A genetic prototype learner

Sandip Sen
Dept of Mathematical & Computer Sciences,
The University of Tulsa, U. S. A.
sandip@kolkata.mcs.utulsa.edu

Leslie Knight
Dept of Math & Computer Sciences,
The University of Tulsa
U. S. A.

Abstract

Supervised classification problems have received considerable attention from the machine learning community. We propose a novel genetic algorithm based prototype learning system, PLEASE, for this class of problems. Given a set of prototypes for each of the possible classes, the class of an input instance is determined by the prototype nearest to this instance. We assume ordinal attributes and prototypes are represented as sets of feature-value pairs. A genetic algorithm is used to evolve the number of prototypes per class and their positions on the input space as determined by corresponding feature-value pairs. Comparisons with C4.5 on a set of artificial problems of controlled complexity demonstrate the effectiveness of the proposed system.

1 Introduction

The induction of concept classification methods has received widespread attention both in the cognitive sciences and in the machine learning communities. This is to be expected because the development and use of concepts is a key characteristic of intelligent behavior. A study of the acquisition of conceptual knowledge involves an analysis of the representation of this knowledge, as well as the process of drawing inductive inferences using these representations.

A number of symbolic and subsymbolic systems have been proposed to solve the problem of inductively acquiring concept classification knowledge from a set of pre-classified instances. Some of the more influential symbolic systems include ID3 [Quinlan, 1986], version spaces [Mitchell, 1986], and COBWEB [Fisher, 1987]. The most widely used subsymbolic systems for concept learning are based either on neural networks [Rumelhart *et ai,* 1986] or on genetic algorithms (GAs) [DeJong, 1990; Holland, 1986]. Some of the more frequently used representations in symbolic concept learning systems use logic expressions and programs, decision trees and lists, hierarchical clusters, and rules. Representations used in subsymbolic systems include networks of simple computational units connected by weighted links and bit strings.

In addition to these mainstream machine learning approaches, there exists a significant body of work on statistical induction schemes [Breiman *et ai,* 1984; Cheeseman *et ai,* 1988]. Other researchers have been motivated by the findings of cognitive psychologists [Schaffer, 1978; Smith and Medin, 1981] and are using exemplar-based classification schemes [Aha *et ai,* 1991; Kruschke, 1992; Stanfill and Waltz, 1986]. These methods store the set of training instances and classify a new instance by some voting scheme using a given number of nearest stored examples. The exemplar-based approach (also called the nearest neighbor classification algorithm [Vosniadou and Ortony, 1989]) is attractive because of its simple representation, and because it can be used to explain a number of human cognitive phenomena [Schaffer, 1978; Smith and Medin, 1981]. Hybrid GA-nearest neighbor algorithms have been developed to learn weights associated with individual attributes [Kelly and Davis, 1991] (used in calculating distance of new instance from the stored instances) and to store only a subset of the set of training instances [Cherkauer, 1992].

The other form of concept representation that has been commonly posited by cognitive scientists as a theory of human concept learning is that of *prototypes* [Reed, 1972; Smith, 1989]. A prototype is a collection of salient features of a concept. An instance can be classified using prototypes by finding the prototype with which it shares most of its features, and then using the class of that prototype. The exemplar based models of concept classification are more flexible than prototype models, but cannot account for some general aspects of ordinary concepts. Smith [Smith, 1989] concludes that for ordinary concepts like *birds,* we use prototypes rather than storing exemplars; for complex concepts like *students in my class,* however, exemplars can be used to construct prototypes or to directly classify instances. In addition, the prototype models produce much more compact concept descriptions compared to exemplar-based models, and hence require much less computational effort to classify new instances. The on-line computation of prototypes, however, has been recognized as a difficult computational problem [Hintzman, 1986; Kahneman and Miller, 1986].

In this paper, we use a genetic algorithm [Holland, 1975] to evolve prototypes from pre-classified instances. We believe that in addition to possessing psychological

plausibility, prototype models can be used to build effective classification techniques for supervised classification problems. To test this hypotheses, we have defined a set of classification problems involving two classes and two real-valued features. The constructed problems are of varying complexity as measured by the number of prototypes per category required for correct classification. We compare our system with the C4.5 system [Quinlan, 1993] on these data sets.

The rest of the paper is organized as follows: Section 2 discusses alternate representations for prototype based classification and introduces our representation; Section 3 describes the GA used to evolve these prototypes; Section 4 describes the classification problem set used to evaluate our system; Section 5 compares experimental results of our system with the C4.5 system, and Section 6 presents the shortcomings of the current system and future research directions to address these problems.

2 Representation

The standard approach to representing prototypes consists of using the descriptions of individual instances of a given class and then abstracting the more frequent properties of these instances [Kahneman and Miller, 1986]. A prototype is often represented in a slot-filler structure containing default attribute values, relationships between attributes, and weights on attributes. In these representations, prototypes are often linked in hierarchies. Other possible representations for prototypes include production systems [Holland *et al.*, 1986] and connectionist networks.

In general, only one prototype is constructed per class[1]. The similarity of an input instance to a prototype is calculated from their respective attribute values using a "contrast rule": attributes not having common values are weighted and subtracted from the weighted sum of the attributes having common values. This method of classification explains typicality effects seen in most natural concepts (e.g., some birds are more easily recognized as bird than others).

Using a single prototype per class, however, is not sufficient to learn linearly non-separable categories. Since most practical classification problems are likely to be linearly non-separable, we allow multiple prototypes per class. The classification problems used in this paper are defined on real-valued attributes. Hence, the calculation of the similarity metric and the representation of prototypes are modified to suit these domains. Our representation of prototypes is identical to that of exemplars, that is a prototype is represented as a list of feature values together with its associated class. The similarity metric used is the Euclidean distance between the input instance and the prototype in the space defined by the features.

[1] Actually, it is widely believed that an ordinary concept definition consist of a prototype constructed from perceptually salient and easy to compute features and a *core* made up of more accurate but less accessible features [Smith *et al*, 1984].

Let $P_{ij}, i = \{1, \ldots, n\}$, $j = \{1, \ldots, m\}$ be the value of the jth attribute of the ith prototype, and I_{kj} be the value of the jth attribute of the kth instance. Then the nearest prototype to the kth instance is calculated as

$$\arg\min_i \sum_{j=1}^{m} (P_{ij} - I_{kj})^2.$$

Note that our prototype model is similar to the exemplar model in the nearest neighbor calculation. The difference is that the prototypes are, in general, distinct from any examples seen by the system, and the number of prototypes per class is restricted to a small number (5 in this paper). Our model is similar to some prototype models which calculate similarity from a distance measure in some underlying psychological space rather than measuring it by common and distinctive properties [Shepard, 1974].

3 Genetic algorithms for learning prototypes

Genetic-Based Machine Learning (GBML) systems are rule-based systems which can be used to determine the class membership of input instances from a set of attributes. A GBML system matches the set of attributes corresponding to an instance against a set of rules to determine the class membership of the instance. These domain-independent classification mechanisms are particularly useful in problem domains for which there is no known precise model to determine the class, or for which determining a precise model is impractical.

There are two principal schools of thought in designing GBML systems [DeJong, 1990]. The first school of researchers proposes to use genetic algorithms to evolve individual rules, a collection of which comprises the classification expertise of the system. This approach to building classifier systems was originally proposed by John Holland at the University of Michigan, and hence is referred to as the Michigan approach [Holland, 1986]. The other school of thought has been popularized by Ken De Jong and Steve Smith [Smith, 1980] from the University of Pittsburgh, and is therefore referred to as the Pitt approach to building classifier systems. In this approach, genetic algorithms are used to evolve structures, each of which represent a complete set of rules for classification. Each structure in the population in the Pitt approach corresponds to the entire set of rules in the Michigan approach.

The Pitt approach seems to be better suited for batch-mode learning (where all training instances are available before learning is initiated) and for static domains. The Michigan approach is more flexible to handle incremental-mode learning (training instances arrive over time) and dynamically changing domains. We have chosen the Pitt approach in this paper to take advantage of the availability of all training data before learning is initiated. The only difference with the Pitt approach is that a structure consists of a set of prototypes instead *of* a set of rules.

3.1 Population structures

Each population structure consists of one or more prototypes belonging to each of the possible classes in the domain. Each prototype is represented as a set of feature values as described in Section 2. Prototypes belonging to the same class are placed adjacent to each other. Prototypes belonging to different classes are separated by a special marker, ||. Let us denote the jth feature value of the ith prototype belonging to the *kth* population structure by P_{ij}. Then given a problem with two classes and two features, the *kth* structure in the population with two prototypes belonging to the first class and three prototypes belonging to the second class is represented as:

$$P_{11}^k P_{12}^k \;\; P_{21}^k P_{22}^k \;||\; P_{31}^k P_{32}^k \;\; P_{41}^k P_{42}^k \;\; P_{51}^k P_{52}^k$$

Traditional GAs use bit-string representations for population structures [Goldberg, 1989]. Our choice of real valued representation has recently received increasing attention in the GA community [Goldberg, 1991] and has been effectively used for supervised concept classification problems [Kelly and Davis, 1991; Corcoran and Sen, 1994].

3.2 Operators for structure manipulation

Genetic algorithms are a class of adaptive techniques that were motivated by the effectiveness of natural evolution in developing organisms well-suited to a variety of environmental conditions. A majority of the structure manipulation operators in GAs are highly abstract versions of operators found in natural genetics.

GAs operate on a population of structures, evolving more well-adapted structures to the given environment over successive generations. Each structure is evaluated in the domain to give it a fitness measure. We formulate the prototype learning problem as a function minimization problem, where we are searching for structures that reduce the number of misclassifications on the training set. The number of misclassifications produced by a structure is used as a measure of its fitness. We formulate the prototype learning problem as a function minimization problem, where we are searching for structures that reduce the number of misclassifications on the training set.

After all the structures in the population are evaluated, a new generation is formed by inserting new structures into the population according to their fitness values, causing the poorer performing structures to be eliminated (some get copied over multiple times depending on their relative fitness). For this *selection* process we use a rank-based method which has been claimed to be superior to the traditional fitness proportionate selection scheme [Whitley, 1989].

Mutation is used to replace the current value of one feature of a prototype by a randomly generated number in the domain of the feature. Thus mutation allows for a drastic change in the position of a prototype along one of the dimensions in the input space (each dimension corresponds to one attribute of the problem). Mutation takes place infrequently. The probability of mutation is set by a parameter p_{mut}.

A *creep* operator is used to displace the current position of a prototype on the input space by a small amount. The creep operator is applied with a probability p_{creep} and changes the values of all the features of a prototype by a small percentage, *8,* of their current values[2].

A two point crossover operator is used to swap subparts of two structures. The crossover points can fall anywhere within a structure. When two parents are selected for crossover, two crossover points are first randomly selected on one of the parents . Now, two points are chosen on the other parent which match up semantically with the previously chosen points on the first parent. For example, if a crossover point on the first parent is in between prototypes belonging to category *X,* the corresponding crossover point in the the second parent should also fall between prototypes for category *X.* Similarly, if the crossover point in the first parent falls between the second and third features of a prototype belonging to category *Y,* the corresponding crossover point in the the second parent should also fall between the second and third features of a prototype belonging to category V. After crossover points are selected in both parents, the portions of the chromosomes in between these points are swapped between the parents to produce two offsprings. Any crossover that results in chromosomes with empty categories or with a number of prototypes for a category exceeding the maximum limit, is disallowed. The following example demonstrates a valid two-point crossover as described above (crossover points are marked by the symbol]):

$$\begin{aligned}
\text{Parent 1:} \quad & P_{11}^k \;]\; P_{12}^k \;\; P_{21}^k P_{22}^k \;||\; P_{31}^k P_{32}^k \;\; P_{41}^k P_{42}^k \;|\; P_{51}^k P_{52}^k \\
\text{Parent 2:} \quad & P_{11}^l \;]\; P_{12}^l \;||\; P_{21}^l P_{22}^l \;]\; P_{31}^l P_{32}^l \\
\text{Offspring 1:} \quad & P_{11}^k P_{12}^l \;||\; P_{21}^l P_{22}^l \;\; P_{51}^k P_{52}^k \\
\text{Offspring 2:} \quad & P_{11}^l P_{12}^k \;\; P_{21}^k P_{22}^k \;||\; P_{31}^k P_{32}^k \;\; P_{41}^k P_{42}^k \;\; P_{31}^l P_{32}^l .
\end{aligned}$$

4 Problem set

In order to evaluate our proposed system, PLEASE, we designed a set of problems of varying complexity. These problems were defined on two continuous attributes, *x* and y, each having a range of [0,1]. So, the input space is a square of unit area. We formulated four different problems by labeling different regions of this unit square with one of two possible categories, 1 or 0. The area of the region allocated to each of the categories is equal in area for all the problems. These regions were chosen so that the classification problem can be solved with a specific number of appropriately positioned prototypes for each category. Different problems require different number of prototypes per category. The minimum number of prototypes required per category to solve the classification problem is used as a measure of the complexity of the problem. Given a particular problem, we randomly generated a set of points from the input space and then labeled each point with the category associated with the region containing that point. These sets of points were then divided into training and testing sets.

[2] In our experiments, we found faster convergence when all the feature values were changed by the creep operator rather than changing only one feature at a time.

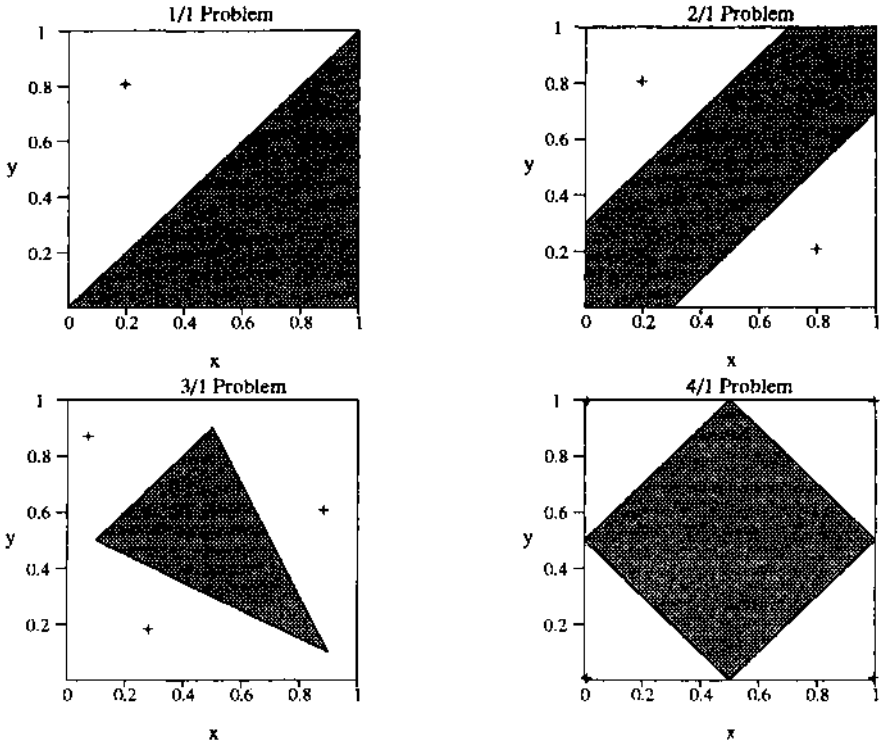

Figure 1: The classification problem set.

We present the problems used in the experiments in this paper in Figure 1. The problems are labeled as N/M, where N and M stand for the minimum number of prototypes in the two categories required to accurately solve the classification problem. The darkened regions in each of the figures are associated with category 0, and the rest of the square is associated with category 1. The prototypes required to solve the problem are labeled '-' and '+' for categories 0 and 1 respectively. We should clarify that the presented problems are not unique instances of N/M problems; their choice was motivated by considerations of the ease of exposition. The following describes each of the problems in more detail:

1/1 problem: The unit square is divided by a diagonal into two right-angled triangles, and each assigned to one category. This is the simplest problem in the problem set, and can be solved by using only two prototypes, one each for the two categories. The two prototypes must be placed on a perpendicular bisector of the dividing diagonal and should be at the same distance from the diagonal. The prototypes should also be correctly labeled, i.e., the prototype in the region assigned to category 1 should be labeled by a '+' and the prototype in the region assigned to category 1 should be labeled by a '-' (we will assume this is the case when describing the other problems). Such an orientation of the prototypes means that any point in the upper triangle will be closer to the prototype in that triangle and hence classified correctly. The interesting thing to note here is that the above solution description allows for an infinite number of minimal solutions[3] to the problem (infinite number of perpendicular bisectors of the diagonal, infinite number of pairs of points on any of these bisectors that are equidistant from the diagonal). One such solution is showed in the figure[4].

2/1 problem: In the particular 2/1 problem chosen for experimentation, a single diagonal strip associated with category 0 separates two triangular regions assigned to category 1. Once again, prototypes placed on any straight line perpendicular to the two diagonal lines can solve the classification problem perfectly. As a result, there are infinitely many minimal solutions to this classification problem with the prototype representation we have chosen. One such placement of prototypes is shown in the figure.

3/1 problem: The presented problem consists of a triangular region associated with category 0, and the rest of the unit square associated with category 1.

[3] Minimal solutions refer to solutions that require the minimal number of prototypes for each category.

[4] The placement of the prototypes in the figures are only approximately correct.

In contrast to the above problems, there is a unique minimal solution to this problem. This solution is found by first drawing perpendicular bisectors of the three sides of the triangle and placing the '-' prototype at the intersection of these three bisectors. The three ' + ' prototypes are then placed at points outside the triangles obtained by reflecting the position of the '-' prototype about each of the three sides of the triangle.

4/1 problem: The presented problem consists of a square embedded within the unit square and with vertices placed on the midpoints of the sides of the unit square. The embedded square is associated with category 0 and the rest of the unit square is associated with category 1. Once again, there exists a unique minimal solution to this problem. This solution is obtained by placing the "-' prototype at the center of the square, and the four ' + ' prototypes at the four vertices of the unit square.

The problem set, therefore, consists of problems of varying complexity, both in terms of the minimal number of prototypes required per category to solve the classification problems, as well as in terms of the number of minimal solutions existing for the problems.

5 Experimental results

In this section, we present results from experiments conducted to solve the previously mentioned classification problems using both PLEASE and CM.5. We have used the *cross validation* error estimation technique [Breiman *el al.*, 1984] for evaluating the performance of the algorithms on the problem set. For each of the problems, 1000 points were randomly generated from the input space and classified according to the region from which it was drawn. We used a five-fold cross validation, generating five different training/testing set splits. In each such split, 800 input instances were used for training, and the remaining 200 were used for testing. The constraint imposed on training/test split was that both the categories were equally represented in the training and test sets. Additionally, within a category, all the regions were also equally represented. Results obtained with the C4.5 system are averaged over these five training/testing set splits for each of the problems. Our PLEASE system uses genetic algorithms for learning prototypes, and the performance of the latter depends on the random initial population. As such, the PLEASE system was run on each training/testing split with 10 different random initializations. The results of PLEASE on each problem, therefore, is averaged over 50 runs.

The parameters used for the PLEASE system in these experiments are as follows: population size = 150, number of generations = 200, p_{mut} = 0.0001, p_{creep} - 0.3, s = 0.1, selection bias = 1.7, maximum number of prototypes per category = 5. Table 1 contains the average and standard deviations of the percentage error rates of C4.5 and PLEASE on each of the four problems in the problem set.

The main observations from these results are as follows:

- PLEASE was consistently able to discover a set of prototypes that achieved near-perfect classification accuracy.
- The error rates for PLEASE and C4.5 increased slightly with more complex problems.
- Though C4.5 performed slightly better on training instances in the some of the problems, these differences were not statistically significant. On the other hand, a two-sample t procedure shows that with at least 99% confidence level it can be stated that PLEASE produced lower error rates on the test instances than C4.5. The other notable difference between C4.5 and PLEASE was their relative performance on the training and test cases. Training and test set errors were much more consistent in PLEASE than in C4.5. In fact, for the C4.5 system, test errors were approximately 5 to 8 times higher than errors on training instances. Since test set performance is more important in supervised classification problems, PLEASE seems to be able to better represent the underlying target concept.

We now analyze the kind of solutions PLEASE generates for the given problem set. Table 2 presents the average number of prototypes used by the best solutions generated by PLEASE for each of the four problems. The numbers show that the solutions produced use more prototypes than required by the minimal solutions to corresponding problems. On analyzing individual solutions, we found that some of the solutions indeed were minimal, but in general, more prototypes were used. The decision trees produced by C4.5 for these problems contained, on the average, 44.2, 57.4, 69, and 70.2 nodes for the 1/1, 2/1, 3/1, and 4/1 problems respectively. Hence, the data compression obtained with the PLEASE system is significantly better compared to that obtained with C4.5.

We will use a typical solution for the 4/1 problem to analyze both the nature of the solutions generated by PLEASE, and the classification errors made by such a solution. Figure 2 presents the 1000 data points for the 4/1 problem and also a typical solution produced by PLEASE together with the training and test set misclassifications of that solution. For the sake of clarity we have also drawn the lines outlining the embedded square for this problem. We find that the solution produced by PLEASE contains several prototypes for category 0, all clustered around the center of the unit square. The minimal solution contains just one prototype at the center. Similarly, there are two prototypes belonging to category 1 placed close to each other at the left corner of the unit square. These two observations suggest the use of a prototype merging operator that will replace two prototypes by a new one in between them, if the distance between the two prototypes is less than a preset threshold. The placement of the prototypes in the solution generated by PLEASE is found to approximate, the prototype locations in the minimal solution. This close approximation to the unique minimal solution produces very low classification errors, 3 on the training set and 3 on the test set.

On analyzing the misclassifications produced we find

Problem type	Training set		Test set	
	PLEASE	C4.5	PLEASE	C4.5
1/1	0.11 (0.17)	0.525 (0.21)	0.32 (0.88)	3.7 (1.44)
2/1	0.81 (0.61)	0.75 (0.09)	1.4 (1.08)	4.3 (1.86)
3/1	0.78 (0.66)	0.8 (0.41)	1.4 (1.02)	4.4 (0.85)
4/1	0.96 (0.41)	0.7 (0.06)	1.7 (1.02)	5.5 (1.45)

Table 1: Average percentage of incorrect classifications by PLEASE and C4.5 (numbers in parentheses represent standard deviations).

Problem type	Average # of Prototypes	
	Category +	Category -
1/1	1.7	1.86
2/1	3.84	2.72
3/1	4.42	1.36
4/1	4.58	4.76

Table 2: Average number of prototypes learned by PLEASE for the two categories for each of the problem types.

no "blatant" errors, i.e., all errors are located close to the boundary of separation between the categories. The other solutions that we have analyzed show similar trends in both the prototypes developed and the errors produced.

6 Conclusions

The choice of prototypes as a representation of target classes and the use of genetic algorithms to learn appropriate prototypes for given classes has been shown to be an effective means for addressing supervised classification problems. In particular, we have provided a novel way of successfully addressing the difficult problem of on-line computation of prototypes from training data [Hintzman, 1986; Kahneman and Miller, 1986].

This paper contains an experimental evaluation of our proposed classification system, PLEASE, on a set of real-valued classification problems of varying complexity. PLEASE produces near-perfect solutions to these problems and performs better than the well-known C4.5 system on these problems. In a further study with an extended set of problems [Knight and Sen, 1995], we have found the PLEASE system performs favorably compared to simple exemplar-based classification mechanism like the nearest neighbor algorithm.

The system as it stands today can be improved in a number of ways. We envision the following experiments with and extensions to the current system:

- Developing a representation for both structured and nominal attributes in addition to the ordinal attributes that are used currently.
* Evaluating PLEASE for noisy immunity.
- Using prototypes with a subset of attributes (necessary to handle domains with large number of irrelevant attributes); this goes back to the more traditional view of prototypes [Kahneman and Miller, 1986].
- Collapsing two prototypes into a single prototype if they are closer to each other than a given threshold.
- Using the inversion operator in GAs, to allow swapping of prototypes between any two classes in a chromosome.
- A prototype add (delete) operator that randomly selects a class in the chromosome and then adds (deletes) a prototype to (from) that class. The prototype to be added may be constructed by averaging feature values of a subset of the instances in the training set of that class that have been misclassified by this chromosome. A prototype may also be deleted if it did not classify any instance.
* Seeding of the population by chromosomes constructed from possibly useful prototypes; these prototypes can be formed by averaging feature values over a certain number of instances of a class that are randomly selected from the training set.

We also plan to develop a prototype-based Michigan-style classifier system.

References

[Aha *et al.*, 1991] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning,* 6(1), 1991.

[Breiman *et al.,* 1984] L. Breiman, J.H. Friedman, and R.A. Olshen. *Classification and Regression Trees.* Wadsworth, Belmont, CA, 1984.

[Cheeseman *et al,* 1988] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: A bayesian classification system. In *Proceedings of the Fifth International Conference on Machine Learning,* pages 54-64, 1988.

[Cherkauer, 1992] K.J. Cherkauer. Genetic search for nearest-neighbor exemplars. In *Proceedings of the Fourth Midwest Artificial Intelligence and Cognitive Science Society Conference,* pages 87-91, 1992.

[Corcoran and Sen, 1994] A.L. Corcoran and S. Sen. Using real-valued genetic algorithms to evolve rule sets for classification. In *Proceedings of the IEEE Conference on Evolutionary Computation,* pages 120-124, 1994.

[DeJong, 1990] Kenneth A. DeJong. Genetic-algorithm-based learning. In Y. Kodratoff and R.S. Michalski, editors, *Machine Learning, Volume III.* Morgan Kaufmann, Los Alamos, CA, 1990.

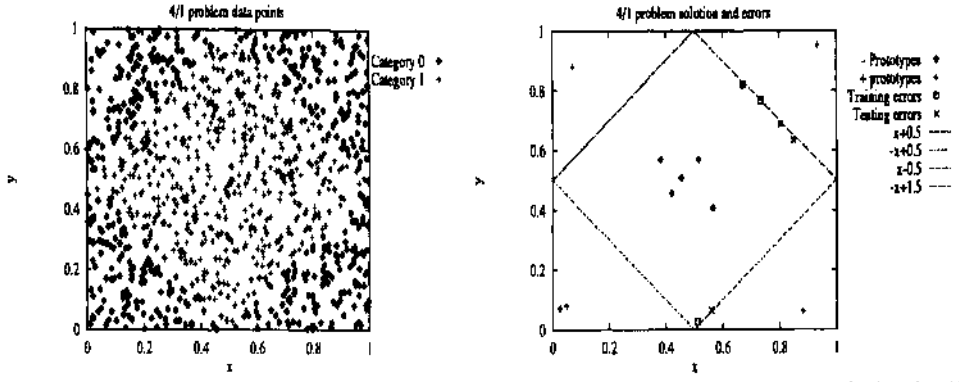

Figure 2: Data points for the 4/1 problem, and a typical solution generated by PLEASE together with the classification error of this solution on training and testing sets for this problem.

[Fisher, 1987] Douglas H. Fisher. Knowledge acquisition via incremental conceptual clustering. Machine Learning, 2:139-172, 1987.

[Goldberg, 1989] David E. Goldberg. Genetic Algorithms in Search, Optimization & Machine Learning. Addison-Wesley, Reading, MA, 1989.

[Goldberg, 1991] David Goldberg. Real-coded genetic algorithms, virtual alphabets, and blocking. Complex Systems, 5:139-168, 1991.

[Hintzman, 1986] D.L. Hintzman. "schema abstraction" in a multiple trace memory model. Psychological Review, 93:411-428, 1986.

[Holland et al, 1986] John H. Holland, K.J. Holyoak, R.E. Nisbett, and P.R. Thagard. Induction: Processes of Inferences, Learning, and Discovery. MIT Press, Cambridge, MA, 1986.

[Holland, 1975] John H. Holland. Adpatatton in natural and artificial systems. University of Michigan Press, Ann Arbor, MI, 1975.

[Holland, 1986] John H. Holland. Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R.S. Michalski, J.G. Carbonell, and T. M. Mitchell, editors, Machine Learning, an artificial intelligence approach: Volume II. Morgan Kaufmann, Los Alamos, CA, 1986.

[Kahneman and Miller, 1986] D. Kahneman and D.T. Miller. Norm theory: Comparing reality to its alternatives. Psychological Review, 93:136-153, 1986.

[Kelly and Davis, 1991] James D. Kelly and Lawrence Davis. A hybrid genetic algorithm for classification. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 645-650, 1991.

[Knight and Sen, 1995] Leslie Knight and Sandip Sen. PLEASE: A prototype learning system using genetic algorithms. In Proceedings of the 6th International Conference on Genetic Algorithms, San Mateo, CA, 1995. Morgan Kaufman.

[Kruschke, 1992] J.K. Kruschke. ALCOVE: An exemplar-based connectionist model of category learning. Psychological Review, 99:22-44, 1992.

[Mitchell, 1986] T. Mitchell. Generalization as search. Artificial Intelligence, 18:203-226, 1986.

[Quinlan, 1986] Ross J. Quinlan. Induction of decision trees. Machine Learning, 1:81-106, 1986.

[Quinlan, 1993] Ross J. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, California, 1993.

[Reed, 1972] S.K. Reed. Pattern recognition and categorization. Cognitive Psychology, 3:382-407, 1972.

[Rumelhart et al., 1986] D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, Parallel Distributed Processing, volume 1. MIT Press, Cambridge, MA, 1986.

[Schaffer, 1978] D. L. Medin & M.M. Schaffer. Context theory of classification learning. Psychological Review, 85:207-238, 1978.

[Shepard, 1974] R.N. Shepard. Representation of structure in similarity data: Problems and prospects. Psychometrika, 39:373-421, 1974.

[Smith and Medin, 1981] E.E. Smith and D.L. Medin. Categories and concepts. Harvard University Press, Cambridge, MA, 1981.

[Smith et al., 1984] E.E. Smith, D.L. Medin, and L.J. Rips. A psychological approach to concepts: Comments on rey's "concepts and stereotypes". Cognition, 17:265-274, 1984.

[Smith, 1980] Steve F. Smith. A learning system based on genetic adaptive algorithms. PhD thesis, University of Pittsburgh, 1980. (Dissertation Abstracts International, 41, 4582B; University Microfilms No. 81-12638).

[Smith, 1989] Edward E. Smith. Concepts and induction. In Michael I. Posner, editor, Foundations of Cognitive Science. MIT Press, Cambridge, MA, 1989.

[Stanfill and Waltz, 1986] C. Stanfill and D. Waltz. Toward memory-based reasoning. Communications of the ACM, 29:1213-1228, 1986.

[Vosniadou and Ortony, 1989] S. Vosniadou and A. Ortony. Similarity and Analogical Reasoning. Cambridge University Press, Cambridge, MA, 1989.

[Whitley, 1989] D. Whitley. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In Proceeding of the 3rd International Conference on Genetic Algorithms, pages 116-121, San Mateo, CA, 1989. Morgan Kaufman.