CMU-CS-78-147

ALGORITHMS FOR MULTIVARIATE NONPARAMETRICS

Jon Louis Bentley
Departments of Computer Science and Mathematics
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

When classical statisticians define a *statistic* they study it from many viewpoints, including its bias, power, robustness, and many other aspects. One facet of a statistic, however, has often been ignored by the classical statistician: the computational difficulty of computing the statistic. The field of *computational statistics* studies precisely this probelm. In this paper we adopt the viewpoint of computational statistics and study the problems of multivariate nonparametrics in this light. After examining one problem in detail, we survey a number of computational problems and then a set of computational structures for solving them.

This research was supported in part by the Office of Naval Research under Contract N00014-76-C-0370.

Table of Contents

1. Introduction

Many are the problems facing the practitioner of statistical computing. These include problem specification, collection and integrity of data, choosing statistical packages and programming languages, and producing and maintaining correct programs. Another problem which arises in statistical computing is that of reducing the computational resources (time and space) used by various programs. One must certainly keep the problem of efficiency in perspective with the other problems (to ensure that the programming costs involved with a faster program do not exceed the corresponding savings, for example), but it is a sad fact that many computer programs currently require enormous amounts of time and space.

There is a field of computer science, called *algorithm design and analysis*, that studies precisely the issues of computational efficiency. The application of the methods of this field to the problems of statistics is the domain of the hybrid discipline of *computational statistics*. Shamos [1976, 1978, 1979] has laid the foundations of computational statistics by identifying and studying a number of fundamental computational problems that arise in statistics. This study is of both practical and theoretical interest: on the practical side it reveals reductions in the costs of many computations, and theoretically one can gain a new perspective on statistics as well as a host of problems fascinating for computer scientists.

One shortcoming of computational statistics to date has been that it has studied primarily univariate and bivariate problems. In this paper we will survey how the methods of computational statistics can be applied to multivariate problems; we will specifically investigate computational problems that arise in multivariate nonparametric statistics. In Section 2 we will study a particular problem in some detail. Sections 3 and 4 survey a number of multivariate problems and the computational structures that can be used to solve them. In Section 5 we will study a couple of those problems in lesser detail, showing how each of the computational structures might be applied. Finally, directions for further research and conclusions are offered in Section 6.

2. The ECDF Problem

An important function in mathematical statistics is the *cumulative distribution function* (abbreviated CDF) associated with a distribution. If a distribution has probability density function (abbreviated PDF) f, then the CDF F of the distribution is the integral of f. Since a distribution is essentially defined by its CDF, knowledge of that function gives great power in making statements about the distribution. When the data analyst given some set S of multivariate observations (which we view as a set of points in a multidimensional space) he usually makes the assumption that they are drawn independently from some underlying

distribution f, but he has no knowledge of what that function (or its CDF F) might be. In this case, however, he can approximate the CDF of the underlying distribution by the *empirical cumulative distribution function* (abbreviated ECDF) of the point set S. To define the ECDF we need two definitions: a point x is said to *dominate* point y if x is greater than y in every coordinate, and the *rank* of a point x in set S (written rank(x,S)) is the number of points in S dominated by x. If a point set S contains N elements then the ECDF of point x in S is

ECDF(x,S) = rank(x,S) / N.

These quantities defined on point sets are illustrated in the planar case in Figure 2.1.

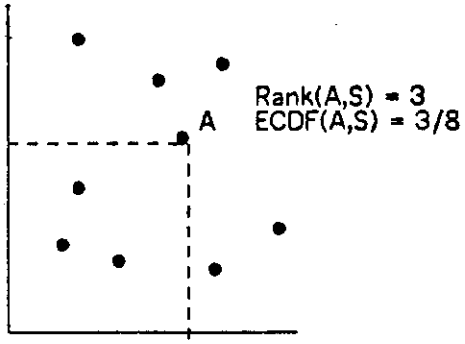

Figure 2.1. Point set S.

Calculation of the ECDF arises in a host of statistical problems. The multivariate two-sample Kolmogorov-Smirnov test of hypothesis that two samples were drawn from the same underlying distribution uses the multivariate ECDF. Other statistical applications that require calculation of the ECDF, including density esitmation and Hoeffding and Cramer-von Mises hypothesis tests, are described by Bentley and Shamos [1977].

There are two computational problems associated with the mathematical definition of ECDFs. The first problem is *ECDF searching*: organize a set S of N points so that subsequent queries asking the rank(x,S) (where x is not necessarily in S) may be answered quickly. The second computational problem is the *all-points ECDF* problem: given a point set S, find the rank of each point in S. Naive algorithms for ECDF searching require O(kN) time for k-dimensional sets of N points, and naive all-points algorithms require $O(kN^2)$ time. In this section we will see algorithms due to Bentley and Shamos [1977] that solve these problems more efficiently. We will examine the ECDF searching problem in Section 2.1 and then the all-points problem in Section 2.2. Although we will restrict our attention in those sections to the planar (two-variate) case, the algorithms generalize to k-space (we mention this in Section 2.3). Throughout this section we will concentrate on the problem of computing ranks rather than the ECDF, but recall that after calculating rank a single division suffices to compute ECDF.

2.1. The ECDF Searching Problem

In the ECDF searching problem we are to organize a set S of points such that subsequent queries asking the rank(x,S) may be answered quickly. If S is a one-dimensional set then we can answer rank queries by first sorting S and then answering queries by performing a binary search in S. The cost of this is O(N lg N) for sorting the elements, and then O(lg N) for answering queries. Unfortunately, this scheme does not seem to generalize directly to the planar case, but we will see how the fundamental notion of binary search can be applied to this problem.

We can develop a fast algorithm for planar ECDF searching based on the fact that for any given point set S there exists a set of rectangles within which the rank remains constant. To illustrate this consider holding a y-value fixed and "sliding" along the x-value. In doing so, the rank of a point can change only as we encounter the x-value of one of the points in the set. A similar observation holds for fixed x-values. It is therefore true that if we were to "draw" vertical and horizontal lines through each of the N points of a set, then the rank is constant within each of the $(N+1)^2$ resulting rectangles. This situation is illustrated in Figure 2.2--the number in each rectangle denotes the rank of any point that lies in that rectangle. Once we have organized the point set S in this way we can answer queries by "looking up" the rectangle in which the point lies, and then returning the integer associated with the rectangle as the rank of the point. The general method underlying this particular algorithm is usually called the "locus method" -- notice that we solved the problem by identifying the loci of all points with equal ranks.

0	1	2	3	4	5
0	1	1	2	3	4
0	0	0	1	2	3
0	0	0	1	1	2
0	0	0	0	0	1
0	0	0	0	0	0

Figure 2.2. Rank in each cell is constant.

The above scheme is easily implemented on a computer if both the x- and y-values are first "normalized" to the integers 1 to N by sorting. After this step the array of rectangles can be stored in a two-dimensional arrary. To answer a query we then do a binary search

for both coordinates to normalize them to integers, after which we perform an array lookup. Analysis of this method shows that $O(N^2)$ storage is required, and a query can be answered with two binary searches and array lookup, which require O(lg N) time. If some care is taken in implementation then the data structure can be built in $O(N^2)$ operations. Thus we see that by paying substantial preprocessing and storage costs we can speed up the query time over the naive algorithm. This structure might therefore provide large savings when a very large number of queries are to be performed.

2.2. The All-Points ECDF Problem

In the previous section we examined the problem of organizing a point set so that subsequent ECDF queries can be answered quickly; in this section we will examine the related problem of calculating for each point in the set its rank in the set. This is, of course, just the all-points ECDF problem. For univariate data one can solve this problem by sorting the observations (points) and then scanning through the sorted set, reporting the number of points less than each point encountered. If an O(N lg N) sorting algorithm is used then the total cost of this procedure is also O(N lg N). Although we cannot immediately generalize sorting from one dimension to two, we will see that we can use the underlying idea of many sorting algorithms, *divide-and-conquer*, to create a fast algorithm for solving the planar all-points ECDF algorithm.

The planar divide-and-conquer algorithm that we will describe operates in the three steps that are sketched in Figure 2.3. The first step of the algorithm (illustrated in part a of the figure), the *divide* step, chooses some vertical "cut line" L that separates the given point set S into two subsets A and B of equal size (that is, N/2). The second (*recur*) step of the algorithm recursively finds for each point in A its rank among the points in A, and for each point in B its rank among the points in B; this is illustrated in part b of the figure. Note that at this stage we have found the correct rank of each point in A: since the x-coordinate of each point in A is less than the x-coordinate of any point in B, no point in A can dominate any point of B. All that remains to be done therefore is to find for each point in B the number of points in A that it dominates. The above observation that all points in A have lesser x values than all points in B means that all we have left to do is calculate for each point in B the number of points in A with lesser y-value. The third step (*marry* step) of our algorithm is therefore to project all points in both A and B onto the line L (remembering whether each was in A or B), and then scan through the resulting list keeping track of the number of A's so far observed. As each B is encountered we add the count of A's to its correct rank among the B's, which gives its correct rank in the entire set. This third step is illustrated in part c of the figure.

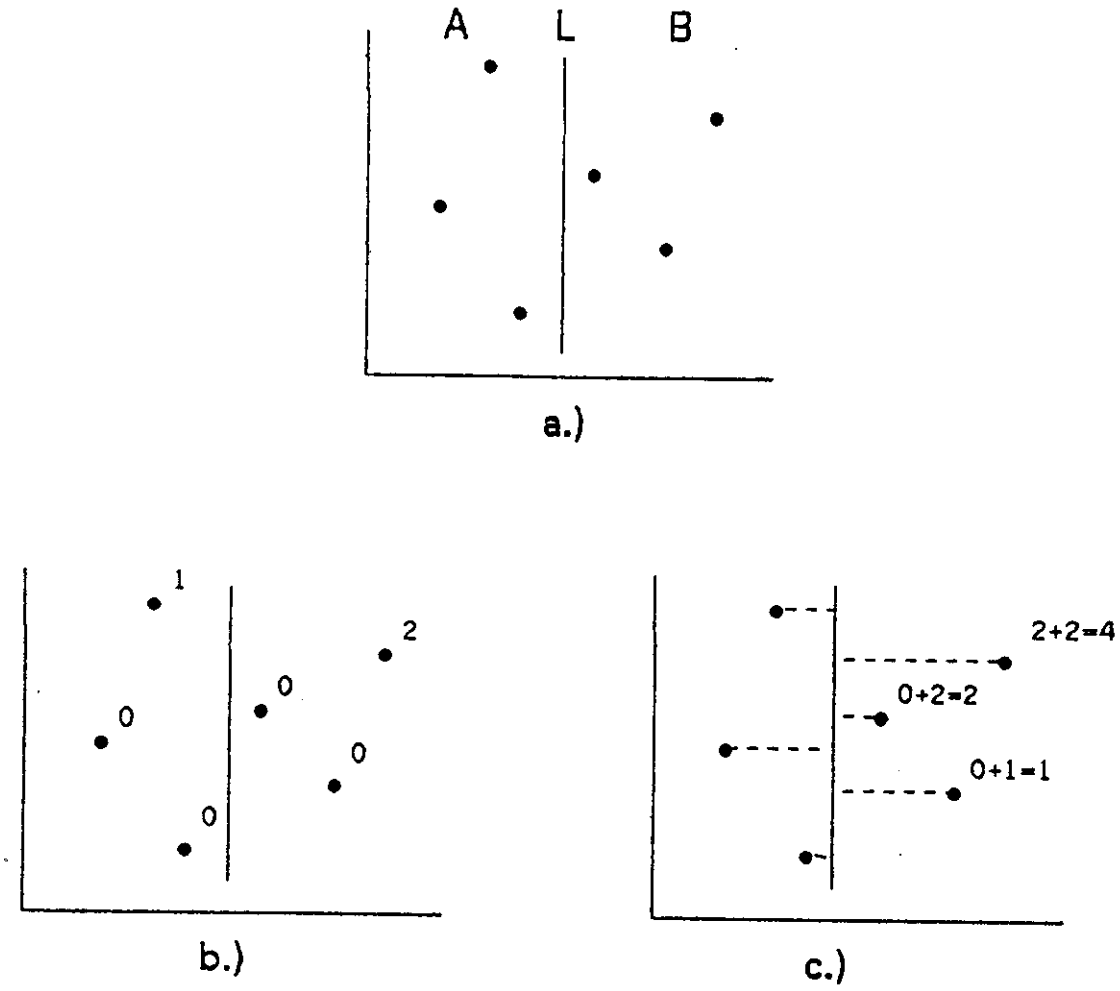

Figure 2.3. Operation of ECDF2.

We will now describe more formally the algorithm that we sketched above. We call it Algorithm ECDF2, and it is a recursive procedure that is initially passed a set S of N points.

1. [Divide.] If S contains just one element then return its rank as 0; otherwise proceed. Choose a cut line L perpendicular to the x-axis such that N/2 points of S have x-value less than L's (call this set A) and the remainder have greater x-value (call this B).

2. [Recur.] Recursively call ECDF2(A) and ECDF2(B). After this step we know the true rank of all points in A.

3. [Marry.] We must now find for each point in B the number of points in A it dominates (i.e., that have lesser y-value) and add that number to its partial rank. To do this, pool the points of A and B and sort them by y-value. Scan through this sorted list in increasing y-value, keeping track in ACOUNT of the number of A's so fa observed. Each time a B is encountered, add the current value of ACOUNT to its partial rank.

The correctness of this algorithm can be established by induction on the problem size, N.

To analyze the time requirements of the above algorithm we will let its running time on a set of N elements be denoted by T(N). Step 1 can be accomplished in O(N) time by the use of a fast median algorithm. Step 2, which solves two subproblems of the same form each of size N/2, requires 2T(N/2) time by the induction hypothesis. If careful bookkeeping is used, the sort of Step 3 can be avoided, and therefore the step can be accomplished in O(N) time. Thus the algorithm as a whole obeys the recurrence

$$T(N) = 2T(N/2) + O(N)$$

which has solution $T(N) = O(N \lg N)$. This algorithm (and many extensions) are described in detail by Bentley [1978].

2.3. Summary of the ECDF Problem

The algorithms that we have seen in the previous two sections for bivariate problems (which we viewed as points in the plane) can be generalized to algorithms for k-variate problems (which we view as point sets in k-space). The searching algorithm of Section 2.1 generalizes to a method that requires $O(N^k)$ preprocessing time and storage; subsequent queries can then be answered in $O((\lg N)^k)$ time. The divide-and-conquer method of Section 2.2 can be generalized to k-space and yields an algorithm with $O(N (\lg N)^{k-1})$ running time.

Although we see that the algorithms are faster asymptotically than their naive competitors, we might wonder what gains they offer in practice. As an example, consider the problem of solving a one-million-element all-points ECDF problem on a one-microsecond computer. In this example the $O(N^2)$ naive algorithm would require some twelve days, while the $O(N \lg N)$ divide-and-conquer algorithm would require just twenty seconds. This asymptotic analysis of algorithms will become more and more important as data sets in practice are becoming larger and larger; this trend will continue as online data collection systems become more commonplace.

At this point we have spent considerable effort on a single computational problem, and we might wonder what all this work has gained us. On the most basic level we have seen two particular algorithms for solving the two computational problems originally posed. More generally, we have seen two algorithmic techniques which can be used for a variety of problems: the locus method and multidimensional divide-and-conquer. Finally, we have achieved a whole new, *algorithmic*, view of a well known statistical problem.

3. A Survey of Problems

In the last section we saw how a number of problems in statistical computing can be solved by reduction to a single geometry problem: calculating the rank of a given point in a set of N points. In this section we will describe a number of problems in computational geometry that arise in statistical problems. The interest of all these problems is that N k-variable observations can be viewed as N points in k-space. A feature that most of the problems we will see share with the ECDF problem is that there are two varieties of each problem: the searching version and the all-points version.

The first few problems that we will investigate are defined in terms of point domination (recall that point x dominates y iff x is greater than y in all k coordinates). The first problem we mentioned was the *ECDF problem*, which asked for each point the number of points it dominates (its rank). We saw that the ECDF problem arises in such statistical applications as hypothesis testing and multivariate density estimation. A related problem is the *maxima* problem, which asks if a given point is dominated. This problem appears in both the all-points and searching forms and arises in econometrics and outlier detecttion. A final problem that can be phrased in terms of domination is *range searching*, which asks for all points dominated by y and dominating x (that is, for all points in some rectilinearly oriented rectangle). This problem occurs in both data analysis and multivariate density estimation.

A related set of problems are defined in terms of point closeness. Perhaps the simplest such problem is the *nearest neighbor* problem, which calls for finding the closest point in the set to a given point. This can be cast in both the all-points and searching frameworks. This problem arises in a host of statistical applications, including classification, clustering, density estimation, and hypothesis testing. A problem related to the "pure" nearest neighbor problem is the *fixed-radius near neighbors* problem: we now ask for all points within some fixed distance d of the given point. This problem arises in many of the same applications as the "pure" nearest neighbor algorithm. A final closeness problem is the *minimal spanning tree* problem, which asks for a set of N-1 edges connecting the N points with minimal total edge length (this problem is only an all-points problem; it has no searching analog). This problem occurs in single linkage clustering, as well as many other data analysis procedures.

4. A Survey of Structures

In Section 2 we saw two general schemes that yielded particular algorithms: multidimensional divide-and-conquer and the locus method. In this section we will mention a number of other *algorithmic paradigms* that can be used to solve a host of computational problems. For each method we will describe it in general terms and then mention how it

might be used to solve the fixed-radius near neighbor searching problems. More detail on these structures can be found in Bentley and Friedman [1978b].

The simplest scheme for solving multidimensional problems is brute force by simply *holding the points in some structure such as an array.* To solve the fixed-radius near neighbor problem wih his method we just hold the poins in an array and answer each query by scanning through the entire array. A more sophisticated approach is the projection method, which stores the points by *projecting them onto one of the coordinate axes.* A near neighbor algorithm based on this method sorts the points by one coordinate and can then use that information to decrease query time. Another multidimensional paradigm is the cell method which calls for *partitioning space into cubes and placing each point in its cube.* In a near neighbor method based on this technique only points in cells near a query point would be examined during a search.

The fourth multidimensional paradigm in common use is recursive paritioning, which *chooses a plane dividing the set in half, and builds two substructures recursively.* Applying this method to the near neighbor problem gives a multidimensional binary search tree. The remaining two structures we have seen applied to the ECDF problem. Multdimensional divide-and-conquer *solves a problem of N points in k-space by solving two subproblems each of N/2 points in k-space, and one subproblem of N points in (k-1)-space.* Finally, the locus method calls for *identifying the locus of all points sharing the same response to a query.*

5. Examples

In Section 3 we saw a number of computational problems in multidimensional geometry and in Section 4 we glimpsed a number of structures useful in solving many different computational geometry problems. In this section we will see a marriage of the material in the two sections by showing how the the structures can be used to solve some of the problems.

The first problem we will examine is the all-nearest neighbors problems, that is, given N points in k-space find the nearest neighbor of each point. A brute force algorithm for this problem yields $O(N^2)$ performance. Friedman, Baskett and Shustek [1975] have shown how projection can be used to give an $O(N^{2 - 1/k})$ algorithm. Bentley, Weide and Yao [1978] used the cell technique to give a linear expected time algorithm for all nearest neighbors, but their result holds only for point sets drawn from "smooth" underlying distributions. The idea of recursive partioning was used by Friedman, Bentley and Finkel [1977] to give an algorithm with expected O(N lg N) performance for a wide class of input distribuions. Finally, multidimensional divide-and-conquer was used by Bentley [1976] to give an algorithm with

provable $O(N (\lg N)^{k-1})$ worst-case running time. Of all the algorithms mentioned above, only brute force, projection, and cells seem to be easily implementable and robust enough to merit practical application. To compare the merit of those three algorithms we can give their running times in Fortran implementations solving a 16,000 point problem in 4-space. Brute force requires some 20 minutes of IBM 370/168 CPU time, projection requires approximately 7 minutes, and recursive partitioning requires just 50 seconds. Thus we see that these methods can be used to yield substantial time savings for realistic problem sizes.

As a final example we mention the minimal spanning tree problem. Brute force yields an $O(N^2)$ algorithm and Bentley and Friedman [1978a] describe how recursive partitioning can give an $O(N \lg N)$ algorithm. For point sets of 16,000 points in 2-space the brute force algorithm requires 45 minutes of IBM 370/168 time, while the recursive partitioning algorithm requires just 45 seconds.

6. Conclusions

In this paper we have been able to scratch the surface of the study of algorithms for multivariate nonparametrics. The primary object of our investigation has been a set of computational problems motivated by mathematical functions needed for nonparametric statistics. Our investigation has led us to study both particular algorithms and data structures as well as a collection of techniques suitable for solving a large class of problems. The algorithms we saw in Section 2 are both theoretically elegant (both algorithms can be proved optimal under a reasonable model) and practical for some applications.

Our study of mulivariate nonparametrics has been typical of most efforts in computational statistics in many ways. From a wide number of statistical tests and procedures we have isolated a kernel of computations often performed. We then developed a set of techniques capable of solving many problems in that kernel, rather than solving particular problems with *ad hoc* techniques. The fruits of this study include both a set of algorithms (theoretically elegant and occasionally practical), and a new *algorithmic* understanding of some statistical problems.

References

Bentley, J. L. [1976]. Divide and conquer algorithms for closest-point problems in multidimensional space, Ph.D. Thesis, University of North Carolina, Chapel Hill, North Carolina, 101 pp.

Bentley, J. L. [1978]. Multidimensional divide-and-conquer, submitted for publication.

Bentley, J. L. and J. H. Friedman [1978a]. "Fast algorithms for constructing minimal spanning trees in coordinate spaces," *IEEE Transactions on Computers C-27*, 2, February 1978, pp. 97-105.

Bentley, J. L. and J. H. Friedman [1978b]. A survey of algorithms and data structures for range searching, submitted for publication.

Bentley, J. L. and M. I. Shamos [1977]. "A problem in mulivariate statistics: algorithm, data structure, and applications," *Proceedings of the Fifteenth Allerton Conference on Communication, Control and Computing*, pp. 193-201.

Bentley, J. L., B. W. Weide, and A. C. Yao [1978]. "Optimal expected-time algorithms for closest-point problems," to appear in *Proceedings of the Sixteenth Allerton Conference on Communication, Control and Computing.*

Friedman, J. H., F. Baskett, and L. J. Shustek [1975]. "An algorithm for finding nearest neighbors," *IEEE Transactions on Computers C-24*, 10, October 1975, pp. 1000-1006.

Friedman, J. H., J. L. Bentley, and R. A. Finkel [1977]. "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathemaical Software 3*, 3, September 1977, pp. 209-226.

Shamos, M. I. [1976]. "Geometry and statisics: problems at the interface," in *Algorithms and complexity: New directions and recent results*, J. F. Traub (ed.), Academic Press.

Shamos, M. I. [1978]. Computational geometry, Ph.D. Thesis, Yale University, New Haven, Conn.

Shamos, M. I. [1979]. Computational statistics, manuscript in preparation.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CMU-CS-78-147	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ALGORITHMS FOR MULTIVARIATE NONPARAMETRICS		5. TYPE OF REPORT & PERIOD COVERED Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Jon Louis Bentley		8. CONTRACT OR GRANT NUMBER(s) N00014-76-C-0370
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Computer Science Dept Schenley Park, PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, VA 22217		12. REPORT DATE October 1978 13. NUMBER OF PAGES 14
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as above		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; Distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)