83-3

Relevant Variable Selection for Inductive Learning Programs

by

Jordan Pollack

January 1983

File No. UIUCDCS-F-83-902

ISG 83-2

Relevant Variable Selection for Inductive Learning Programs

by

Jordan Pollack

Janauary 1983

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

ISG 83-2

File No. UIUCDCS-F-83-902

This work was supported, in part, by a National Science Foundation Grant, No. MCS-82-05166.

1. SUMMARY

The AQ technique for inductive inference, learning, and classification [Michalski, 1977] is a robust and sensitive method which has, at its heart, an efficient and controllable approximate solution [Michalski, 1971] to the NP-complete covering problem. As an approximate technique, it does not suffer the same intractability problems as an exact technique [Garey & Johnson, 1978], but, nonetheless, very large problems can rapidly eat up all reasonable computer resources.

There are several approaches to minimizing the cost of forming inductive rules. One approach, used in the ESEL program [Michalski & Larson, 1978], is to reduce the number of events in the problem. Another approach, investigated in this project, is the reduction of variables.

We have written a program which embodies 4 distinct approaches to this problem of relevant variable selection. The first technique is to throw away variables until no more can be thrown away (sieve); The second is to select variables one at a time (greedy); the third and fourth involve iterative random selection of variables and evaluation of the selected set with feedback into the selection process. The third technique uses an approximate evaluation scheme while the fourth actually uses the GEM version of AQ.

These four techniques were applied to some problems and the results showed the third technique (random selection with approximate evaluation) to be the most cost-effective.

2. INTRODUCTION

Within a set of observed features for an inductive learning/pattern classification problem there is usually much more information present than is actually needed to discriminate the pattern classes. In addition to observed feature variables, constructive inference programs will actually generate many new variables and thus even more redundant information. This overabundance of information causes the computational techniques to use more resources than are actually necessary. One approach to this problem is to select a reasonable subset of feature variables. This process of relevant variable selection must be optimized with respect to two conflicting principles. The economy principle is that the selection should use as little computer time as possible -- it should be substantially more efficient to select a subset of variables and then run AQ rather than to run AQ on all the variables. The parsimony principle is that the classification rules found by AQ should be as elegant as possible.

Several programs have been written which approach the variable selection problem with these principles in mind. These programs have been run on two sets of data. Economy and parsimony statistics have been collected, and the results analysed.

2.1. SIMPLIFYING ASSUMPTIONS

These programs have all been written with the assumption that the classification data completely discriminates the pattern classes. This assumption is needed because the GEM program can currently only handle disjoint covers, and because the difference between completely discriminatory variable sets from incomplete ones is easy to detect and thus forms the basis for many selection techniques.

3. TECHNIQUES

Four algorithms for relevant variable selection have been written and are briefly described below. The actual computer program (in PASCAL) is included as an appendix.

3.1. SIEVE

The sieve technique is intended to be the most economic process for finding a set of relevant variables. First, an initial measure is taken of the "promise", or relevancy, of each variable. The information theoretic measure of "mutual information" [Shannon, 1949] is used, and provides a real number between 0 and 1 for each variable. If the promise of a variable is 0, then it is complete noise, and does not help discriminate the pattern classes at all; if it is 1, then it completely discriminates the data. The formula for Mutual Information is:

$$MI(v) = \sum_{w,x} p(w,x) \log \frac{p(w,x)}{p(w)p(x)}$$

where w stands for a pattern class and x for a value of the variable v. P(w) is the probability of class w, which is the size of class w divided by the number of events; P(x) is the probability that v takes on value x, and p(w,x) is the probability that both the class is w and the variable is x.

After the promise of each variable is computed, the variables are sorted by increasing promise. Then, one at a time, variables are removed, and the remaining set is tested for complete discrimination. If the removal of a variable causes incomplete discrimination, it is added back into the set.

3.2. GREEDY

Whereas the sieving technique described above started with the full set of variables and threw irrelevant ones away, the greedy method [Horowitz & Sahni, 1978] starts with no variables and adds them one at a time. The mutual information measure does not only apply to individual variables, but also to sets of variables. The greedy algorithm first selects the single variable with the maximal promise. Then it looks at that variable in conjunction with each of the remaining variables and picks the pair with the maximal

promise. Then it looks at that pair in conjunction with the remaining variables and selects the best triple. This continues until the mutual information is 1, and the data can be completely discriminated.

This greedy method is an $O(n^2)$ (where n represents the number of variables) approximate algorithm for finding the minimal discriminatory variable set (MDVS). An early hypothesis was that the MDVS would indirectly satisfy the parsimony principle since the generated rules would be expressed in terms of the fewest variables. This hypothesis is easily proven false. Consider the following example:

	var 1	var 2	var 3
class 1	1	1	3
	1	2	4
	1	1	1
class 2	2	4	4
	3	4	1
class 3	2	1	2
	3	2	2
	4	3	2

Although the set of variables {v1,v2} completely discriminate the classes, its rules would have to be at least:

[v1=1]=>class1 [v2=4]=>class2 [v1=2..4][v2=1..3]=> class3

whereas using all three variables give the rules:

[v1=1]=>class1 [v2=4]=>class2 [v3=2]=>class3

which are clearly more parsimonious rules.

3.3. RANDOM SELECTION/APPROXIMATE EVALUATION

Both of the techniques described above make only a single selection of variables, and both are quite direct in their implementation. The random selection/evaluation technique makes multiple guesses at a good set of variables, and then factors the evaluation of those sets back into the selection procedure. The program for random selection is driven by a vector of real number between 0 and 1 (initially it is the mutual information measure for each variable.) Based on these weights, variables are selected one at a time until at least a specific number of variables have been chosen, and the set of variables is completely discriminatory. The approximate evaluation function is to multiply the number of distinct events by the number of variables. The vector of weights is updated by first decaying every value (multiply by .95) then if the selected set of variables is better than

average, a fractional root is taken of the weight for each variable in the selected set, between .5 (square root) and 1 (no change), depending on how "good" the whole variable set is. This update function was arrived at by experimentation on functions whose domain and range are (0,1).

The effect of this random, iterative process, is that the probability of finding good sets of variables is increased when good sets are found. It is not a completely convergent process, but it seems to work satisfactorily.

3.4. RANDOM SELECTION / EXTERNAL EVALUATION

The random process described above uses a very simple evaluation function that does not involve actually running an AQ program. This technique, which runs in a procedure on the cyber, actually runs GEM on subsets of variables, and uses an external evaluation function. The selection of variable sets and the update of weights is done exactly the same as in the approximate evaluation technique. The evaluator is a separate pascal program which reads the output file from GEM and counts selectors and complexes.

4. RESULTS

All four of the techniques were run on two test cases. The approximate random algorithm was set to loop 15 times, while the external random one was set to loop only 10 times. The first test case is a small artificial problem, to classify faces based on simple features. The second test case is the soybean disease diagnosis problem [Michalski & Chilausky, 1980]. The first test case is quite small, with only 8 variables, 3 classes and 10 events, but has a rich set of discriminatory subsets and a wide range of variable relevancy. The second case is quite large, with 35 variables, 19 classes and 74 events.

The two tables below summarize the statistics for these test cases. The columns correspond to the different selection algorithms. The rows contain the following information:

1. CPU time in seconds to run selection algorithm
2. Size of variable set selected
3. CPU time in seconds to run GEM procedure
4. Total number of Selectors in result
5. Total number of Complexes in result
6. Average number of selectors per Complex
7. Average number of selectors per event class
8. Average number of Complexes per event class
9. Total CPU time in seconds

4.1. COMPUTATIONAL RESULTS

Test Case 1: 3 classes, 10 events, 8 variables.					
	Sieve	Greedy	Approx Random	Extern Random	All Vars
Select CPU (sec)	0.051	0.075	0.201	~3	
# vars	4	3	4	6	8
Gem CPU	0.189	0.145	0.170	0.238	0.293
# Selectors	13	12	12	12	12
# Complexes	7	7	7	6	6
Selectors/Complex	1.8	1.7	1.7	2.0	2.0
Selectors/class	4.3	4.0	4.0	4.0	4.0
Complexes/class	2.3	2.3	2.3	2.0	2.0
Total CPU	0.240	0.220	0.371	~3.2	0.293

Test Case 2: 19 classes, 74 events, 35 variables					
	Sieve	Greedy	Approx Random	Extern Random	All Vars
Select CPU (sec)	2.261	7.277	2.588	~45	
# Vars	7	5	7	9	34
Gem CPU	4.509	4.149	3.685	3.743	16.181
# Selectors	124	122	83	77	121
# Complexs	35	37	35	31	20
Selectors/Complex	3.5	3.3	2.3	2.5	6.1
Selectors/class	3.6	3.6	2.4	2.3	3.6
Complexes/class	1.03	1.08	1.02	0.91	0.59
Total CPU	6.770	11.426	6.237	~48.7	16.181

4.2. ANALYSIS

There are several points to notice here. First, that the external random algorithm takes an excessive amount of time, and thus should be discarded as an candidate for variable selection, unless combined with other techniques which will minimize the number of calls to GEM. (Possibly, the resource usage of this technique would be better if run inside an operating system which allowed coroutines and pipes (like UNIX), or if the core of GEM were isolated to be used as a subroutine, so there would not be as much file I/O.) Second, notice that although the greedy algorithm found the smallest number of variables, the resultant rules from GEM were not the most parsimonious, and that the N^2 nature of this algorithm is showing

through. The sieve technique used the minimum amount of time, as expected, but was not too impressive in the rules generated by GEM.

Finally, and most importantly, the clear winner out of all this was the random selection with approximate evaluation. It used computer time very sparingly with very good and parsimonious rules found by GEM.

5. CONCLUSION

Computer resources stand to be gained if bad features are weeded out of the input to an inductive inference program. Several techniques were implemented in PASCAL on the CYBER, and compared for both economy and parsimony. The clear winner was a random selection technique with an approximate evaluation of variable sets, and feedback.

6. REFERENCES

[Garey & Johnson, 1978] Computers and Intractability: A guide to the theory of NP-Completeness San Francisco: Freeman & Co.

[Horowitz & Sahni, 1978] Fundamentals of Computer Algorithms Maryland: Computer Science Press.

[Michalski, 1971] "A Geometric Model for the Synthesis of Interval Covers", Report 461, DCS, Urbana.

[Michalski, 1977] "Toward Computer Aided Induction: A brief Review of currently Implemented AQVAL programs" Report 874, DCS, Urbana.

[Michalski & Larson, 1978] "Selection of most representative Training Examples and Incremental Generation of VL1 Hypothesis: the underlying methodology and the description of programs ESEL and AQ11", Report 867, DCS, Urbana.

[Michalski & Chilausky, 1980] "Learning by Being Told and Learning from Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis", International Journal of Policy Analysis and Information Systems, Vol. 4, No. 2, pp. 125-161.

[Shannon, 1949] The Mathematical Theory of Communication Urbana: University of Illinois Press.

7. APPENDICES

7.1. Data for Test Case 1

Test Case one is an artificial problem in facial recognition. There are three classes of faces, and eight variables, as follows. Pictures are shown on the following page.

	hat	mouth	eyes	ears	nose	brow	mustache	beard
class 1	beret	frown	slit	square	hump	normal	up	no
	beret	frown	round	square	slope	normal	curly	yes
	derby	slit	slit	normal	hump	normal	up	no
class 2	derby	smile	round	round	slope	normal	no	yes
	derby	smile	normal	pointy	slope	normal	down	yes
class 3	fedora	slit	normal	square	normal	normal	curly	no
	beret	frown	slit	square	hump	single	up	no
	derby	smile	normal	pointy	slope	normal	no	yes
	berby	smile	round	round	slope	normal	up	yes
	fedora	slit	normal	square	normal	normal	no	yes

7.2. Data for Test Case 2

The data for this test case is from the soybean disease diagnosis problem. See [Michalski & Chilausky, 1980] for details.

7.3. Sample output for Test Case 1

This data shows the weighting behavior of the random selection / approximate evaluation program, as run on test case 1. The first set of weights is derived from the mutual information measure for each variable taken independently. Each subsequent iteration updates these weights as variables are used together.

7.4. Procedure for Random Selection / External Evaluation

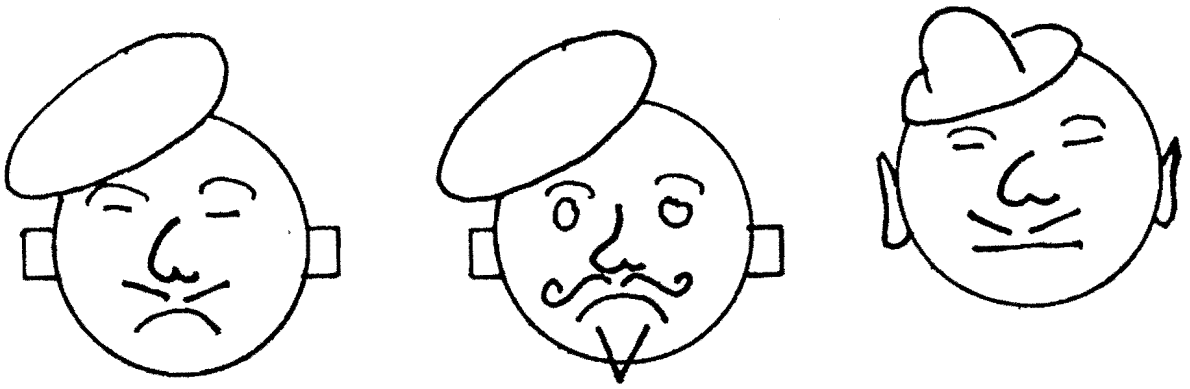
This is the CYBER JCL for running the Random Selection / External Evaluation algorithm. All sorts of temporary files are used.

7.5. External Evaluation Program

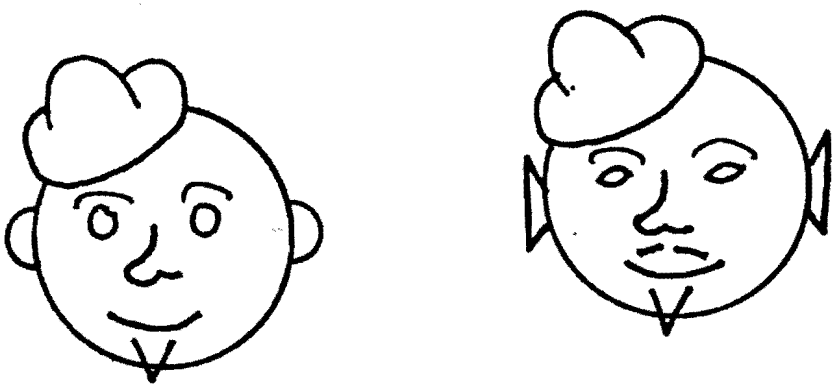
The random/external algorithm uses an independent evaluation function which should read a GEM output file and return an integer. The smaller the integer, the better the results of GEM. This appendix contains one such program.

7.6. Main Program A large program for your perusal, with lots of comments.

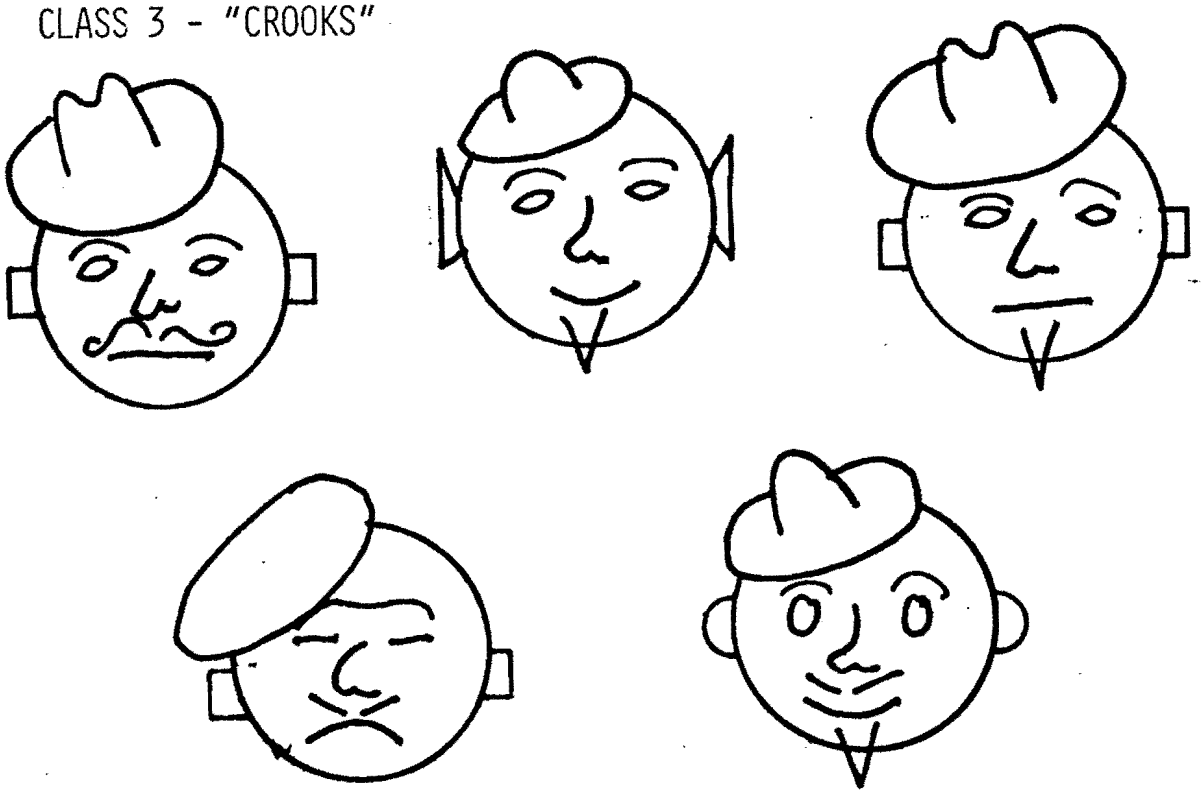
CLASS 1 - "ROBBERS"



CLASS 2 - "THIEVES"



CLASS 3 - "CROOKS"



19 (classes)
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 9 8 8 4 (events in each class)
74 35 (number of events, variables)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
6	0	2	1	0	3	0	1	1	1	1	1	0	2	2	0	0	0	1	1	3	1	1	1	0	0	0	0	4	0	0	0	0	0	0
4	0	2	1	0	3	0	1	1	2	1	1	0	2	2	0	0	0	1	1	3	0	1	1	0	0	0	0	4	0	0	0	0	0	0
4	0	2	1	0	2	0	2	1	1	1	1	0	2	2	0	0	0	1	0	3	1	1	1	0	0	0	0	4	0	0	0	0	0	0
6	0	0	2	1	3	3	1	1	0	1	1	0	2	2	0	0	0	1	0	0	3	0	0	0	2	1	0	4	0	0	0	0	0	0
5	0	0	2	1	2	2	1	0	2	1	1	0	2	2	0	0	0	1	0	0	3	0	0	0	2	1	0	4	0	0	0	0	0	0
6	0	0	2	1	3	3	1	1	1	1	1	0	2	2	0	0	0	1	0	0	3	0	0	0	2	1	0	4	0	0	0	0	0	0
1	1	2	0	0	2	1	2	0	2	1	0	0	2	2	0	0	0	1	0	1	1	0	1	1	0	0	3	4	0	0	0	0	0	0
1	1	2	0	0	1	1	2	0	1	1	0	0	2	2	0	0	0	1	0	1	1	0	1	0	0	0	3	4	0	0	0	0	0	0
1	1	2	0	0	2	1	1	0	1	1	0	0	2	2	0	0	0	1	0	1	1	0	1	0	0	0	3	4	0	0	0	0	0	0
3	1	1	1	-1	3	1	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	1	-1	3	2	-1	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	1
3	1	1	1	-1	2	1	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	1	-1	3	2	-1	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	1
2	1	1	1	-1	3	1	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	1	-1	3	2	-1	0	0	0	0	-1	-1	-1	-1	-1	-1	-1	1
5	0	0	2	0	3	2	1	1	1	0	1	0	2	2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
4	0	0	1	0	1	3	1	1	2	0	1	0	2	2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
4	0	0	1	0	3	2	1	0	1	0	1	0	2	2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
4	1	1	0	0	2	2	0	2	0	0	1	0	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	1	0	1	0	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	0	1	0	1	1	1	0	1	0	1	0	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	2	1	0	1	1	0	1	2	0	1	2	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
5	1	2	1	0	2	3	0	1	2	0	1	1	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
4	1	2	0	0	1	2	1	0	1	0	1	2	0	1	0	0	2	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
2	0	2	1	0	2	2	1	0	1	0	1	2	1	1	1	0	0	1	0	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0
3	1	2	1	0	3	3	1	0	2	0	1	2	1	1	1	0	0	1	0	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	2	1	0	2	2	1	0	1	0	1	2	1	1	1	0	0	1	0	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	2	1	1	1	2	1	1	1	1	1	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	1	0	3	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	1	0	1	2	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	1	1	2	3	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	2	2	0	1	2	1	1	2	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
3	1	1	0	0	2	0	0	0	2	0	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
6	0	2	1	0	3	3	0	1	2	0	1	2	0	0	0	0	0	1	0	0	3	0	0	0	0	0	1	1	1	0	1	0	0	0
6	0	2	0	0	2	2	0	1	1	0	1	2	0	0	0	0	0	1	0	0	3	0	0	0	0	0	1	1	1	0	1	0	0	0
3	0	2	0	0	3	3	0	0	2	0	0	0	2	2	0	0	0	0	0	0	3	0	0	0	0	0	0	0	1	0	1	0	0	0
6	1	2	1	0	2	2	1	1	1	0	0	0	2	2	0	0	0	1	0	3	2	1	0	0	0	0	1	2	1	1	0	1	1	0
5	0	2	1	0	1	1	1	0	1	0	0	0	2	2	0	0	0	1	0	3	2	1	0	0	0	0	1	2	0	0	0	0	0	0
5	0	2	1	0	2	2	1	0	2	0	0	0	2	2	0	0	0	1	0	3	2	1	0	0	0	0	1	2	0	0	0	0	0	0
3	1	2	2	0	3	2	0	1	1	0	1	2	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	2	2	0	1	3	0	1	1	0	1	2	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	2	2	0	3	0	0	1	1	0	1	2	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	2	2	0	2	2	0	1	1	0	1	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
5	0	2	2	0	2	1	1	0	0	0	1	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	1	2	2	0	3	2	1	1	2	0	1	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	2	1	0	1	2	1	0	0	0	1	2	0	1	0	0	0	1	0	3	2	0	1	0	0	0	1	1	0	0	0	0	0	0
6	1	2	2	0	3	1	0	1	2	0	1	2	0	1	0	0	0	1	0	3	2	0	1	0	0	0	1	1	0	0	0	0	0	0
5	0	2	1	0	2	0	0	0	1	0	1	2	0	1	0	0	0	1	0	3	2	0	1	0	0	0	1	1	0	0	0	0	0	0
5	0	2	2	-1	3	3	-1	-1	0	0	0	-1	-1	-1	-1	-1	-1	1	-1	0	1	1	0	0	0	0	1	2	1	1	1	1	1	-1
5	0	2	2	-1	3	3	-1	-1	0	0	0	-1	-1	-1	-1	-1	-1	1	-1	0	1	1	0	0	0	0	1	2	1	1	1	1	1	-1
6	0	2	2	-1	2	3	-1	-1	1	0	0	-1	-1	-1	-1	-1	-1	1	-1	0	1	1	0	0	0	0	1	2	1	1	1	1	1	-1
5	-1	2	2	-1	1	3	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	1	-1	0	1	1	0	0	0	0	1	2	1	1	1	1	1	-1
6	-1	2	2	-1	1	3	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	1	-1	0	1	1	0	0	0	0	1	2	1	1	1	1	1	-1
5	-1	2	2	-1	2	3	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	1	-1	0	1	1	0	0	0	0	1	2	1	1	1	1	1	-1
6	-1	2	2	-1	2	3	-1	-1	-1	0	0	-1	-1	-1	-1	-1	-1	1	-1	0	1	1	0	0	0	0	1	2	1	1	1	1	1	-1
3	-1	-1	-1	-1	3	2	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	1	0	-1	1	-1	2
4	-1	-1	-1	-1	2	1	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	1	0	-1	1	-1	2
3	-1	-1	-1	-1	3	2	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	1	0	-1	1	-1	2
3	-1	-1	-1	-1	2	1	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	1	0	-1	1	-1	2
4	-1	-1	-1	-1	3	2	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	1	0	-1	1	-1	2
3	-1	-1	-1	-1	2	1	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	1	0	-1	1	-1	2
3	-1	-1	-1	-1	2	1	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	1	0	-1	1	-1	2
3	-1	-1	-1	-1	2	2	-1	-1	-1	1	1	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	2	-1	1	0	-1	1	-1	2
0	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	1	0	2	2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	1	0	2	2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	2	2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
1	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	0	2	2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	2	2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	0	2	2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	3	-1	-1	-1	-1	1	0	2	2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	-1	-1	-1	-1	-1	2	-1	-1	-1	-1	1	0	2	2	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
2	1	-1	0	-1	1	3	-1	-1	-1	1	1	2	1	1	0	1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	3	-1	-1	-1	-1	-1	-1	1
2	1	-1	0	-1	0	0	-1	-1	-1	1	1	0	2	2	0	1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	3	-1	-1	-1	-1	-1	-1	1
1	1	-1	0	-1	1	3	-1	-1	-1	1	1	2	1	1	0	1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	3	-1	-1	-1	-1	-1	-1	1
0	1	-1	0	-1	0	0	-1	-1	-1	1	1	0	2	2	0	1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	3	-1	-1	-1	-1	-1	-1	1

Sample output of Random selection/Approximate Evaluation
For Test Case 1

```
  0.3023  0.3598  0.2760  0.4039  0.3023  0.0727  0.4106  0.1414
SELECTED:
   7   5   4   2   6
  0.2872  0.3418  0.2622  0.3837  0.2872  0.0691  0.3901  0.1343
SELECTED:
   1   3   7   4   8   5   2   6
  0.2728  0.3247  0.2491  0.3645  0.2728  0.0656  0.3706  0.1276
SELECTED:
   2   7   5   4   8   1   3   6
  0.2592  0.3085  0.2366  0.3463  0.2592  0.0624  0.3521  0.1212
SELECTED:
   4   3   1   5   2   6   7
  0.2462  0.2931  0.2248  0.3290  0.2462  0.0592  0.3345  0.1151
SELECTED:
   3   1   4   7   8   5   2   6
  0.2339  0.2784  0.2136  0.3125  0.2339  0.0563  0.3177  0.1094
SELECTED:
   5   7   3   8   4   2   1   6
  0.2222  0.2645  0.2029  0.2969  0.2222  0.0535  0.3019  0.1039
SELECTED:
   4   5   3   2   7   8   1   6
  0.2111  0.2513  0.1927  0.2821  0.2111  0.0508  0.2868  0.0987
SELECTED:
   5   3   7   1   8   2   4   6
  0.2005  0.2387  0.1831  0.2680  0.2005  0.0482  0.2724  0.0938
SELECTED:
   5   2   7   3   4   6
  0.1905  0.2916  0.2340  0.3210  0.2524  0.0773  0.3255  0.0891
SELECTED:
   2   4   1   5   7   3   6
  0.1935  0.2913  0.2358  0.3195  0.2535  0.0813  0.3237  0.0846
SELECTED:
   3   7   8   6
  0.1838  0.2768  0.3509  0.3035  0.2408  0.1666  0.4381  0.1713
SELECTED:
   3   7   8   2   1   4   5   6
  0.1746  0.2629  0.3333  0.2883  0.2288  0.1583  0.4162  0.1627
SELECTED:
   5   4   7   8   1   6
  0.1891  0.2498  0.3167  0.3011  0.2430  0.1726  0.4230  0.1771
SELECTED:
   2   7   3   6
  0.1797  0.3310  0.3972  0.2860  0.2308  0.2492  0.4963  0.1683
SELECTED:
   4   3   7   6
  0.1707  0.3145  0.4709  0.3654  0.2193  0.3285  0.5593  0.1599
```

Procedure file for External Random Algorithm

```
.PROC,LOOP,FN=MYDATA,FO=OUTPUT,N=5.
RFL,125000.
SETTL,25.
USE,GEM01/UN=3RSMRSM.
USE,PROMGO,EVLGO.
IF(.NOT.FILE(PROMGO,AS)) GET,PROMGO.
IF(.NOT.FILE(EVLGO,AS)) GET,EVLGO.
IF(.NOT.FILE(GEM01,AS)) GET,GEM01/UN=3RSMRSM.
PROMGO,FN,FO/I_N.
WHILE,FILE(CONTINU,AS),A.
RETURN,GEMOUT.
GEM01,GEMINP,GEMOUT.
RETURN,GEMINP,EVALUE,CONTINU.
EVLGO,GEMOUT,EVALUE.
PROMGO,FN,FO/C.
ENDW,A.
$REVERT.
```

Program for Evaluation of Gem Output

```
PROGRAM EVAL(INPUT,OUTPUT);
(* THIS PROGRAM DOES AN EVALUATION OF THE OUTPUT OF THE GEM
   PROCEDURE. IT BASICALLY COUNTS SELECTORS AND TERMS IN THE
   OUTPUT-HYPOTHESES. APPLYING TO THE 'PRINCIPLE OF PARSIMONY',
   WE FEEL THAT THE FEWER THE NUMBER OF SELECTORS IN THE RULES
   GENERATED BY GEM, THE BETTER.        *)

CONST
      BS       = 80;
VAR
      I,N,T    : INTEGER;
      CH       : CHAR;
      BUF      : ARRAY[1..BS] OF CHAR;

BEGIN
      N:=0;
      T:=0;
       REPEAT READ(INPUT,CH); UNTIL CH='#';
      REPEAT
         READLN(INPUT);(* NOW COUNT THE TERMS IN A CLASS *)
         REPEAT(* READ A CARD INTO BUF *)
            FOR I:= 1 TO BS DO
               IF EOLN(INPUT) THEN BUF[I]:=' ' ELSE READ(INPUT,BUF[I]);
            READLN;(* NOW COUNT THE NUMBER OF NON-ASKERISKS ON THE LINE *)
            I:=1;
            N:=N-1;
            REPEAT
               WHILE (I<BS) AND (BUF[I]<>' ') DO I:=I+1;
               WHILE (I<BS) AND (BUF[I]=' ') DO I:=I+1;
               IF (BUF[I]<>'*') AND (BUF[I]<>' ') THEN N:=N+1;
               UNTIL I=BS;
          T:=T+1;
            UNTIL EOLN(INPUT);
    REPEAT READ(INPUT,CH) UNTIL (CH='#') OR EOF(INPUT);
        UNTIL EOF(INPUT);
      WRITELN(N,' ',T);
      END.
```

```
PROGRAM FS(INPUT,OUTPUT);     (*$I'RANDOM'*)
(*************************************************************
 RELEVANT VARIABLE SELECTION ALGORITHMS
            JORDAN POLLACK
            MAY, 1982

THERE ARE SEVERAL RVS ALGORITHMS AVAILABLE WITHIN THIS CODE.
THEY MAY BE INVOKED BY THE USE OF OPTIONS IN THE CALLING STATEMENT.
1- A SIEVE APPROACH WHEREBY THE VARIABLES ARE SORTED BY INITIAL
   PROMISE AND THEN DISCARDED ONE AT A TIME,  MAINTAINING ENOUGH
   INFORMATION FOR A DISCRETE COVER BY AQVAL PROGRAMS. THIS MAY BE
   INVOKED WITH THE 'S' OPTION:
       PROMGO,INFILE,OUTFILE/S

2- A GREEDY ALGORITHM WHICH CHOOSES ONE VARIABLE AT A TIME MAXIMIZING
   THE INFORMATION CONTENT (THUS MINIMIZING THE NUMBER OF VARIABLES
   SELECTED) MAY BE INVOKED WITH THE 'G' OPTION.

3- USER SELECTION OF KEYS CAN BE ACCOMPLISHED WITH THE 'K' OPTION
   AND THE EXISTENCE OF A EXTERNAL FILE CALLED 'KEYS', WHICH SHOULD
   CONTAIN AN INTEGER EXPRESSING THE NUMBER OF VARIABLES TO BE
   SELECTED, FOLLOWED BY THE INDICES OF THOSE KEYS.

4- A RANDOM SELECTION/EVALUATION APPROACH WHICH USES AN APPROXIMATE
   EVALUATION FUNCTION MAY BE INVOKED WITH THE 'RN' OPTION, WHERE N
   IS THE NUMBER OF ITERATIONS TO BE DONE. THE SELECTION PROCEDURE
   MAY BE CONTROLLED WITH THE 'FN' OPTION, WHICH SPECIFIES THE
   MINIMUM NUMBER OF FEATURES TO BE SELECTED (DEFAULT 1).

5- A RANDOM SELECTION/EVALUATION APPROACH WHICH USES GEM AS
   PART OF THE LOOP MAY BE INVOKED WITH THE 'IN' AND 'C' OPTIONS.
   THE 'IN' OPTION SPECIFIES HOW MANY INTERATIONS TO PERFORM, THE
   'C' OPTION SPECIFIES THE CONTINUATION OF THE ITERATIVE APPROACH.
   THIS ALGORITHM USES SEVERAL EXTERNAL FIXED-NAME FILES: IT CREATES
   'GEMINP' WHICH CONTAINS THE SELECTED INPUT DATA FOR GEM; IT READS
   AND WRITES A FILE CALLED 'GLOBALS' WHICH CONTAINS PROGRAM STATE
   INFORMATION (SINCE CYBER LACKS A CO-ROUTINE FACILITY); IT
   CREATES A FILE CALLED 'CONTINU' EVERY ITERATION EXCEPT THE LAST,
   WHICH ACTS AS A FLAG TO A JCL ROUTINE, AND THE 'C' OPTION EXPECTS
   A FILE CALLED 'EVALUE' WHICH SHOULD CONTAIN AN INTEGER WHICH
   REPRESENTS THE EVALUATION OF THE PREVIOUSLY SELECTED VARIABLE
   SET; THE SMALLER THE INTEGER, THE BETTER THE SET.
       THE 'MN' OPTION IS A CONVENIENT WAY TO PASS A MAXSTAR PARAMETER
   TO GEM; ITS DEFAULT IS 1.

6- FINALLY, A TECHNIQUE FOR INCLUDING ALL THE VARIABLES IS PROVIDED
   WITH THE 'A' OPTION.
 NOTE: WITH THE EXCEPTION OF THE 'MN' AND 'FN' OPTIONS, THE OTHERS
 ARE INTENDED TO BE USED SEPARATELY!!!!

 SEE THE PROCEDURE LOOP IN PROCFIL/UN=3DOOPLZ FOR EXAMPLE OF HOW
 TO USE THE 'C' AND 'I' OPTIONS.
 *************************************************************)
```

Main Program

```
CONST

      MAXNE      = 75;
      MAXNC      = 20;
      MAXNV      = 35;
 (*************************************************************
   THE CONSTANTS ARE THE CURRENT LIMIT ON PROBLEM SIZE:
        MAXNE IS THE MAXIMUM NUMBER OF EVENTS
        MAXNC "   "     "       "      " CLASSES
        MAXNV "   "     "       "      " VARIABLES(FEATURES)
 *************************************************************)
TYPE
      BSTRESULT = (EQ,LT,GT);
      SETTING   = PACKED RECORD
                  CASE SWITCH: BOOLEAN OF
TRUE:   (ONOFF     : CHAR);
FALSE:  (SIZE      : 0..999999)
                  END;
 (*************************************************************
    THE BSTRESULT TYPE IS FOR THE BINARY TREE SEARCH
      AND INSERTION ALGORITHMS
    THE SETTING TYPE IS FOR THE 'OPTIONS' PACKAGE
 *************************************************************)
```

Main Program

```
VAR

     DATA       : ARRAY[1..MAXNE,1..MAXNV] OF INTEGER;
     CLSIZE     : ARRAY[1..MAXNC] OF INTEGER;
     TABLE      : ARRAY[1..MAXNE,1..MAXNC] OF INTEGER;
     FIX        : REAL;
     BEST,WORST,EVAL,RCNT,SUM,AVG: INTEGER;
     NUMKEYS,NEXT,I: INTEGER;
     NC,NV,NE   : INTEGER;
     NF,BNKEYS:INTEGER;
     MUTI,SCALE: REAL;
     KEYS,BKEYS: ARRAY[1..MAXNV] OF INTEGER;
     CLASS      : ARRAY[1..MAXNE] OF INTEGER;
     TREEX,TREER,TREEL,SUMEVNTS: ARRAY[1..MAXNE] OF INTEGER;
     PROMISE    : ARRAY[1..MAXNV] OF REAL;
     VRSIZE     : ARRAY[1..MAXNV] OF INTEGER;
     OPTS       : SETTING;
     INFILE:TEXT; COUNT,MAXSTAR:INTEGER;

(*****************************************************************
THE VARIABLES, WHICH ARE ALMOST ALL USED AS GLOBALS BY THE
SUBROUTINES ARE AS FOLLOWS:
       DATA[I,J] IS THE VALUE OF FEATURE J OF EVENT I
       CLSIZE[I] IS THE SIZE OF THE ITH CLASS
       CLASS[I] IS THE CLASS OF THE ITH EVENT
THE INPUT FILE SHOULD BE ORGANIZED AS FOLLOWS:
   NUMBER-OF-CLASSES  SIZE-OF-CLASS-1 SIZE-OF-CLASS-2 ETC.
   NUMBER-OF-EVENTS  NUMBER-OF-VARIABLES
   FOLLOWED BY THE DATA
EXAMPLE:
  3 3 1 2
  6 5
  1 2 3 2 1          )
  2 1 1 2 1          ) CLASS 1 EVENTS
  3 2 1 3 2          )
  3 1 3 1 3      ) CLASS 2
  2 3 2 2 2    )
  2 1 2 1 3    )CLASS 3 EVENTS

THE SUBPROGRAMS BUILDIT AND TESTIT BUILD A EVENT/CLASS
CROSS-PRODUCT USING A BINARY TREE. THE FOLLOWING VARIABLES
ARE USED:
      TABLE[I,J] IS (EVENTUALLY) THE NUMBER OF INCIDENCES OF
                 EVENT-I IN CLASS-J.
      TREEX[I]  IS THE INDEX OF EVENT-I INTO DATA
      TREEL[I] AND TREER[I] ARE THE LEFT AND RIGHT POINTERS.
      NUMKEYS IS A INTEGER DENOTING THE NUMBER OF FEATURES
              PARTICIPATING IN THE CURRENT COMPUTATION, AND
      KEYS[I] IS THE INDEX VECTOR USED TO SELECT FEATURES.
*****************************************************************)
```

Main Program

```
(* THE OPTION PACKAGE IS INCLUDED, AS DESCRIBED IN
   WRITEUP,PASCAL -- PASLIB  *)
FUNCTION OPTION
     (NAME       : CHAR;
      VAR S      : SETTING): BOOLEAN;
      EXTERN;


 (* READDATA READS IN THE VARIABLES NC,NE,NV AND THE DATA
    ARRAY. IT ALSO SETS UP THE CLSIZE AND CLASS VECTORS. *)

PROCEDURE READDATA;

VAR
      I,J,K,COUNT,INDEX: INTEGER;
BEGIN
      READ(NC);
      FOR I := 1 TO NC DO READ(CLSIZE[I]);
      READ(NE,NV);
      FOR I:=1 TO NV DO VRSIZE[I]:=0;
      INDEX := 1;
      FOR I := 1 TO NC DO BEGIN
         FOR COUNT:=1 TO CLSIZE[I] DO BEGIN
            CLASS[INDEX]:= I;
            FOR J := 1 TO NV DO BEGIN
               READ(DATA[INDEX,J]);
               IF (DATA[INDEX,J]+1)>VRSIZE[J] THEN VRSIZE[J]:=DATA[INDEX,J]+1;
               END;
            INDEX := INDEX+1;
            END;
         END;
      END;
```

Main Program

```
 (*  COMPARE IS THE INDEXED VECTOR ELEMENT BY ELEMENT
     COMPARISON NEEDED TO COMPARE EVENTS. ROOT AND
     INDEX ARE BOTH POINTERS INTO THE BINARY SEARCH
     TREE/INCIDENCE MATRIX DATA STRUCTURE.          *)

FUNCTION COMPARE
     (INDEX,ROOT: INTEGER): BSTRESULT;

VAR
      I          : INTEGER;
      FLAG       : BSTRESULT;
BEGIN
      I:=1;       FLAG:=EQ;
      WHILE (I<=NUMKEYS)AND(FLAG=EQ) DO BEGIN
         IF DATA[TREEX[ROOT],KEYS[I]]= DATA[INDEX,KEYS[I]] THEN I:=I+1
         ELSE IF DATA[TREEX[ROOT],KEYS[I]]< DATA[INDEX,KEYS[I]] THEN FLAG:=LT
                                                                ELSE FLAG:=GT
         END;
      COMPARE:=FLAG
      END;
```

Main Program

```
(* INSERT DOES AN EVENT INSERTION INTO THE BINARY SEARCH
   TREE/INCIDENCE MATRIX DATA STRUCTURE. ROOT IS THE CURRENT
   NODE OF THE BST, AND INDEX IS AN INDEX INTO THE DATA
   ARRAY OF THE EVENT WE ARE TRYING TO INSERT. NEXT IS THE
   "FREE-STORAGE" POINTER TO THE UNUSED NODES OF THE BINARY
   TREE, AND IS INCREMENTED IF THE EVENT IS NEW.  INSERT
   RETURNS WITH ROOT POINTING TO THE EVENT NODE IN THE
   TABLE/TREE STRUCTURE.                                    *)


PROCEDURE INSERT
     (VAR ROOT,NEXT: INTEGER;
      INDEX      : INTEGER);

VAR
      TEST       : BSTRESULT;
BEGIN
      REPEAT
         TEST:=COMPARE(INDEX,ROOT);
         IF TEST=LT THEN
            IF TREEL[ROOT]=0 THEN BEGIN
               NEXT:=NEXT+1;       TREEL[ROOT]:=NEXT;       ROOT:=NEXT;
               TREEX[NEXT]:=INDEX
               END
            ELSE ROOT:=TREEL[ROOT];
         IF TEST=GT THEN
            IF TREER[ROOT]=0 THEN BEGIN
               NEXT:=NEXT+1;       TREER[ROOT]:=NEXT;       ROOT:=NEXT;
               TREEX[ROOT]:=INDEX
               END
            ELSE ROOT:=TREER[ROOT];
         UNTIL TEST=EQ;
      END;
```

Main Program

```
(* PROCEDURE BUILDIT COMPUTES THE MUTUAL INFORMATION (I.E
   PROMISE) FUNCTION FOR EITHER SINGLE FEATURES OR MULTIPLE
   FEATURES (GIVEN THE DATA IN KEYS/NUMKEYS VECTOR). THE
   FUNCTION IS NORMALIZED TO BETWEEN 0 AND 1.0 BY USING
   THE SCALE VARIABLE (SEE SETSCALE). FIRST THE BINARY
   TREE/INCIDENCE MATRIX IS CLEARED; THEN IT IS COMPUTED;
   FINALLY, THE REDUCTION TO A MAGIC NUMBER IS PERFORMED.
   THE ANSWER (TYPE REAL) IS LEFT IN THE VARIABLE MUTI.     *)

PROCEDURE BUILDIT
     (VAR NEXT  : INTEGER;
      VAR MUTI  : REAL);

VAR
      INDEX,ROOT,J: INTEGER;
      PX,PW,PWX : REAL;
BEGIN(*FIRST CLEAR OUT ALL TABLES*)
      FOR INDEX := 1 TO NE DO BEGIN
         FOR J := 1 TO NC DO TABLE[INDEX,J]:=0;
         TREEX[INDEX]:=0;      TREEL[INDEX]:=0;      TREER[INDEX]:=0;
         SUMEVNTS[INDEX]:=0
         END;
      MUTI:=0.0;(*NOW PROCEED TO FILL THE TABLE AND TREE*)
      NEXT:=1;       (*THIS IS THE POINTER TO NEXT NODE IN BST*)
      TREEX[1]:=1;       (*INITIALIZE BINARY TREE*)
      FOR INDEX := 1 TO NE DO BEGIN
         ROOT:=1;       (*START AT THE TOP*)
         INSERT(ROOT,NEXT,INDEX);(*THIS MODIFIES ROOT & NEXT*)
         TABLE[ROOT,CLASS[INDEX]]:=TABLE[ROOT,CLASS[INDEX]]+1
         END;(*NOW COMPUTE THE SUMS OF EVENTS*)
      FOR INDEX := 1 TO NEXT DO
         FOR J := 1 TO NC DO
         SUMEVNTS[INDEX]:=SUMEVNTS[INDEX]+TABLE[INDEX,J];
    (*NOW CMPUTE MUTUAL INFORMATION*)
      FOR INDEX := 1 TO NEXT DO BEGIN
         PX:=SUMEVNTS[INDEX]/NE;
         FOR J:= 1 TO NC DO BEGIN
            PW:=CLSIZE[J]/NE;      PWX:=TABLE[INDEX,J]/NE;
            IF TABLE[INDEX,J] <> 0 THEN BEGIN
               MUTI:=MUTI+PWX*LN(PWX/(PX*PW));
               END;
            END;
         END;
      MUTI:=MUTI/SCALE; (*NORMALIZE TO 1.0*)
      END;
```

Main Program

```
(* THE FUNCTION TESTIT RETURNS A TRUE VALUE IF ALL THE EVENTS
   IN EACH CLASS ARE DISTINCT FROM ALL EVENTS IN ALL OTHER
   CLASSES.  THIS IS A SHORT CUT TO FIND IF THE SELECTED
   FEATURES COULD BE USED TO FIND A DISJOINT COVER; IT IS
   COMPUTATIONALLY CHEAPER THAN BUILDIT.                    *)

FUNCTION TESTIT: BOOLEAN;

VAR
     FLAG      : BOOLEAN;
     INDEX,J,ROOT: INTEGER;

BEGIN(* FIRST CLEAR OUT DATA STRUCTURE *)
     FOR INDEX := 1 TO NE DO BEGIN
        TREEX[INDEX]:=0;      TREEL[INDEX]:=0;      TREER[INDEX]:=0;
        END;(* NOW INSERT STUFF INTO THE BINARY TREE UNTIL A CLASS
         CLASH IS DETECTED. *)
     NEXT:=1;      TREEX[1]:=1;      FLAG:=TRUE;      INDEX:=1;
     WHILE (INDEX<=NE)AND FLAG DO BEGIN
        ROOT:=1;      INSERT(ROOT,NEXT,INDEX);
        IF CLASS[TREEX[ROOT]]<>CLASS[INDEX] THEN FLAG:=FALSE;
        INDEX:=INDEX+1;
        END;
     TESTIT:=FLAG;
     END;
```

Main Program

```
(*  PRINTIT IS A DEBUG ROUTINE USED TO PRINT OUT THE INCIDENCE
    ARRAY. IT SEEMS TO HAVE OUTLIVED ITS USEFULNESS, BUT IS
    LEFT AROUND ANYHOW.                                        *)

PROCEDURE PRINTIT;

VAR
      I,J,K     : INTEGER;
BEGIN
      WRITELN;
      FOR I:= 1 TO ROUND((4*NUMKEYS-4)/2+1) DO WRITE(' ');
      WRITE('KEYS');
      FOR I:=1 TO ROUND((4*NUMKEYS-4)/2+(5*NC-7)/2) DO WRITE(' ');
      WRITE('CLASSES');
      FOR I:=1 TO ROUND((5*NC-7)/2+1) DO WRITE(' ');
      WRITE('EVENTSUM');      WRITELN;
      FOR I := 1 TO NUMKEYS DO WRITE(KEYS[I]:4);
      WRITE('/');
      FOR I:= 1 TO NC DO WRITE(I:5);
      WRITELN;
      FOR I:=1 TO (4*NUMKEYS+5*NC+10) DO WRITE('_');
      WRITELN;
      FOR I := 1 TO NEXT DO BEGIN
         FOR J:=1 TO NUMKEYS DO WRITE(DATA[TREEX[I],KEYS[J]]:4);
         WRITE('/');
         FOR J := 1 TO NC DO WRITE (TABLE[I,J]:5);
         WRITE(SUMEVNTS[I]:6);      WRITELN
         END;
      END;
 (* SETSCALE SETS A GLOBAL VARIABLE CALLED SCALE WHICH IS THE
    BASE FOR THE LOGARITHM IN A MUTUAL INFORMATION MEASURE. *)

PROCEDURE SETSCALE
     (VAR SCALE : REAL);

VAR
      I         : INTEGER;
BEGIN
      SCALE:=0.0;
      FOR I:=1 TO NC DO SCALE:=SCALE+CLSIZE[I]/NE*LN(NE/CLSIZE[I]);
      END;
```

Main Program

```
(* THE PROCEDURE SIEVE IS MY FIRST NAIVE ATTEMPT AT FEATURE
   SELECTION; IT RUNS QUITE FAST SINCE IT MAKES ONE PASS
   THRCUGH THE FEATURES THROWING AWAY WHATEVER IT COULD
   WHILE MAINTAINING ENOUGH INFORMATION FOR A DISJOINT
   COVER. THE NUMKEYS/KEYS STRUCTURE MUST BE INITIALIZED
   BEFOR SIEVE IS CALLED, AND THEN IT GETS MODIFIED.     *)

PROCEDURE SIEVE;

VAR
      INDEX,J,HOLD,NEXT: INTEGER;
BEGIN
      INDEX:=1;
      WHILE INDEX<=NUMKEYS DO BEGIN
         NUMKEYS:=NUMKEYS-1;       HOLD:=KEYS[INDEX];
         FOR J:=INDEX TO NUMKEYS DO KEYS[J]:=KEYS[J+1];
         IF NOT TESTIT THEN BEGIN
            NUMKEYS:=NUMKEYS+1;
            FOR J:=NUMKEYS DOWNTO (INDEX+1) DO KEYS[J]:=KEYS[J-1];
            KEYS[INDEX]:=HOLD;       INDEX:=INDEX+1;
            END;
         END;
      END;

(* THE OPEN AND CLOSE FUNCTIONS ARE INCLUDE HERE FOR NAMED
   EXTERNAL FILE MANIPULATION.  SEE PASLIB DOCUMENT *)

PROCEDURE OPEN
     (VAR F      : TEXT;
      N          : ALFA;
      OPENWRITE  : BOOLEAN);
      EXTERN;

PROCEDURE CLOSE
     (VAR F      : TEXT);
      EXTERN;
```

Main Program

```
(* PROCEDURE CREATEF BUILDS A FILE AS INPUT TO GEM.
   IT USES THE GLOBAL VARIABLES NUMKEYS/KEYS WHICH SELECT
   WHICH VARIABLES TO INCLUDE; VRSIZE, WHICH CONTAINS THE
   NUMBER OF NOMINAL LEVELS FOR EACH VARIABLE; AND MAXSTAR,
   WHICH IS SET USING THE 'MN' OPTION  *)
PROCEDURE CREATEF;

VAR
      FNAME     : TEXT;
      FILENAME  : ALFA;
      I,J,CL,COUNT,NEXT: INTEGER;
      MUTI      : REAL;


BEGIN
      BUILDIT(NEXT,MUTI);
(* OPEN AND INITIALIZE THE NAMED FILE 'GEMINP'*)
      FILENAME:='GEMINP    ';
      OPEN(FNAME,FILENAME,TRUE);
      REWRITE(FNAME);
(* WRITE THE PARAMETERS 'ECHO' AND 'MAXSTAR' TO THE FILE*)
      WRITELN(FNAME,'PARAMETERS');
      WRITELN(FNAME,'MAXSTAR ECHO');
      WRITELN(FNAME,MAXSTAR,' P');    WRITELN(FNAME);
(* WRITE THE VARIABLE DESCRIPTIONS TO THE FILE *)
      WRITELN(FNAME,'VARIABLES');
      WRITELN(FNAME,'NAME TYPE LEVELS');
      FOR I := 1 TO NUMKEYS DO
         WRITELN(FNAME,' V',KEYS[I]:1, ' NOM ',VRSIZE[KEYS[I]]);
      WRITELN(FNAME);
(* NOW WRITE THE CLASSES AND EVENTS TO THE FILE *)
      CL:=0;
      FOR I:=1 TO NEXT DO BEGIN
         IF CLASS[TREEX[I]]<>CL THEN BEGIN
            CL:=CLASS[TREEX[I]];      COUNT:=1;      WRITELN(FNAME);
            WRITELN(FNAME,'CLASS',CL:1,'-EVENTS');   WRITE(FNAME,' # ');
            FOR J:=1 TO NUMKEYS DO WRITE(FNAME,'   V',KEYS[J]:1);
            WRITELN(FNAME);
            END;
         WRITE(FNAME,COUNT:4);      COUNT:=COUNT+1;
         FOR J:=1 TO NUMKEYS DO WRITE(FNAME,DATA[TREEX[I],KEYS[J]]:4);
         WRITELN(FNAME);
         END;
(* FINALLY, CLOSE THE FILE *)
      CLOSE(FNAME);
      END;
```

Main Program

```
(* THE PROCEDURE SAVEGL REWRITES THE FILE NAMED 'GLOBALS' WITH
   THE VARIABLES AND ARRAYS NEEDED FOR CONTINUATION *)

PROCEDURE SAVEGL;

VAR OUTFILE:TEXT;
    FILENAME:ALFA;
    I:INTEGER;
BEGIN
(* OPEN THE FILE 'GLOBALS' FOR WRITING*)
    FILENAME:='GLOBALS   ';
    OPEN(OUTFILE,FILENAME,TRUE);
    REWRITE(OUTFILE);
(* WRITE OUT THE CURRENT SELECTED VARIABLE SET *)
    WRITE(OUTFILE,NUMKEYS,' ');
    FOR I:=1 TO NUMKEYS DO WRITE(OUTFILE,KEYS[I],' ');
    WRITELN(OUTFILE);
(* WRITE OUT THE CURRENT STATE OF THE PROMISE VECTOR*)
    FOR I:=1 TO NV DO WRITE(OUTFILE,PROMISE[I],' ');
    WRITELN(OUTFILE);
(* WRITE OUT THE BEST KNOWN VARIABLE SET *)
    WRITE(OUTFILE,BNKEYS,' ');
    FOR I:=1 TO BNKEYS DO WRITE(OUTFILE,BKEYS[I],' ');
    WRITELN(OUTFILE);
(* WRITE THE VARIABLES NEEDED FOR UPDATE AND LOOP CONTROL*)
    WRITELN(OUTFILE,BEST,' ',SUM,' ',RCNT,' ',COUNT);
    CLOSE(OUTFILE);
(*SET THE GLOBAL FILE CONTINUE AS TRUE*)
    FILENAME:='CONTINU   ';
    OPEN(OUTFILE,FILENAME,TRUE);
    REWRITE(OUTFILE);
    WRITELN(OUTFILE,'TRUE');
    CLOSE(OUTFILE);
END;
```

Main Program

```
(* PROCEDURE LOADGL LOADS THE VARIABLES THAT SAVEGL STORED IN
   EXTERNAL FILE 'GLOBALS' *)

PROCEDURE LOADGL;

VAR INFILE:TEXT;
    FILENAME:ALFA;
    I:INTEGER;
BEGIN
    FILENAME:='GLOBALS   ';
    OPEN(INFILE,FILENAME,FALSE);
    RESET(INFILE);
    READ (INFILE,NUMKEYS);
    FOR I:=1 TO NUMKEYS DO READ(INFILE,KEYS[I]);
    FOR I:=1 TO NV DO READ(INFILE,PROMISE[I]);
    READ(INFILE,BNKEYS);
    FOR I:=1 TO BNKEYS DO READ(INFILE,BKEYS[I]);
    READ(INFILE,BEST,SUM,RCNT,COUNT);
    CLOSE(INFILE);
END;
(* PROCEDURE INDEXSORT IS USED ONLY ONCE, BEFORE THE SIEVE PROCEDURE
   SO A BUBBLE SORT IS SUFFICIENT... IT DOES AN INDEX SORT WITH
   THE VARIABLE SET IN NUMKEYS/KEYS ON PROMISE. *)

PROCEDURE INDEXSORT;

VAR
      I,J,HOLD  : INTEGER;
BEGIN
      FOR I:=1 TO (NUMKEYS-1) DO BEGIN
         FOR J:=(I+1) TO NUMKEYS DO
            IF PROMISE[KEYS[I]]>=PROMISE[KEYS[J]] THEN BEGIN
               HOLD:=KEYS[I];      KEYS[I]:=KEYS[J];      KEYS[J]:=HOLD;
               END;
         END;
      END;
```

Main Program

```
(* PROCEDURE GREEDY IS THE HEART OF THE GREEDY ALGORITHM. IT SELECTS
   VARIABLES ONE AT A TIME BY VIRTUE OF HOW WELL THIS VARIABLE (IN
   CONJUNCTION WITH THE ALREADY SELECTED ONES) DISCRIMINATE THE DATA.
   TIES ARE RESOLVED BY THE INDIVIDUAL PROMISE OF THE CONTENDERS*)

PROCEDURE GREEDY;

VAR
      I,MAXI: INTEGER;
      MAXMUTI,MUTI: REAL;

PROCEDURE SWAP
     (I,J        : INTEGER);
VAR
      HOLD       : INTEGER;
      ROLD       : REAL;
BEGIN
      HOLD:=KEYS[I];      KEYS[I]:=KEYS[J];      KEYS[J]:=HOLD;
      ROLD:=PROMISE[I];   PROMISE[I]:=PROMISE[J];
      PROMISE[J]:=ROLD;
      END;

BEGIN
      FOR I:=1 TO NV DO KEYS[I]:=I;
      NUMKEYS:=0;      MAXMUTI:=0.0;
      REPEAT
         NUMKEYS:=NUMKEYS+1;      MAXI:=NUMKEYS;
         FOR I:=NUMKEYS TO NV DO BEGIN
            SWAP(I,NUMKEYS);      BUILDIT(NEXT,MUTI);
            IF (MUTI>MAXMUTI) OR ((MUTI=MAXMUTI) AND (PROMISE[I]>PROMISE[MAXI]))
               THEN BEGIN
               MAXMUTI:=MUTI;      MAXI:=I;
               END;
            SWAP(I,NUMKEYS);
            END;
         SWAP(NUMKEYS,MAXI);
         UNTIL MAXMUTI>0.9999;
      END;
```

Main Program

```
(*PROCEDURE SELECT DOES A WEIGHTED RANDOM SELECTION OF VARIABLES
  BASED ON THEIR PROMISE UNTIL A MINCOMPLETE SET IS FOUND*)
PROCEDURE SELECT(N:INTEGER);

VAR
      I,HOLD     : INTEGER;
      SUM        : REAL;
BEGIN
      NUMKEYS:=0;       SETRAN(CLOCK);
      FOR I:=1 TO NV DO KEYS[I]:=I;
      REPEAT
         NUMKEYS:=NUMKEYS+1;
         SUM:=0.0;
         FOR I:=NUMKEYS TO NV DO SUM:=SUM+PROMISE[KEYS[I]];
         SUM:=SUM*RAN;       I:=NUMKEYS-1;
         REPEAT I:=I+1; SUM:=SUM-PROMISE[KEYS[I]]; UNTIL SUM<=0.0;
         HOLD:=KEYS[I];       KEYS[I]:=KEYS[NUMKEYS];       KEYS[NUMKEYS]:=HOLD;
         UNTIL TESTIT AND (NUMKEYS>=N);
      WRITELN('SELECTED:');
      FOR I:=1 TO NUMKEYS DO WRITE(KEYS[I]:4);
      WRITELN;
      END;
```

Main Program

```
(* PROCEDURE UPDATE UPDATES THE PROMISE VECTOR BASED ON THE RESULT
   OF THE EVALUATION OF FEATURES. CURRENTLY, IT DECAYS THE PROMISE
   OF ALL THE VARIABLES, THEN BASED ON HOW THE EVALUATION FALLS
   BETWEEN THE BEST AND AVERAGE CASES, IT TAKES FROM THE SQUARE ROOT
   TO THE FIRST ROOT OF THE PROMISE OF EACH KEY SELECTED. *)

PROCEDURE UPDATE(VAR BEST:INTEGER; AVG,EVAL:INTEGER);

VAR I:INTEGER; FIX:REAL;
BEGIN
            FOR I:=1 TO NV DO PROMISE[I]:=0.95*PROMISE[I];
            IF EVAL<BEST THEN
            BEGIN
                 BEST:=EVAL;
                 BNKEYS:=NUMKEYS;
                 FOR I:=1 TO NUMKEYS DO BKEYS[I]:=KEYS[I];
            END;
            IF (EVAL<AVG) AND (AVG<>BEST) THEN
            BEGIN
                 FIX:=(BEST-EVAL)/(BEST-AVG)*0.7+0.3;
                 FOR I:=1 TO NUMKEYS DO
                     PROMISE[KEYS[I]]:=EXP(FIX*LN(PROMISE[KEYS[I]]));
           END;
            FOR I:=1 TO NV DO WRITE(PROMISE[I]:8:4);
            WRITELN;
END;
(*INITPROM SETS UP THE PROMISE VECTOR BY CALLING BUILDIT FOR EACH
  VARIABLE INDIVIDUALLY*)

PROCEDURE INITPROM;

VAR I:INTEGER;
    NEXT:INTEGER;
    MUTI:REAL;
BEGIN
      NUMKEYS:=1;                (*COMPUTE MI FOR EACH VAR*)
      FOR I := 1 TO NV DO BEGIN
         KEYS[1]:=I;      BUILDIT(NEXT,MUTI);      PROMISE[I]:=MUTI;
         WRITE(MUTI:8:4);
         END;
       WRITELN;
END;
```

Main Program

```
(* MAIN PROGRAM BEGINS HERE FINALLY! *)
BEGIN

      READDATA;                  (*READ DATA IN*)
      SETSCALE(SCALE);           (*SETUP SCALE FOR MI LOG*)

(* THE M OPTION CONTROLLS THE MAXSTAR PARM IN GEM INPUT FILES*)
IF OPTION('M',OPTS) THEN MAXSTAR:=OPTS.SIZE ELSE MAXSTAR:=1;

(* TEST HERE FOR THE 'A' OPTION, JUST BUILDING A GEM INPUT FILE*)
      IF OPTION('A',OPTS) THEN BEGIN
         NUMKEYS:=NV;
         FOR I:=1 TO NUMKEYS DO KEYS[I]:=I;
         CREATEF;
         END;


      IF OPTION('G',OPTS) THEN BEGIN
         INITPROM;
         GREEDY;      CREATEF;
         END;

 (* THE F OPTION SELECTS THE NUMBER OF FEATURES TO SELECT IN
    EITHER RANDOM ALGORITHM *)
   IF OPTION('F',OPTS) THEN NF:=OPTS.SIZE ELSE NF:=0;

       IF OPTION('S',OPTS) THEN BEGIN
          INITPROM;
          FOR I:=1 TO NV DO KEYS[I]:=I;
          NUMKEYS:=NV;      INDEXSORT;      SIEVE;      CREATEF;
          END;
```

Main Program

```
(* THE 'R' OPTION IS FOR RANDOM SELECTION WITH APPROXIMATE
   EVALUATION. THE RANDOM SELECTION IS WEIGHTED BY THE PROMISE
   OF EACH VARIABLE COMPUTED BY BUILDIT; SELECTIONXCONTINUES
   UNTIL A MINIMUM COMPLETE SET IS FOUND (I.E. TESTIT);
   THE APPROXIMATE EVALUATION IS SIMPLY THE SIZE OF THE
   EVENT REFERENCE TABLE -- NUMKEYS*NEXT; AND THE ADJUSTMENT
   OF WEIGHTS IS INEXPLICABLE.  *)
      IF OPTION ('R',OPTS) THEN BEGIN
         INITPROM;
         BEST:=MAXINT;
         SUM:=0;
         FOR RCNT:=1 TO OPTS.SIZE DO BEGIN
         SELECT(NF);
            EVAL:=NUMKEYS*NEXT;
            SUM:=SUM+EVAL;
            AVG:=SUM DIV RCNT;
           UPDATE(BEST,AVG,EVAL);
            END;
            NUMKEYS:=BNKEYS;
            FOR I:=1 TO BNKEYS DO KEYS[I]:=BKEYS[I];
            CREATEF;
         END;

(* THE 'I' OPTION INITIALIZES THE PROGRAM FOR RANDOM SELECTION
   AND EVALUATION USING GEM AND AN EXTERNAL EVAL FUNCTION. *)
IF OPTION('I',OPTS) THEN
BEGIN
     RCNT:=OPTS.SIZE;
     SUM:=0;
     COUNT:=1;
     BEST:=MAXINT;
     INITPROM;
     SELECT(NF);
     CREATEF;
     SAVEGL;
END;
```

Main Program

```
(*THE 'C' OPTION CONTINUES THE EXTERNAL RANDOM SELECTION *)
IF OPTION('C',OPTS) THEN
BEGIN
    LOADGL;
    IF COUNT<=RCNT THEN
    BEGIN
         OPEN(INFILE,'EVALUE     ',FALSE);
         RESET(INFILE);
         READ(INFILE,EVAL);
         CLOSE(INFILE);
         SUM:=SUM+EVAL;
         AVG:=SUM DIV COUNT;
         UPDATE(BEST,AVG,EVAL);
         SELECT(NF);
         CREATEF;
         COUNT:=COUNT+1;
         SAVEGL;
    END
    ELSE
       BEGIN
            NUMKEYS:=BNKEYS;
            FOR I:=1 TO BNKEYS DO KEYS[I]:=BKEYS[I];
            CREATEF;
       END;
END;

(* THE K OPTION IS FOR READING NUMKEYS AND KEYS FROM A FILE CALLED
   'KEYS' FOR USER EDIFICATION *)
IF OPTION('K',OPTS) THEN BEGIN
   OPEN(INFILE,'KEYS       ',FALSE);
   RESET(INFILE);
   READ(INFILE,NUMKEYS);
   FOR I:=1 TO NUMKEYS DO READ (INFILE,KEYS[I]);
   IF TESTIT THEN CREATEF ELSE
      WRITELN('KEYS GIVEN DO NOT FORM A COMPLETE SET');
   CLOSE(INFILE);
   END;
END.
```

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-83-902	2.	3. Recipient's Accession No.

4. Title and Subtitle

Relevant Variable Selectionfor Inductive Learning Programs

5. Report Date: January 1983

6.

7. Author(s): Jordan Pollack

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

Department of Computer Science
University of Illinois
Urbana, IL

10. Project Task Work Unit No.

11. Contract Grant No.

12. Sponsoring Organization Name and Address

National Science Foundation
Washington, DC

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

Four different techniques of relevant attribute selection for inductive learning systems are briefly described and experimentally compared. The first technique is to throw away variables until no more can be thrown away (sieve); the second is to select variables one at a time (greedy); the third and fourth involve iterative random selection of variables and evaluation of the selected set with feedback into the selection process. The third technique uses an approximate evaluation scheme while the fourth actually uses the GEM version of AQ.

17. Key Words and Document Analysis. 17a. Descriptors

feature selection
attribute selection
inductive learning
pattern recognition
machine learning

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 37
	20. Security Class (This Page UNCLASSIFIED	22. Price