand Reinhold, New York, NY, 1994.
- [15] R.S. Lehman, *Statistics and Research Design in the Behavioral Sciences*, Wadsworth Publishing, Pacific Grove, CA, 1988.
- [16] W.-Y. Loh, Y.-S. Shih, Split selection methods for classification trees, *Statistica Sinica* 7 (1997) 815–840.
- [17] J.S. Long, *Regression Models for Categorical and Limited Dependent Variables*, SAGE Publications, Thousand Oaks, CA, 1997.
- [18] T. Richards, *Clausal Form Logic: An Introduction to the Logic of Computer Reasoning*, Addison-Wesley, New York, NY, 1989.
- [19] D.E. Rumelhart, J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, Cambridge, MA, 1986.
- [20] P.D. Scott, A.P.M. Coxon, M.H. Hobbs, R.J. Williams, SNOUT: an intelligent assistant for exploratory data analysis, *Principles of Data Mining and Knowledge Discovery, First European Symposium, PKDD '97*, 1997, pp. 189–199.
- [21] A.P. Sheth, S.K. Gala, S.B. Navathe, On automatic reasoning for schema integration, *International Journal of Intelligent and Cooperative Information Systems* 2 (1) (March 1993) 23–50.
- [22] M. Smith, *Neural Networks for Statistical Modeling*, Van Nostrand Reinhold, New York, NY, 1993.
- [23] S.M. Weiss, C.A. Kulikowski, *Computer Systems That Learn*:

Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems, Morgan Kaufmann Publishers, San Mateo, CA, 1991.

- [24] T. Williams, B. Bainbridge, Rule based systems, in: G.A. Ringland, D.A. Deuce (Ed.), Approaches to Knowledge Representation: An Introduction, Wiley, New York, NY, 1998, pp. 101–115, Chap. 5.
- [25] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kauffman, San Francisco, CA, 1999.



representation, application, and reuse of knowledge.

Cecil Eng Huang Chua is a PhD student at Georgia State University. In 1995 he received both a Bachelor of Business Administration in Computer Information Systems and Economics and a Masters Certificate in Telecommunications Management from the University of Miami. He received a Masters of Business by Research from Nanyang Technological University in 2000. His research interests include methods for the re-



His research interests are in data and knowledge management and intelligent systems, particularly in database reverse engineering, database integration, data mining, and common sense reasoning and learning. He is currently on the editorial board of Journal of Database Management and International Journal of Intelligent Systems in Accounting, Finance and Management. His research has been published in a number of international journals including ACM Transactions on Database Systems, Data Base, Data and Knowledge Engineering, Decision Support Systems, and the Journal of Database Administration. He is a member of AAAI, ACM, AIS, IEEE Computer Society, and INFORMS.

Roger Chiang is an Associate Professor of Information Systems at the College of Business Administration, University of Cincinnati. He received his BS degree in Management Science from National Chiao Tung University, Taiwan, MS degrees in Computer Science from Michigan State University and in Business Administration from University of Rochester, and PhD degree in Computers and Information Systems from University of Rochester.



His current research interests include database integration, web warehousing, and digital libraries.

Ee-Peng Lim received the BS (Honours) degree in information systems and computer science from the National University of Singapore, in 1989, and the PhD degree in computer science from the University of Minnesota, Minneapolis, in 1994. Since 1994, he has been on the faculty of the School of Applied Science at the Nanyang Technological University, Singapore, where he founded the Centre for Advanced In-