Prototype Selection & Pattern Recognition using Genetic Algorithms

by
R.Pradeep

Under the guidance of

Dr.V.Susheela Devi
Scientific Officer
Computer Science and Automation
Indian Institute of Science
Bangalore

July 7, 2005

Abstract

Machine learning is an area of artificial intelligence concerned with the development of techniques which allow computers to learn. More specifically, machine learning is a method for creating computer programs by the analysis of data sets. Machine learning overlaps heavily with statistics, since both fields study the analysis of data, but unlike statistics, machine learning is concerned with the algorithmic complexity of computational implementations. Many inference problems turn out to be NP-hard so part of machine learning research is the development of tractable approximate inference algorithms.

Machine learning has a wide spectrum of applications including search engines, medical diagnosis, stock market prediction, robot locomotion, game playing, handwriting & speech recognition and classifying DNA sequences.

Genetics Based Machine Learning(GBML), is a technique that incorporates evolutionary computing methods for Machine Learning. *Classifier Systems* are the most common GBML architecture. In this project, we have developed a simple classifier system and tested it on various input data.

Contents

Chapter 1

Introduction to Classifier Systems

1.1 Classifier Systems

A classifier system is a machine learning system that learns syntactically simple string rules (called *classifiers*) to guide its performance in an arbitrary environment. A classifier system consists of three main components:

1. Rule and message system
2. Apportionment of credit algorithm
3. Genetic Algorithm

The rule and message system of a classifier system is a special kind of *production system*. A production system is a computational scheme that uses rules as its only algorithmic device. Although there is a wide variation in syntax among production systems, the rules are generally of the following form:

$$\textbf{\textit{if}}\ \triangleleft\textbf{\textit{condition}}\triangleright\ \textbf{\textit{then}}\ \triangleleft\textbf{\textit{action}}\triangleright$$

The meaning of the production rule is that the action may be taken (the rule is "fired") when the condition is satisfied.

At first glance, the restriction to such a simple device for the representation of knowledge may seem too constraining. Yet it has been shown that

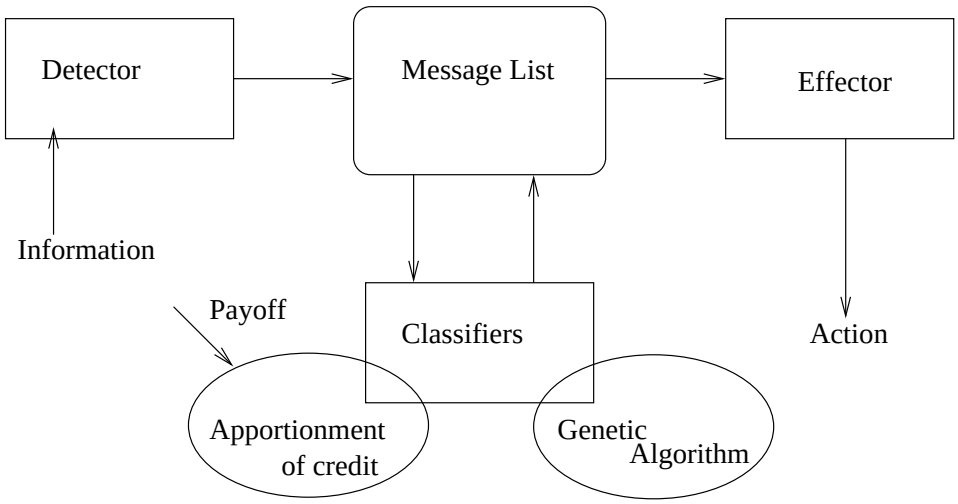

Figure 1.1: Block Diagram of a Learning Classifier System

production systems are computationally complete. Their power in representing knowledge involves more than this. They are also computationally convenient. A single rule or small set of rules can represent a complex set of thoughts compactly. The explosion of rule-based expert system applications over the past decade is strong empirical testimony to this claim.

1.2 The Rule and Message System

A schematic depicting the rule and message system, the apportionment of credit system, and the genetic algorithm is shown in 1.1. The rule and message system forms the computational backbone of the classifier system. Information from the environment is obtained through *detectors* and decoded into one or more finite length messages. These messages are then posted into a finite length message list. These messages may then invoke other classifiers or they may cause the system to take action using one of its *effectors*.

A message within a classifier system is simply a finite length string over some finite alphabet. If we limit ourselves to a binary alphabet, we obtain the following definition:

`<message> ::=` $\{0,1\}^l$

Messages are matched by classifiers in the rule-set. The syntax of a classifier is:

<classifier> ::= <condition><message>

The condition is a simple pattern recognizing device where a wild card(#) is added to the underlying alphabet:

<condition> ::= $\{0, 1, \#\}^l$

Once a classifier's condition is matched, that classifier becomes a candidate to post its message to the message list on the next time step. Whether the candidate classifier posts its message is determined by the outcome of an activation auction, which in turn depends on the evaluation of a classifier's value. This is taken care of by the apportionment of credit.

1.2.1 Advantages of the Representation

As we see from the previous section, classifier systems use a fixed length representation. This restriction has two benefits. First, all strings under the permissible alphabet are syntactically meaningful. Second, a fixed string representation permits the string operators of the genetic kind. This leaves the door open for a genetic search of the space of permissible rules.

1.2.2 Rule activation

Classifier systems use a parallel rule activation while traditional systems use serial rule activation. During each matching cycle, a traditional expert system activates a single rule. This rule-by-rule procedure is a bottleneck to increased productivity, and much of the difference between competing expert system architectures concerns the selection of the better single rule activation strategies for a particular problem. Classifier systems overcome this bottleneck by allowing parallel activation of rules. When choices must be made due to space/time constraints or mutually exclusive actions, the choice is postponed until the last possible moment. This promotes rational decision making without arbitrary arbitration strategies. Also this parallelism may permit the development of extremely fast hardware implementations.

1.3 Apportionment of credit and Learning

In traditional expert systems, the value or rating of a rule relative to another rule is fixed by the programmer in conjunction with the expert or group of experts being emulated. The relative value of different rules is one key piece

of information that has to be learned. To facilitate this learning Classifier systems force classifier's to coexist in an information-based service economy. A competition is help among bidders where the right to answer relevant messages goes to the highest bidders, with the subsequent payment of bids serving as a source of income to previously successful message senders. The competitive nature of the economy ensures that good rules survive and bad rules die. The survival and strength of a rule is characterized by the amount of balance(credit) with it.

The division of payoff among contributing classifiers helps ensure the formation of an appropriately sized sub-population of rules. Thus different types of rules can cover different types of behavioral requirements without undue interspecies competition. In a rule-learning system of any consequence, we cannot search for one master rule. we must instead search for a set of co-adapted set of rules that together cover a range of behavior that provides ample payoff to the learning system.

1.4 Genetic Algorithm

The amount of balance/credit with a certain rule is dependent on the expected payoff of the rule. Hence the balance/credit can be used as a measure of fitness and a genetic search be applied to them, in an effort to find better rules. The genetic algorithm then uses a steady state replacement policy with crowding to replace the less fit rules with newly formed children from highly fit rules. We restrain ourselves from using a generational genetic algorithm as we risk losing existing rules that perform well.

Together, apportionment of credit via competition and rule discovery using genetic algorithms form a reasonable basis for constructing a machine learning system atop the computationally convenient and complete framework of classifiers.

Chapter 2

Mathematical Analysis of Classifier Systems

2.1 Analyzing the apportionment of credit scheme

First we shall outline the apportionment of credit scheme: First classifiers make bids B_i during the auction. Winning classifiers turn over their bids to the clearing house as payments P_i. A classifier may also have receipts R_i from its previous message sending activity or from environmental reward. In addition to bids and receipts, a classifier may be subject to one or more taxes T_i. Taken together, we may write a difference equation governing the depletion or accretion of the strength of ith classifier as follows:

$$S_i(t+1) = S_i(t) - P_i(t) - T_i(t) + R_i(t)$$

$S_i(t)$ the strength of the classifier i at time t. Next we quantify bids, payments and taxes. A classifier bids in proportion to its strength:

$$B_i = C_{bid} S_i$$

Here C_{bid} is the bid coefficient. Each classifier is taxed to prevent useless existence and thereby biasing the population toward productive rules. We tax a classifier, again in proportion to its strength:

$$T_i = C_{tax} S_i$$

Hence for an active classifier,

$$S(t+1) = S(t) - C_{bid}S(t) - C_{tax}S(t) + R(t)$$

$$S(t+1) = (1-K)S(t) + R(t)$$

where K = $C_{bid} + C_{tax}$. Z-transform analysis on the homogeneous part of the equation, reveals that the systems remains stable for bounded input if $0 \leq K \leq 2$. However in practice we take $0 \leq K \leq 1$ to enforce non-negativity of the classifier strengths.

The strength at time-step n of a classifier is given by

$$S(n) = (1-K)^n S(0) + \sum_{j=0}^{n-1} R(j)(1-K)^{n-1-j}$$

If the process continues indefinitely with a constant receipt $R(t) = R_{ss}$ we obtain the steady-strength by setting $S(t+1) = S(t) = S_{ss}$. This computation results in the following equation:

$$S_{ss} = R_{ss}/K$$

The steady bid is given by:

$$B_{ss} = \frac{C_{bid}}{K}R_{ss} = \frac{C_{bid}}{C_{bid} + C_{tax}}R_{ss}$$

Since C_{tax} is small $B_{ss} \approx R_{ss}$. In other words, for steady receipts, the bid value approaches the receipt. For time-varying receipt values, we see that the bid is a geometrically weighted average of the input. As such, it acts as a filter of the possibly intermittent and noisy receipt values.

2.2 Performance Analysis with Specificity dependent bidding

It is generally preferred that the rules evolved in the rule-set of the classifier system organize themselves into what is known as a *default hierarchy*. In a default hierarchy general rules (those with many #'s) cover the general conditions and more specific, possibly overlapping cover the exceptions. The formation of default hierarchies can be encouraged by the use of an appropriate bidding structure. Default hierarchies have two advantages over rule-sets that have non-overlapping rules:

1. Parsimony

2. Enlargement of the solution set

We can encourage the formation of a default hierarchy by making the bid proportional to the product of the strength and some linear function of specificity:

$$B_i = C_{bid} f(S_p).S_i$$

where $f(S_p) = b_1 + b_2 * S_p$. Then the steady stae strength of a rule is:

$$S_{ss} = \frac{R_s s}{C_{bid} f(S_p) + C_{tax}}$$

and the steady-state bid:

$$B_{ss} = \frac{C_{bid} f(S_p)}{C_{bid} f(S_) + C_{tax}} R_{ss}$$

In trying to evolve default hierarchies, we find that the fitness is distributed unequally among classifiers with different specificities. This would mean that a specific rule may have more expected payoff than a general one. But the low credit does not mean that the general rule is of no value. Thus the formation of a default hierarchy requires *crowding* that restricts the competition for population space among similar rules. Crowding also helps maintain diversity since the representative population is greatly reduced compared to the size of the problem space.

Chapter 3

Equivalence of recursive and non-recursive rule-sets

3.1 The input space

The aim of a learning classifier system is to find a set of rules that would map a set of given inputs onto a set of outputs. Rather than just finding a mapping using those rules, we would like the classifier system to generalize. If we limit our discussion to finite length messages over the binary alphabet, we see that the size of the input space is 2^l, where l is the length of the input messages. Taking $l = 4$, we obtain the position of messages 1101, 1000 and 1001 on a number line as shown in the following figure.

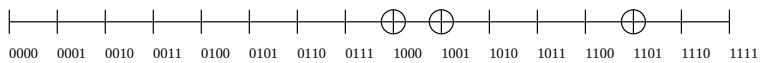

Figure 3.1: input space over a binary alphabet with length 4

For a message over the binary alphabet, the classifier is over a ternary alphabet, the additional alphabet being a wild card as mentioned previously in 1.2. A classifier spans a sub-space on the input space, whose size is equal to the number of wild cards in the condition. Figure 3.2 illustrates the sub-spaces spanned by the classifiers 1#1# and 11##. The sub-space spanned by a classifier need not be restricted to only adjacent points of the sub-space as we see in the case of 1#1#. It is to be noted that input space and the spanned sub spaces are discrete. The illustrations do not express this fact explicitly.

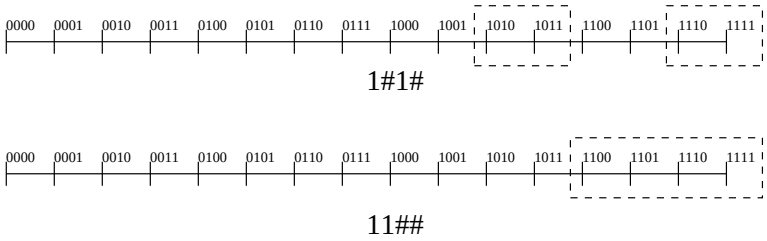

Figure 3.2: sub spaces spanned by 1#1# and 11##

By assigning a message to each condition and forming a classifier, we are in effect assigning different actions to different sub-spaces in the input space. Thus we are able to produce an arbitrary, possibly many to one, mapping from the input space to the output space. The learning classifier system thus classifies input messages based on the output messages. The internal mechanism used for this classification is the classifier. The use of a co-adapted set of rules in the classifier system that enable the classification of non-linearly separable problems.

Though all messages,classifiers and actions can be represented in a sufficiently large single dimensional space, If we were to characterize the messages and classifiers as being in an N-Dimensional hyperspace, we could think of classifiers as assigning actions to a single or a set of hyper-cuboids in N-Dimensional hyperspace.

3.2 Recursive rule sets

Recursive rule-sets are those that cause the generation of internal messages. Such systems require the maintenance of an internal message list. In other words, classifiers that are activated may post messages to the internal message list instead of effecting an action. Such systems may use many cycles of classifier matching for a single input message.

Such a mechanism adds no power to the underlying classifier mechanism. Such recursive rules only designate certain input messages as equivalent, thereby providing an alternative method of generalization. This generalization can however be achieved with extra rules, that use only the wild-card to generalize. Thus we find that classifiers that use non-recursive set of rules are as powerful as ones that use a recursive set of rules.

But implementing a classifier system with a recursive set of rules is more

complex, requiring the maintenance of a message list along with a strategy to prevent cycling. Recursive rule-sets are seemingly more difficult to evolve due to existence of cycling. Cycling leads to formation of many rules in the system that are unable to perform any useful work. Many potent rules may be wasted because they are in the middle of a cycle.

As a counter argument: recursive rule sets might be able to achieve high performance quickly, especially in cases where the total input subspace associated with a particular action is quite large. In such cases, the rules evolved have less restrictive mapping requirements as they can choose to map either into the output space or into any one element of the large subspace of the input space associated with the appropriate action. Evolution and search is much easier as the total search space is reduced in proportion to the size of the input subspace.

Chapter 4

Heuristics for adjusting the parameters of the Classifier System

In the following section some heuristics to adjust the parameters of the genetic algorithm and classifier system are described:

Population size generally needs to be large to allow enough material for the genetic operators to work on. Population sizes may be reduced, when the problem of maintaining diversity is addressed by increasing mutation rates or carefully choosing replacement strategies. A population size of 3000 was chosen. Selecting too large a population makes the program slow.

Classifier system iterations The number of times the apportionment of credit is tried is decided by the equation:

$$S(t+1) = (1-K)S(t) + R(t)$$

First the number of steps in which the above equation converges is found out for the maximum and minimum specificity values. The number of iterations is adjusted such that it is near the product of the number of input environmental messages and the greater of the number of steps obtained for convergence.

Bidding strategy may be chosen arbitrarily. However the implementation uses $b_1 = 0.9$ and $b_2 = 0.1$ to keep the difference in specificity

dependent bidding minimum, while achieving the necessary difference in bidding due to specificity. The values of C_{bid} and C_{tax} are 0.1 and 0.01 respectively. The payoff value is fixed at 10 such that the active classifier does not get an unfair advantage over others due to a large receipt.

Effective Bid is usually done by using*bidding noise* or*roulette wheel selection* among the matching classifiers to prevent unwanted biasing. In the implementation however, effective bidding is not employed. Children and parents are arranged such that children are matched before the parents so that children are matched firs and if they are not fit enough, the parents take over a few iterations later.

Other settings The values of other parameters used are:
$n = 500$
$P_{cross} = 1$
$P_{mut} = 0.01$
Selection: Tournament Selection
Replacement: Steady state replacement with crowding

Chapter 5

Implementation Issues

Genetic algorithms are generally quite slow because they deal with a huge populations. So quite often GA's are implemented to run on parallel computers such as SMP systems and cluster computers. Nevertheless, an increase in speed due to algorithmic optimization is always welcome. More so for genetic algorithms that use large populations and run on single processor configurations.

5.1 Identifying bottle-necks

In the learning process, the genetic algorithm plays a secondary role. The computationally intensive task is the classifier matching in during the apportionment of credit. A naive algorithm for matching classifiers and messages by sequentially scanning through them is computationally inefficient. Matching n messages with m classifiers in the rule-set, each of length l, takes time $O(lmn)$. In the following subsections we present different strategies for getting over this bottleneck.

5.1.1 Using pattern tries

One way to make the search process efficient is to make use of tries to search through existing patterns. In other words, we maintain a trie-index on the set of classifier conditions. At each step of matching, we can eliminate the unmatched alphabet. An example trie structure is shown in figure 5.1.

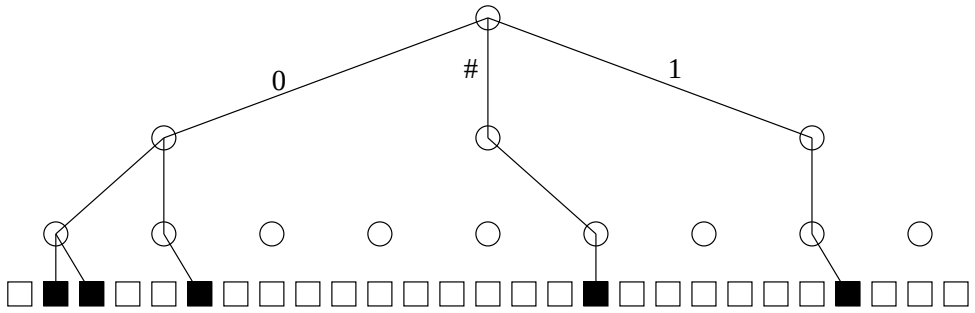

Figure 5.1: A trie that indexes 1#1, 0#1, 00#, #1# and 001

We search through the trie by starting a traversal at its root and proceeding through the branches that index matching alphabet and #. We traverse the whole of the tree, rather than stopping at the first matching pattern. This would take time proportional to the number of matching classifiers. The search time for each classifier is reduced by a constant factor in the best case. However the running time remains the same as that of the naive algorithm i.e $O(lmn)$. More improvement may be achieved using level compressed tries or PATRICIA tries.

5.1.2 Average case performance of the naive algorithm

On the other hand, the average case running time of the naive algorithm may still be acceptable. For matching a random condition with a random message, we can expect that we detect a mismatch at the third position on the average. This can be arrived at by considering the random variable $P(X = x) = \textit{Mismatch occurs at position } x$. The random variable X is geometrically distributed, with mean 3. So the naive algorithm performs quite well on the average case.

Though the population of a genetic algorithm (here the rule-set) cannot exactly be characterized as a random set of conditions, empirical results suggest that on the average only 20% of the condition in a rule and incoming messages are compared before detecting a mismatch.

5.1.3 Caching

During a single iteration of the genetic algorithm, it happens that a single message is generated multiple times by the environment for comparison.

Matching rule sub sets for a a particular message can be cached and used later as the matching subset does not change within an iteration. This caching can be done with the help of a hash table that caches rule subsets with the message as the key. Though unquantified, this approach produced a noticeable increase in performance.

This method is simpler and effective than using trie indexing as building and maintaining a multiply linked structure such as a trie is an effort in itself. As an extension one can use both trie-indexing and caching in an effort to glean more performance.

Chapter 6

Training set reduction using the Condensed Nearest Neighbor Algorithm

The CNN rule, one of a class of ad hoc decision rules, was motivated by statistical considerations pertaining to the nearest neighbor decision rule (NN rule).

The NN rule assigns an unclassified sample to the same class as the nearest of n stored, correctly classified samples. In other words, given a collection of n reference points, each classified by by some external source, a new point is assigned to the same class as its nearest neighbor. The most interesting theoretical property of the NN rule is that under very mild regularity assumptions on the underlying statistics, for any metric, and for a variety of loss functions, the large-sample risk incurred is less than twice the Bayes risk. (The Bayes decision rule achieves minimum risk but requires complete knowledge of the underlying statistics). From a practical point of view, however, the NN rule is not a prime candidate for many applications because of the storage requirements it imposes. The CNN rule is suggested as a rule with retains the basic approach of the NN rule without imposing such stringent storage requirements.

The consistent subset of a sample set, is which when used as stored reference set for the NN rule, correctly classifies all the remaining points in the sample set. A minimal consistent subset is a consistent subset with a minimal number of elements. Every set has a consistent subset, since every set is trivially a consistent subset of itself. Obviously, every finite set has

a minimal consistent subset, although the minimum size is not, in general, achieved uniquely. The CNN rule uses the following algorithm to determine the consistent subset of the original sample set. In general, however the algorithm will not find the minimal consistent subset. We assume that the original sample set is arranged in some order; Then we set up bins called STORE and GRABBAG as follows:

1. *The first sample is placed in* STORE
2. *The second sample is classified by the NN rule, using as a reference set the current contents of* STORE. *If the second sample is classified correctly, it is placed in* GRABBAG; *otherwise it is placed in* STORE.
3. *Proceeding inductively, the ith sample is classified by the current contents of* STORE. *If classifies correctly it is placed in* GRABBAG; *Otherwise it is placed in* STORE.
4. *After one pass thorugh the original sample set, the procedure continues to loop through* GRABBAG *until termination, which can occur in one of the two ways:*
   (a) *The* GRABBAG *is exhausted, with all its members now transferred to* STORE *(in which case, the consistent subset found is the entire original set), or*
   (b) *One complete pass is made through* GRABBAG *with no transfers to* STORE. *(If this happens, all subsequent passes through* GRABBAG *will results in no transfers, since the underlying decison surface has not been changed).*
5. *The final contents of* STORE *are used as reference points for the NN rule; The contents of* GRABBAG *are discarded.*

Qualitatively the rule behaves as follows: If the Bayes risk is small, i.e, if the underlying densities of the various classes have small overlap, the the algorithm will tend to pick out points near the (perhaps fuzzy) boundary between the classes. Typically, points deeply embedded within a class will not be transferred to STORE, since they will be correctly classified. If the Bayes risk is high, then STORE will contain essentially all the points in the original sample set, and no important reduction in sample size will have been achieved. No theoretical properties of the CNN have been established.

Chapter 7

Training set reduction using Genetic Algorithms

An alternate approach to training set reduction would be to apply Genetic Algorithms. In this case each possible subset of the dataset, represented as a bit-vector, forms the*chromosome*. The *fitness function* is the accuracy obatined when the NN-rule is applied to the reduced subset and the testing set. Additionally the number of set bits in the bit-vector is subtracted to get the final fitness. This is to direct the search towards the minimal consistent subsets.

Thus training set reduction may be performed on the dataset by using a GA to search among possible subsets of the training sets, While evaluating their fitness using the NN-rule.

Chapter 8

Results

Dataset	**Accuracy**	
	k-NN	GCS
Simulated Datasets		
Linearly Separable	100%	100%
Non Linearly Seperable	100%	100%
6-Multiplexer	100%	100%
Realtime Datasets[1]		
Lenses	100%	100%
Yeast	91%	90%
Yeast-Condensed (CNN)	92.5%	93%
Yeast-Condensed (GA)	93%	94%

Table 8.1: Results obtained for different datasets using k-NN and Genetic Classifier Systems

Chapter 9

Conclusion

The results suggest that Genetic Classifier Systems have considerable accuracy in classifying various kinds of data. Again genetic algorithms have also been sucessfully applied to the prototype selection problem. The amount of reduction obtained is slightly more than the reduction obtained by the CNN algorithm.

The above experiments demonstrate two uses of Genetic Algorithms: Prototype Selection using Genetic Algorithms and Pattern Classification using Genetics Based Machine Learning/Learning Classifier Sytems. Thus we see that Genetic algorithms are a viable method for prototype selection as well as pattern classification.

Chapter 10

References

1. Goldberg, David E.*Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Pub. Co. 1989.

2. Hart, Peter E.*The Condensed Nearest Neighbour rule.* Applied Physics Lab, Stanford Research Institute.

3. NN-rule paper

4. *The UCI Machine Leaning Repository*, www.ics.uci.edu/ mlearn/MLSummary.html