54

Using partitioned databases for statistical data analysis*

by RUVEN BROOKS
University of Texas Medical Branch
Galveston, Texas

MEERA BLATTNER
University of California, Davis, and
Lawrence Livermore National Laboratory
Livermore, California

ZDZISLAW PAWLAK
Polish Academy of Sciences
Warsaw, Poland

and

EAMON BARRETT
Jaycor Corporation
Washington, D.C.

INTRODUCTION

The statistical analysis of scientific data is a process that can be viewed as consisting of three fundamental phases. First, the observations are recorded. Next, they are encoded into a numeric form suitable for statistical analysis. Finally, the calculations are performed for the particular type of analysis needed for the design of the study. This ordering is, however, only conceptual; in most real studies, the three phases interact and are overlapped. Thus, it may be the case that a preliminary analysis run indicates that a more refined coding scheme is needed or that the coding process reveals deficiencies in the data collection.

A consequence of this interaction is that the data analysis calculations are done repeatedly over time using new coding schemes, new variables, and new selections of grouping of the cases. We propose the structure of a data organization that can result in substantial savings in the amount of calculation needed across these repeated calculations.

OPERATIONS NEEDED IN STATISTICAL ANALYSIS

Case Structure

For most statistical analyses, the data is structured in terms of a row vector consisting of the information relating to a particular observational unit. Typical observational units might include a single individual in a psychological study, a single experimental animal in a medical study, or a single household in a sociological survey. The information recorded in each case can be a mixture of classification variables which indicate to which experimental group the particular case belongs and measurement variables which are the values of measurements for that particular case. (Note that it is possible for some variables to serve both functions.) The set of cases for any particular study or experiment can then be viewed as a rectangular matrix with each case being a row and all the values across cases for a given variable occurring in a single column.

User Operations

In preparing data for statistical analysis and in the course of performing the analyses, users of statistical packages typically perform the following operations:

1. Addition and deletions of cases (row addition and deletion).
2. Updating of variable values within individual cases.
3. Addition and deletion of variables across cases (column addition and deletion).
4. Variablewise transformations, such as multiplying a variable by a constant, adding two variables together to create a new one, etc. Such transformations may also involve conditional operations.
5. Value replacement (recoding) in which all occurrences

*This research was supported in part by NSF Grant No. MCS 77-02470.

of specific values for a particular variable are replaced by new values. This may involve collapsing several values into a single value.

6. Case selection on the basis of predicates applied to variable values. This operation may be used to designate subsets for particular analyses or for the application of variable transformations and recodings.

Statistical Operations

While the variety of possible statistical analyses is extremely large, nearly all of the commonly used analyses begin with a common set of calculations. If X is an $m \times k$ data matrix in which rows are cases and columns are variables, then these calculations can be specified as follows:

1. On the basis of values of the classifier variables, partition the rows of the matrix into sets which are not necessarily disjoint. Let n be the number of rows in a set.
2. For each set, form the following sums:
   a. For each of the j variables (columns), sum x_{ij}, $i = 1 \ldots n$, where n is the number of rows in the partition.
   b. For all pairs of variables j and k, sum $x_{ij}\,{}^{*}x_{ik}$, $i = 1, \ldots, n$. (Note that $j = k$ is included.)

These sums form the basis for further calculations in analysis of variance, analysis of covariance, multiple regression analysis, discriminant analysis, factor analysis, partial correlation, canonical correlation, and multivariate analysis. They are also useful for the calculation of descriptive statistics, such as mean and standard deviation.

In conventional statistical systems such as SPSS[1] and BMD[2], these quantities are calculated *de novo* for each analysis, an extremely costly process if the amount of data is large. The data organization we propose here is intended to save this cost by updating and preserving these quantities across user manipulations.

BASIS FOR THE PROPOSED ORGANIZATION

Partitioned Databases

The organization is based on the work of Z. Pawlak and colleagues (Marek and Pawlak[3], Lipski and Marek[4], Lipski[5]) on partitioned databases. In their terminology, a database consists of a set of objects which have attributes. If our objects are patients participating in a research study, then the patients have certain attributes of interest for data analysis purposes. Typical attributes might include sex, age group, weight, blood pressure, amount of cigarettes smoked, region of the country, etc. A case for analysis purposes then consists of the values of these attributes for some particular object.

In a partitioned database, those attributes which will be used to identify and select cases from the database are designated as *selection* attributes. Typically, though not necessarily, selection attributes have a small number of possible values and are referred to as "nominal" or "categorical" variables. The values of selection attributes are considered to be divided into disjoint descriptors so, for example, sex is divided into male and female, age is divided into ranges of years, etc. Each object must have a value for each selection attribute, though certain values might be used to indicate "unknown" or "data missing." The attributes will be referred to by capital (subscripted or unsubscripted) letters and the descriptors will be referred to by small letters. The selection attributes are ordered in some way so that the list of attributes is $A_1, A_2, \ldots, A_n$. Since each attribute has a list of descriptors, $a, b, c, \ldots, m$ (we need not have the same number of descriptors for each attribute), the description of an object may be made by a vector $(a_1, a_2, \ldots, a_n)$ where a_i is a descriptor in the set of attributes of A_i. We will ordinarily use the designation $a_1 a_2 \ldots a_n$ instead of $(a_1, a_2, \ldots, a_n)$. The $A_1, A_2, \ldots A_n$ are a coordinate system for class descriptors. Observe that each object falls into one and only one class, hence the name *partitioned database*.

An example will illustrate this notation. Suppose that the following selection attributes have the following values:

sex
  a. male
  b. female
age range
  a. 20–30
  b. 31–40
  c. 41–50
  d. over 50
region
  a. north
  b. east
  c. south
  d. west

The class of all women 41–50 years old and living in the north would be designated by bca; there would be a total of 32 possible classes.

The proposed data organization is based on specifying selection attributes which define groupings of cases that are important for analysis purposes and storing with the classes that these attributes define the quantities described earlier. Performing a statistical analysis would then require forming a query which specified the classes that were required, retrieving the summary quantities stored with the class, and performing the necessary statistical calculations on these quantities. For example, if sex and age group had been specified as the only selection attributes, and an analysis of variance were to be performed on blood pressure data for cells defined by sex and age group, sum (x) and sum (x^2) would be retrieved for each of the eight classes of cases that sex and age group define. To collapse across these classes, the sums for the individual classes need only to be added together.

A Query Language for Partitioned Databases

In order to specify which classes of objects are to be used for a statistical analysis, a query language is needed. A formal syntax for a formal language for partitioned databases was thoroughly and rigorously described by Lipski and Marek[4]. The query language described in this section is not the same one described by Lipski[5], however, much of our reasoning follows his proofs closely. Our *query language* is called Q.

Definition: An *alphabet* for Q is:

1. The set of lower case Latin letters, L.
2. The set of lower case Latin letters with a superscripted c, L^c.
3. The set of lower case Latin letters with superscripts g, L^g.
4. The set of lower case Latin letters with superscripts 1, L^1.
5. The symbol, @.

Definition: If the *descriptors* D(B) *of attribute* B are (a,b,c,d), the *complement of x*, x^c, are those descriptors, $D(B) - \{x\}$. The set of descriptors *greater than x*, x^g, are those elements of D(B) that appear after x in the linear ordering of the descriptors (for convenience, we will assume that the descriptors of an attribute always begin with a and continue sequentially through the alphabet) and the set of descriptors *less than x*, x^1, are those elements of D(B) that come before x. @ is a *"don't care" symbol*, used to indicate that an attribute is not of interest for a particular query.

Definition: A *simple term* is a concatenation of n symbols from Q, where n is the number of attributes. A *constant simple term* is a concatenation of n symbols from L while a *variable simple term* contains an occurrence of a symbol not in L. Note that a simple constant term is also a class description as long as $t = a_1\ a_2\ .\ .\ .\ a_n$ and each a_i is in the set of descriptors of A_i.

A *term* is defined recursively as:

1. A simple term.
2. If t and s are terms, then t, $t + s$, t^*s, $t \rightarrow s$ are terms. If we wish we may define a set of formulas over Q that use the equality sign as well as the above Boolean operations and introduce the symbols, TRUE and FALSE. We will omit this part of the query language from the current discussion, since formulas do not increase the complexity of the implementation procedures as presented here. The *value of a term* $v(t)$ is as follows:

1. If t = @@@@@...@, then $v(t)$ is the set of all objects X. If $t = e$, then $v(t) = \phi$.
2. The value of @@...@a@...@, where a is in the ith position, is the set of all objects that have descriptor a of A_i. The value of @@...@x^c @...@ is the set of all objects that have y in the A_i attribute, where y is in the complement of x and x is a descriptor of the ith attribute. Similarly with @@...@x^g @...@ and @@...@x^1 @...@.
3. If t is a simple term and $t = a_1\ a_2 ... a_n$, then $v(t)$ is the set of objects $v(a_1$ @...@$)^*v($@a_2 @...@$)^*...^*v($@@...@$a_n)$.
4. $v(-t) = X - v(t)$, where X is the set of all objects in the data base, $v(t + s) = v(t) + v(s)$, where + is union, $v(t^*s) = v(t)^*v(s)$ where * is set intersection, and $v(t \rightarrow s) = (X - v(t)) + v(s)$.

Definition: $t = s$ if $v(t) = v(s)$.

Definition: Term t is in *normal form* if $t = t_1 + t_2 + ... + t_n$ and each t_i, $0 \leq i \leq k$, is simple.

Theorem 1. Let t be a term. Then there is a term s such that $t = s$ and s is in normal form.

The proof is straightforward and omitted.

The result we obtain from these definitions and Theorem 1 is that if we wish to identify the classes that have class descriptors satisfying a query, then we may put the query in normal form and find the set of class descriptions for each simple variable term separately.

STORAGE ORGANIZATION

We now describe a storage organization that will permit storage of data for statistical analysis as a partitioned database, and that will allow efficient implementation of the user data manipulation operations described earlier while preserving the advantages of a partitioned database for the statistical analyses themselves.

Data Structures

For each equivalence class defined by a set of values of the selection attributes, a block of storage would be allocated to contain the following items:

1. The summary quantities described in *Statistical Operations;*
2. Head and tail pointers to a chain of blocks containing the values of the variables for each case.

In order to minimize overhead, users creating a new database would be asked to estimate the maximum number of variables they are likely to require. This estimate would be used to determine the size of both the class header blocks and the data blocks. (Suitable utilities could be provided to rebuild the database, should the estimate prove grossly in error.)

Retrieval

The major problem in retrieval from a partitioned database is that the number of possible classes may be very large; if there are 10 selection attributes with only 5 values each, 5^{10} classes are possible. There are two circumstances which make it likely that, at any point in time, a substantial number of the classes will be empty. The first is that in the initial stages of the research, when the researcher is still making decisions about appropriate coding schemes, the number of cases actually entered may be quite small, even though the number of classes needed to store them may be quite large. Second, there may be functional dependencies within the data that insure that some classes remain empty; for example, if everyone who smokes has at least one heart attack, then the class for smokers who have not had a heart attack will always remain empty.

In order to avoid storage wastage, it would be desirable to avoid creating class headers for non-existent classes. Hence, the problem becomes one of searching a sparse space of class identifiers.

The solution we propose is to treat the search for class headers as a substring search problem. The nonempty class descriptors are formed into a string sequentially. By sequentially, we mean one class descriptor follows another without separator characters. The order in which class descriptors are placed in this string is the same as the order of the class headers. This string will be referred to as the data stream.

Let f be a comparison function where:
1. $f(a,b) = 0$ if $a = b$ and 1 if $a \neq b$.
2. $f(@,a) = 0$ for all a.
3. $f(a^c,b) = 0$ if $a \neq b$ and 1 if $a = b$.
4. $f(a^g,b) = 0$ if $b > a$ and 1 otherwise.
5. $f(a^l,b) = 1$ if $b < a$ and 1 otherwise.

We may assume that when a query is entered, it may be analyzed and simplified and changed to normal form. The query is now of the form where each term is of the form, $t_1 + t_2 + \ldots + t_n$, and each t_i is a simple term. A pattern is query in normal form where the terms are concatenated in the following order: $p = t_1\, t_2^R\, t_3\, t_4^R \ldots t_n$ or t_k^R if k is even and t_k^R is the reversal of t_k. The data stream is $d_1\, d_2 \ldots d_n$, where each d_i is a class descriptor.

First, we shall take the case where p has only one simple term. Then the pattern p is passed over d_1 and compared symbol by symbol using the comparison function. The comparison continues until the comparison function registers a 1 or all n symbols in the pattern have been compared. If the comparison function registers a 0 for all the symbols in the pattern, then we know d_1 is in the value set of the query. The pattern then goes on to d_2. If a 1 has been registered, then the pattern goes directly to d_3. The pattern match continues this way until the entire data stream has been matched.

If the pattern is composed on more than one simple term, then after t_1 is compared, the pattern and data flow are reversed and d_1 is compared to t_2^1, and so on until the entire pattern is compared to d_1. A similar procedure is used for $d_2 \ldots d_m$. A tally is kept as the data stream advances so that if the d_i class description is being examined, the tally contains an i. The system can then locate the ith class header to retrieve the desired information.

If each term is t descriptors long, there are c classes in the database, and there are q terms in the query, the algorithm will have a worst case performance of $0(q*t*c*t)$. While this bound will be large, particularly if the number of descriptors is large, we note several advantages to this approach over other methods of locating items in a sparse space. First, since the information is stored in a highly compact form, without any space needed for pointers, it will usually be possible to contain the entire data stream in primary memory; hence, the time required for a search is kept to a minimum. Second, it is easy to add new classes by simply adding their descriptions to the end of the data stream; deletions can be handled by replacing a description by a symbol not in the alphabet so that a match never occurs. Finally, since the pattern matching problem already occurs in a wide variety of applications, it is a likely candidate for hardware enhancement, such as the use of VLSI components.

User Operations

In a previous section we specified operations that users of statistical analysis systems desired to perform on their data before doing the statistical analyses. We now describe how these operations would be performed using the structures proposed here.

Operations that do not create new classes

Adding cases. If a case to be added falls into an existing equivalence class, then addition of a case will involve (1) locating the class header via the pattern match mechanism just described, (2) updating the summary information in the header by adding the quantities from the new case, and (3) storing the case values in a data block. This last step could be accomplished most rapidly by using the tail pointer stored in the header.

Deleting a case. If a case to be deleted was not the only case in the class, it could be deleted by the following steps: (1) Locate the class header. (2) Subtract out the case quantities to update the summary information. (3) Search the chain of data blocks to locate the case. (4) Mark the case as deleted. (A special variable or bit in each case might be reserved for this purpose.) If desired, the spaces for deleted cases might be chained into a free storage list.

Adding a non-selector variable. If the preallocation scheme described earlier is used, then values for the new variable would be placed in the next available slot in all of the cases. (We assume that each case has the same number of variables even though some of the variables may have missing values.) The problem that this presents is how to locate a particular case for which a value is to be added. If each case has stored as a variable an identification number or case number, then it is easy to specify which case is desired. Finding it, however, could, conceivably, require a search of the entire file. A more tractable scheme is to require that the user specify values of selector variables for the case. Search would then be confined to a single equivalence class.

Deleting a variable. This would require a pass through all the data blocks and changing the value of the variable to some distinguished value. Alternatively, if a table is maintained which links names of variables to their locations within a case, the entry for the variable could be removed.

Transforming a non-selector variable, including recoding it. This operation would, in general, also require a pass through all data blocks. Note that if all the blocks for all classes are stored in the same file, this can be done by reading the blocks sequentially as if they were not linked. If the transformation is needed for only one analysis and is not to be saved afterward and if the transformation only involves operations with a constant, the summary information in the header block is sufficient to calculate the quantities needed for statistical analysis of the new variable.

Operations that create new classes

Operations that do not change the length of class descriptions

Adding cases that create new classes. This will occur when the combination of selection attributes in a new case has not occurred previously. When this happens, (1) a new class header is set up and the class description for the new class is added to the end of the search string, and (2) a new data block is allocated and the values of the new case are placed in it.

Deleting the only case in a class. This can be accomplished by (1) replacing the class description in the search string with symbols not in the alphabet so that a match against it always fails and (2) returning the data block to a free storage list.

Operations that change the length of class descriptors

All of the following operations will require rewriting the entire search string. Note, however, that even for the 100,000 class example, this could be accomplished in less than 50 milliseconds on the slowest commercially available processors.

Adding a new selector variable. This includes situations in which an existing non-selection variable is declared to be a selection variable, as well as those situations in which a new variable is created either by user entry or by transformations on existing variables. The steps involved are: (1) Recopy the search string to permit extending the length of a class. Pad each descriptor with a dummy, constant attribute. (2) For each existing class, examine the value of the new variable for the first case in that class. Use this value to replace the dummy attribute for that class. If the remainder of the cases in that class have the same value, no further work is needed. Otherwise, delete each case from its former class and create a new class for it. Repeat this process for all existing classes.

CONCLUSION

The problem of performing a statistical analysis on an entire database is equivalent to computing a function over that entire database. Addressing or indexing schemes that aid in the localization of particular entries are not of much help in such situations; indeed, if all access to the database must be through these schemes, their use can be slower than sequential processing of an unordered file. The scheme described here relies, instead, on computing and updating that function on the data as it is entered into the database. This notion is similar to the concept of "alerters" in relational databases (Buneman & Clemons[6]). Since most statistical analyses are based on a common set of initial computations, this means that if these quantities are computed at the time of entry they are available for all subsequent analyses. The use of this type of partitioned architecture for the database, thus, offers considerable advantage over simple, sequential organizations or conventional indexing structures.

REFERENCES

1. Nie, N.H.; Hull, C.H.; Jenkins, J.G.; Stenbrenner, K.; and Bent, D.H.: *Statistical Package for the Social Sciences*, New York: McGraw-Hill Book Company, 1975.
2. Dixon, W. J. (Series Ed.) *Biomedical Computer Programs P—Series*. 1977, Berkeley, Ca.: University of California Press.
3. Marek W. and Pawlak, Z. Information storage and retrieval systems: Mathematical Foundations. *Theoret. Computer Science*, 1976, *1*, 331-354.
4. Lipski, W. and Marek, W. On information storage and retrieval systems. In *Mathematical Foundations of Computer Science*, A. Mazurkiewicz and Z. Pawlak (Eds.), 1977, Banach Center Publications, Vol 2, Warsaw Poland: Polish Scientific Publishers, 215-259.
5. Lipski, W. On semantic issues connected with incomplete information databases. *A.C.M. Transactions on Database Systems*, 1979, *4*(3), 272-296.
6. Buneman, O. Peter and Clemons, E. K. Efficiently monitoring relational databases. *A.C.M. Transactions on Database Systems*, 1979, *4*(3), 368-382.
7. Wong, E. and Chiang, T. C. Canonical structure in attribute based file organization. *Comm. A.C.M.*, 1971, *14*, 593-597.