Summarizing Data Sets for Classification[1]

Christopher W. Kinzig *and* Krishnaprasad Thirunarayan[2]
Department of Computer Science and Engineering
Wright State University, Dayton, Ohio-45435.
Email: tkprasad@cs.wright.edu
URL: www.cs.wright.edu/~tkprasad
Tel: (937)-775-5109
FAX: (937)-775-5133

Gary B. Lamont, Professor
Dept of Electrical and Computer Engineering
School of Engineering and Management
Air Force Institute of Technology
2950 P. St., WPAFB
Dayton, Ohio 4543
Email: gary.lamont@afit.edu
Tel: (937)-255-3450 (X-4718)
FAX: (937)-656-4055

Robert E. Marmelstein, Major, USAF (Ph.D.)
Deputy Chief, Information Systems Division (AFRL/IFS)
525 Brooks Road, Bldg 3 - Rm 1046
Rome, NY 13441-4505
Email: Robert.Marmelstein@rl.af.mil
Tel: (315)-330-1782
FAX: (315)-330-7083

This paper describes our approach and experiences with implementing a data mining system using genetic algorithms in C++. In contrast with earlier classification algorithms that tended to "tile" the data sets using some pre-specified "shapes", the proposed system is based on Marmelstein's work on determining *natural* boundaries for class homogeneous regions. These boundaries are further refined to construct a compact set of simple data mining rules for classification.

Keywords: Classifier, Data Mining, Genetic Algorithm, Supervised Learning, Porting.

1. Introduction

While organizations today store enormous amounts of data, the data is useless unless it can be properly interpreted and summarized. Data mining tools help turn data into information, and eventually into knowledge [1].

The Genetic Rule and Classifier Construction Environment (GRaCCE) is a data mining tool developed by Robert E. Marmelstein [2]. In addition to learning rules to accurately classify data, GRaCCE attempts to summarize its results in a form that is humanly understandable and simple. In contrast with other approaches that try to "tile" the data sets with pre-specified "shapes" [3], GRaCCE seeks natural boundaries between classes, which isolate points

[1] This work was supported by the AFRL/DAGSI Joint Research Program, Project #: SN-AFIT-99-04
[2] Contact Author and Potential Presenter

belonging to a class into regions that are amenable to "simple" description, without sacrificing classification accuracy. This leads to a small set of compact rules that are relatively easy to comprehend, compared with those found by other methods.

GRaCCE was prototyped in MATLAB [2]. While this implementation successfully demonstrated GRaCCE's ability to classify data in an understandable way, it also manifested some shortcomings that motivated its translation into C++. Since MATLAB is an interpreter, it produces code that is inherently slower than the corresponding compiled code. Translation of GRaCCE into C++ was originally attempted to speed up its execution, and to enable, at a later date, a fair comparison with other data mining algorithms.

Even though tools do exist to convert MATLAB code into C++ automatically, these tools are not mature enough to deal with the constructs used in the GRaCCE code. Furthermore, even if automatic translation were feasible, the resulting C++ code would not have been readily understandable to a programmer seeking to enhance it further.

Our approach to porting GRaCCE was guided by practical software engineering concerns of reuse, modularity, and verifiability, as explained in Section 3 [4]. Furthermore, it was necessary to build the system using only public domain software as much as possible, in order to be able to distribute it free of charge, via the web.

GRaCCE is still a work in progress. The C++ version that we have coded is a stepping stone towards further algorithmic improvements, generalization, and parallelization.

The rest of the paper is organized as follows. Section 2 provides a brief overview of the GRaCCE design. Section 3 describes our approach to implementing GRaCCE in C++. Section 4 sketches the performance results. Finally, Section 5 presents our conclusions.

2. GRaCEE Background

The GRaCCE algorithm consists of the following phases/steps:

(1) Preprocessing.
- (optional) Feature Selection
- (mandatory) Winnowing.

(2) Partition Generation.
(3) Data Approximation.
(4) Region Identification.
(5) Region Refinement.
(6) Partition Simplification.

The Preprocessing phase prepares the data set for subsequent rule induction. The optional feature selection step involves choosing the most relevant subset of features for classification, to deal with "the curse of dimensionality." GRaCCE provides three options for feature selection: (1) deterministic forward search, (2) genetic algorithm (GA) based search, and (3) hybrid approach in which the result of a limited deterministic forward search is used to initialize a population for the GA search.

The mandatory preprocessing step is the winnowing of the data set. The data points that fall within overlapping class boundaries interfere with the training of the classifier, potentially causing misclassifications. GRaCCE removes such data points, resulting in linearly separable class homogenous (CH) clusters of data. Winnowing uses the edited k-Nearest Neighbor (kNN) algorithm in which repeated passes over the data set remove instances whose class differs from that of the majority of the k-nearest neighbors, until either no remaining instances are misclassified, or the misclassification rate does not change in consecutive passes.

The remaining phases of GRaCCE construct the classification rules. The Partition Generation phase is responsible for generating a set of hyperplanes that are *sufficient* to separate each CH cluster. The equations for these hyperplanes are based on the Bayes decision boundary, and thus closely represent the natural boundaries between different classes [5]. There are two different kinds of partitions (hyperplanes): *Global partitions* are based on the global distribution of the data, and each pair of classes yields one global partition. *Local partitions*, on the other hand, are based on local distributions of data, and each boundary point pair of CH clusters of different classes yields a single local partition. A global partition provides a clean and natural boundary for two separable classes compared to a piecewise combination of local partitions (as abstracted in Figure 1), but it is not sufficient when the data points are distributed in an interleaved or checkerboard fashion (as abstracted in Figure 2). The local partitions provide sufficient means for separating each pair of CH regions.

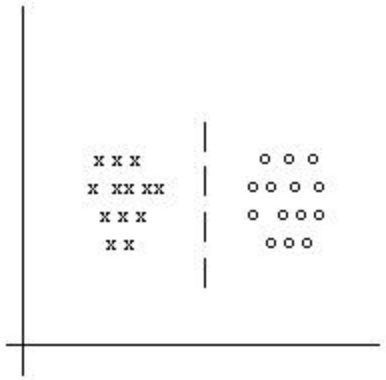

Figure 1

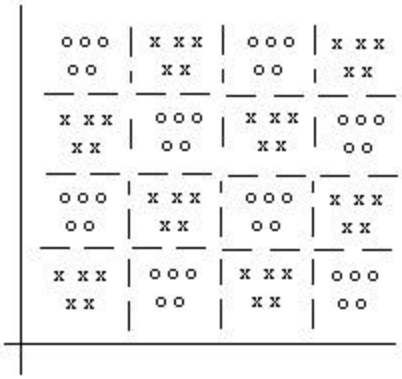

Figure 2

Since the boundary points of a winnowed data set outline the borders of CH regions, regions created to enclose the boundary points are a good approximation for enclosing all of the data points in the CH regions. Thus, a Data Approximation phase is used by GRaCCE in which the data set is approximated by the boundary points of each CH cluster, with weights assigned to each boundary point based on the number of points closest to it. This has the advantage of accelerating later phases.

The Region Identification phase is the heart of the GRaCCE algorithm. During this phase, genetic algorithms (GAs) are used to find combinations of partitions to enclose each CH region. Each region is represented as a binary chromosome, where a "1" signifies that the designated partition is to be utilized and a "0" signifies that it is not. A population of these binary chromosomes is maintained and evolved during the GA search. The objective function used to evaluate the fitness of each member of the population takes into account the accuracy, coverage, and number of partitions used.

Once a set of regions is found during the Region Identification phase to isolate each CH cluster of data, a Region Refinement phase is used to simplify the set of regions. This includes three substeps. First, any partition whose removal does not increase the misclassification rate is removed from the region. Next, any region that has a combination of small coverage and large number of partitions (defined by a "Region Utility Ratio") is removed from the system. Finally, the partitions are reoriented to improve classification accuracy.

The final step of the GRaCCE algorithm is the Partition Simplification phase. This phase reduces the complexity of each partition in a region by deleting terms that do not significantly affect the overall misclassification rate of the rule set.

The GRaCCE approach has a number of advantages over the other approaches. Firstly, GRaCCE gleans natural class boundaries in the data, the search for which is facilitated by the removal of noisy data in the winnowing phase. The use of both local and global partitions ensures that a sufficient set of partitions to enclose CH regions is available.

A second advantage of GRaCCE is its ability to find globally optimum solutions in a wide variety of search spaces. This is mainly due to its use of genetic algorithms to examine the search space that avoids convergence to local optima [6]. This is in contrast with other data mining methods, such as variants of decision trees, which make decisions based on local results [3].

A third advantage of GRaCCE is that it creates a set of decision rules that are more compact and simpler than those obtained through other methods. This is because the hyperplanes used are based on Bayes' natural class boundaries, and the objective function used in the Region Identification phase explicitly minimizes the number of partitions required to enclose regions. Furthermore, the Region Refinement and Partition Simplification phases help to simplify the regions and the partitions that make up the regions.

GRaCCE is amenable to parallel processing. This can improve its efficiency and scalability. Genetic algorithms are known to be fairly easily parallelized [7]. In fact, a parallel version of the Region Identification phase has already been implemented in MATLAB.

A final key benefit of GRaCCE is that the classification accuracy is robust with respect to the initial values of the user-specified parameters. This

contrasts with the behavior of many other clustering algorithms. This aspect unburdens the user from knowing in advance the properties of the data for adjusting these parameters.

3. Porting GRaCCE to C++

The MATLAB version of GRaCCE was reimplemented in C++ for the following reasons: (1) To improve the efficiency by translating from an interpreted language to a compiled language. (2) To enable other researchers ready access to GRaCCE in C++ for experimentation, comparison, and potential incorporation as a component in their system. (In fact, this entire effort has been initiated and funded by the U. S. Air Force.) (3) To develop a version of GRaCCE using public domain software as much as possible so that it can be distributed and used free of charge.

Automatic translation of GRaCCE from MATLAB to C++ was deemed doomed because: (1) The available translators are not general and mature enough to deal with the existing GRaCCE code. (2) Even if the translation were possible, the resulting C++ code would not be easy to read, understand, modify, and evolve, especially given that information implicit in the names used in the MATLAB source code would be lost.

So, manual translation of GRaCCE into C++ was undertaken with special attention paid to the software engineering concerns, and the context of distribution and use. The following requirements were imposed on this exercise:

The C++ code should maintain a close correspondence with the original MATLAB code. This enabled existing users of the MATLAB version a smooth transition to the C++ implementation. It facilitated "ease of understanding" and "reuse of experience" for the existing GRaCCE users, and "reuse of architecture, specification and design" for the GRaCCE implementers. Instead of trying to "reinvent the wheel" and build the algorithms from scratch, the implementers could better focus on developing the C++ version as quickly as possible, and improve only the critical sections ("hotspots") of the code, when necessary. In other words, a tangible benefit of our approach was that the MATLAB code served as an abstract specification that could be *relatively* easily refined into C++ code in a modular fashion, so that its correctness can be verified by inspection.

In order to accomplish this, the interface that MATLAB provided to GRaCCE was isolated first. In particular, MATLAB primitives that were used in the original GRaCCE code were identified, and abstracted. This layer was then reimplemented by reusing code from the Template Numerical Toolkit (TNT) that provides classes for manipulating numerical matrices. (TNT, a successor to LAPACK, is a freely distributable package of matrix classes and operations developed at the National Institute of Standards and Technology [URL: http://math.nist.gov/tnt]. Packages such as ISML were not considered, as they are not freely distributable.) While TNT does not implement all of the matrix operations needed for GRaCCE, it does provide a solid foundation to build on in terms of the basic data structures and algorithms (such as LU factorization and equation solving) it supports. The additional matrix operations were built on top of TNT as a separate module.

Overall, the GRaCCE code has been organized into three main modules: Feature Selection, Winnowing, and Rule Induction. The Rule Induction module includes Partition Generation, Data Approximation, Region Identification, Region Refinement, and Partition Generation.

Currently, GRaCCE expects the data in a plain text file, each line corresponding to a data instance. The format of each line is: an integer class ID followed by one or more numerical (integer or real) values, one per attribute. For example, the first few lines of the Iris data set (see next section) is:

```
3     5.800   2.700   5.100   1.900
2     6.300   2.300   4.400   1.300
3     6.900   3.100   5.400   2.100
2     5.600   3.000   4.100   1.300
```

Figure 3

GRaCCE generates a report that lists the error rates on the training data and the test data due to the rule set after different phases of the algorithm. It then lists the partitions used to define each region, followed by the regions themselves. The partitions are given in the form shown in Figure 4, where the V-elements make up the vector normal to the hyperplane, and the X-elements make up the anchor point through which the hyperplane passes. Each class homogeneous region is presented as shown in Figure 5, where each utilized partition is listed along with the region's orientation (1 or –1) with respect to the partition, as

well as the center of the region. In this example, just two hyperplanes are sufficient to isolate the entire "open" region.

The original implementation of GRaCCE consisted of approximately 6700 lines of MATLAB code, plus about 850 lines of MATLAB code in a third-party supplied GA toolbox (including only the functions actually used).

```
  Partition 1              ...
     V(1  )                    ==> 0.00
     V(2  )                    ==> 0.00
     V(3  )                    ==> 0.00
     V(4  )                    ==> -0.23
     X(1  )                    ==> 5.10
     X(2  )                    ==> 2.94
     X(3  )                    ==> 2.27
     X(4  )                    ==> 0.76
```

Figure 4

```
Cluster Number              ==> 2
   Primary Class               ==> 2
   Number of Members           ==> 43
   Number of Partitions        ==> 2
   Partition Map                  ...
      Partition 1              ==> -1
      Partition 2              ==> -1
   Cluster Center              ...
      X(1  )                   ==> 5.92
      X(2  )                   ==> 2.76
      X(3  )                   ==> 4.25
      X(4  )                   ==> 1.32
```

Figure 5

Our version of GRaCCE consists of approximately 7600 lines of C++ code plus 850 lines of C++ code for the GA routines. Additionally, the code for the TNT files actually used includes about 1900 lines of C++ code, our additional matrix operations consist of about 2300 lines of C++ code, and an optional code for a GUI designed to work under a Windows environment consists of approximately 3000 lines of C++ code.

4. Performance Results

GRaCCE's effectiveness was measured by running it on six data sets obtained from the UCI-Irvine ML Repository (URL:http://mlwww.diee.unica.it/ML/MLSummary.html) and one from [8]. Tables 1 and 2 summarize the results of these test runs. GRaCCE was executed on each data set five times, and the averages of these times are given in these tables. All code was compiled under Visual C++ 6.0 on a 667 MHz Intel Pentium III Dell PC with 128 MB RAM running Windows 2000. While these results obviously depend on the hardware used, it does give us a reasonable estimate of the expected performance on the state-of-the-art PC.

The tests from which all of the results given in these tables were performed were done without using Feature Selection. Executing Feature Selection took much longer than the other phases combined (even by factors in the hundreds when using a GA-based method for Feature Selection), while executing the later phases on a reduced feature set still took approximately 80 percent of the time as on a full feature set on average. This is mostly due to the kNN algorithm used to evaluate the fitness of each member in the population.

Table 1 gives some information relevant to the complexity of the data sets as well as the raw times needed to complete the GRaCCE algorithm. GRaCCE took anywhere from one second on the simplest data set tested to six minutes on the most complex data set tested. Also included in this table for each data set is the ratio of the total time taken by the previous MATLAB code over the total time taken by the new C++ code. As can be seen, an overall speedup factor of anywhere from 9 to 37 was obtained on each data set. The Region Identification phase experienced the greatest speedup (by far), and thus data sets in which this phase takes the largest percentage of time tends to have the largest overall speedup. The amount of speedup for each phase seems to depend largely on the percentage of work that is done during the phase inside the MATLAB primitives, which are usually compiled and optimized (unlike the user-written code which is usually interpreted).

Table 2 gives the break-up of the time taken by each phase as compared with the entire task. The Partition Generation phase seems to be a bottleneck, accounting for approximately 45 percent of the total time on average. This is true despite the fact that the Region Identification has the highest growth rate of any of the phases [2]. Also note that the Region Identification phase is likely to vary widely in execution time due to the modality of the data set (more modal data requires more regions to be found). Finally, the time taken by Winnowing, Region Identification, Region Refinement, and Partition

Generation take approximately equal fractions of the total time on average, while Data Approximation takes by far the smallest fraction.

5. Conclusions

The implementation of GRaCCE in C++ has demonstrated several key points. Firstly, it has constructively demonstrated perceptible improvements in running time. This speedup can especially be considered significant depending on the time scales involved. While improving an algorithm that takes a few seconds to one that takes a fraction of a second is not worthy of attention in practice, speeding up an algorithm that took an hour to execute to just a couple of minutes is very significant. As real-world data sets can often be much larger than those listed, the time savings from the user's perspective can be quite dramatic. As stated previously, the C++ version of GRaCCE will provide a foundation for further experimentation and future enhancements, both on the algorithmic front and on the usability front. Furthermore, the analysis of the results gives us insights into the relative resource requirements of the various phases, to better focus future efforts. Since the Partition Generation and Feature Selection phases seem to be the major bottlenecks, in part being provably intractable, they seem like the best candidates for parallel implementation.

Table 1 Average Raw Times for Each Phase (in sec)

Data Set	Classes	Samples	Features	Time (sec)	Factor of Speedup over MATLAB[3]

[3] Times for larger data sets with the MATLAB version were not available due to memory limitations of the Student version of MATLAB that was used for testing.

4 Times for each phase were measured to the nearest tenth of a second. Thus, any phase reported as taking zero percent of the total time are due to the phase taking less than 0.1 seconds for all tests.

Iris	3	150	4	1.0	17.7
Wine	3	178	13	1.2	15.6
Glass	6	214	9	6.0	23.3
Ionosphere	2	351	34	16.4	36.9
Diabetes	2	768	8	17.8	8.6
Cancer	2	699	9	18.8	19.2
FLIR	2	1000	6	33.4	27.6
Thyroid	2	3163	24	343.8	--
Soybean	19	683	35	642.5	--

Table 2 Average Fractions of Total Time for All Phases[4]

	Winnow	Partition Gener.	Data Approx.	Region ID	Region Refine.	Part. Simp.	Rule Ind Total
Iris	0.10	0.46	0.00	0.24	0.20	0.00	0.90
Glass	0.19	0.43	0.00	0.09	0.17	0.14	0.81
Wine	0.04	0.39	0.01	0.28	0.09	0.20	0.96
Cancer	0.20	0.67	0.01	0.02	0.10	0.01	0.80
Ionosphere	0.06	0.87	0.01	0.04	0.05	0.09	0.94
Pima	0.19	0.35	0.02	0.21	0.12	0.12	0.81
FLIR	0.17	0.38	0.02	0.24	0.14	0.05	0.83
Thyroid	0.20	0.47	0.00	0.01	0.31	0.00	0.80
Soybean	0.01	0.12	0.00	0.03	0.02	0.82	0.99

References

[1] Tobin, Daniel R. 1997. *The Knowledge-Enabled Organization*. AMACOM Books.

[2] Marmelstein, Robert E. 1999. "Evolving Compact Decision Rule Sets." Ph.D. diss., Air Force Institute of Technology.

[3] Safavian, S. Rasoul and David Landgrebe. May 1991. "A Survey of Decision Tree Classifier Methodology." *IEEE Transactions on Systems, Man, and Cybernetics* 21(3): 660-674.

[4] Meyer, Bertrand. 1997. *Object-Oriented Software Construction*. 2nd Edition. Prentice Hall.

[5] Lee, Chulhee and David Landgrebe. 1993. "Feature Extraction and Classification Algorithms For High Dimensional Data." Ph.D. diss, Purdue University.

[6] Patnaik, Lalit M. and Srinivas Mandavilli. "Adaptation in Genetic Algorithms." 1996. In *Genetic Algorithms for Pattern Recognition*, ed. Sankar K. Pal and Paul P. Wang, 45-64. Boca Raton, FL: CRC Press.

[7] De, Susmita, Ashish Ghosh, and Sankar K. Pal. 1996. "Fitness Evaluation in Genetic Algorithms with Ancestors' Influence." In *Genetic Algorithms for Pattern Recognition*, ed. Sankar K. Pal and Paul P. Wang, 1-23. Boca Raton, FL: CRC Press.

[8] Ernisse, Brian E. 1996. "Automatic Target Cuer/Recognizer System for Tactical FLIR Images." M.S. Thesis, Air Force Institute of Technology.