Chapter 14

Saman K. Halgamuge and Manfred Glesner
Darmstadt University of Technology
Institute of Microelectronic Systems
Karlstr. 15, D-64283 Darmstadt, Germany

saman@microelectronic.e-technik.th-darmstadt.de

Input Space Segmentation with a Genetic Algorithm for Generation of Rule Based Classifier Systems

Abstract
The rule based transparent classifiers can be generated by partitioning the input space into a number of subspaces. These systems can be considered as fuzzy classifiers assigning membership functions to the partitions in each dimension. A flexible genetic algorithm based method is applied for generation of rule based classifiers. It is shown that for complex real world types of applications, a preprocessing step with neural clustering methods reduces the running time of the genetic algorithm based method drastically. A heuristic method is compared to show the strength of genetic algorithm based method.

14.1 Introduction

The task of a classifier is to attribute a class to a given pattern which can be represented by measurements of some of its features. Thus a pattern can be seen as a vector in the pattern space of which dimensions are the measured features. Some of those dimensions are more relevant to distinguish between the classes while others are less useful. It would be interesting to remove unnecessary dimensions in order to simplify the pattern space and require less measurements. But the usefulness of a dimension is not always independent from the choice of the other dimensions.

In automatic generation of fuzzy rule based classifiers from data, the grade of importance of the inputs to the final classification result can be obtained, which leads to more compact classifier systems. The most important part of a fuzzy classifier is the knowledge base containing different parameters for fuzzification, for defuzzification and the fuzzy rules which contribute to the transparency. Those IF-THEN fuzzy rules contain terms like Low, Medium, High to describe the different features expressed as linguistic variables.

A rule based classifier can be seen as a group of hyper cuboids in the pattern space. Those hyper cuboids should represent parts of the space that belong to the same class. The elements used for the partition of the space can be either input data vectors or compressed clusters generated by artificial neural nets such as Radial Basis Function Networks (RBFN) [PHS+94] or Dynamic Vector Quantisation (DVQ) [PF91]. When learning vectors — or learning patterns — are concerned, they are seen as the limit case of clusters generated by neural networks with the forms of hyper cuboids or hyper spheres.

14.2 A Heuristic Method

This method is based on the analysis of variations of proportions of input vectors or clusters belonging to different classes in each dimension. Even though some information is lost due to the projection of the pattern space on the input dimensions this simplification makes the algorithm very fast. Since variations are to be calculated, a discrete approach has to be taken. The dimensions are to be cut into segments and the proportions of classes are to be computed for each segment. In this method, the lengths between two segmentation lines are initially equal. They begin to adjust when the heuristic method proceeds.

Both Figures 14.l(a) and 14.l(b) have in common that the slope of the border separating the classes 1 and 2 is close to 45°. Suppose that both dimensions are normalized to unity. These 2 figures are among the most difficult cases of partition and the ideal solution would involve first a change of both axes so that the slope would be about 0° or 90° steep. But in such a case the meaning of the input variables x and y would be lost. Since a transparent classifier has to be generated, rules must be easily understandable, therefore transformation of input variables must be avoided.

The slopes in Figure 14.1 indicate that one of the dimensions is slightly more important than the other. The steeper the slope, the more important the dimension. Since a decision has to be taken for the limit case (when none of the

dimension is more important than the other, that is for a 45° slope), this will give a threshold. Suppose that the limit case was divided into *ns* segments and that a decision has to be made. Since in this case the variation of proportions between two segments is always the same it is not possible to cut depending on the variations.

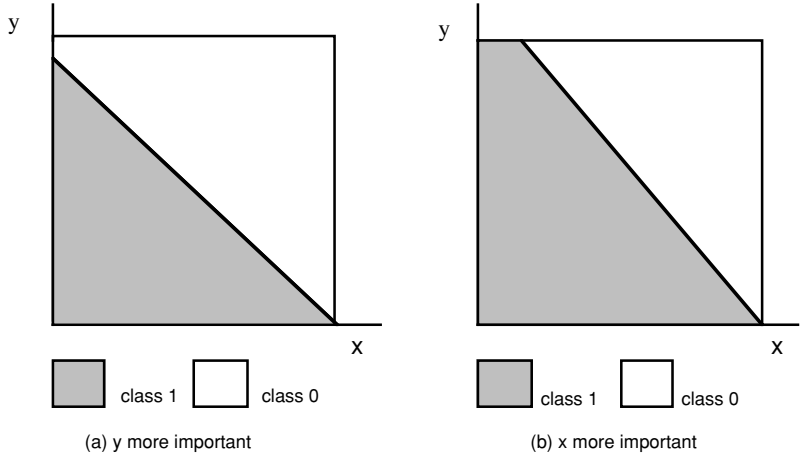

Figure 14.1: Defining a threshold.

A 45° slope corresponds to 100% of variation, if a very large number of partitions *ns* are allowed. At one end 100% of class 1 and at the other 0% of the class 1 are on the left of the cut. If the number of segments is *ns,* the threshold between 2 segments is 100%/*ns*.

The heuristic algorithm can be described as follows:

1. take the next dimension of the pattern space

2. divide this normalized dimension into *ns* equal segments

3. in each subspace generated by each segment, calculate the proportions of each class

4. if the variation of proportion between two neighboring subspaces for at least one class is greater than a given threshold, it is decided to cut this dimension between the two neighboring segments

5. go back to step 1 until last dimension is reached

In Figure 14.2(a), dimension x is divided into 4 segments and is cut between segments 2 and 3, and between segments 3 and 4. Proportions for class 1 varies from 100% in segment 1, over 80% in step 2 and 13% in segment 3 to 40% in segment 4. The variation in dimension x is higher than 100%/*ns* = 25% between segment 2 and 3 and between segment 3 and 4. In Figure 14.2(b), step 1 contains

70% of class 1, step 2, 76% step 3, 16% and step 4, 33%, hence the decision to cut between step 2 and 3. Segmentation and cuts of dimension *y* are independent from what has been with dimension *x*.

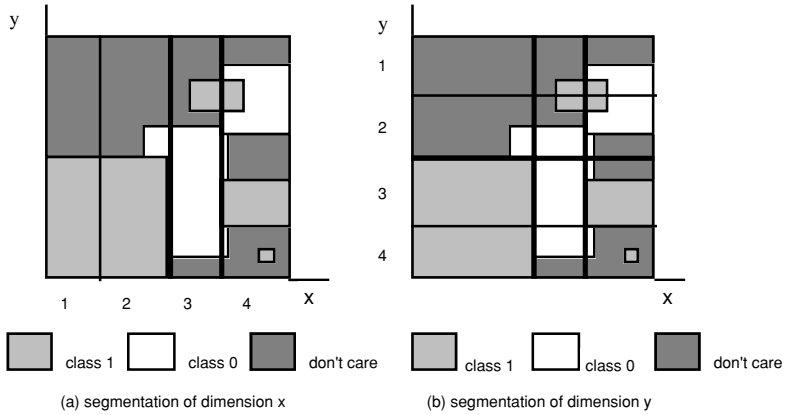

Figure 14.2: Segmentation of a two-dimensional pattern space.

This threshold value may vary according to the problem. If the threshold is too low, too many — sometimes irrelevant — cuts will be made and if the threshold is too high, some needed cuts could have been neglected, increasing the classification error. The range of empirical values is typically from 80%/*ns* to 180%/*ns*.

In order to evaluate the speed of this algorithm, it must be known that centers of subspaces have to be ordered in every dimension. Assuming that an ordering algorithm of order $s.\log(s)$ is used, the order of this method is: $d.s.\log(s)$, with s the number of subspaces and d the number of dimensions.

It is easy to see that this algorithm is fast but loses information because dimensions are treated independently, and that the accuracy of the partition cannot be better than the length of the segments.

14.3 Genetic Algorithm Based Method

Genetic Algorithms are solution search methods that can avoid local minima and that are very flexible due to their encoding and evaluation phases [Hol75, Gol89, BS93]. Indeed the form of a desired solution has to be encoded into a binary string so that a whole population of encoded possible solutions can be initialized at random. Evaluation is realized by a fitness function that attributes a value of effectiveness to every possible solution of the population. The best ones are allowed to exchange information through genetic operations on their respective strings. With this process, the population evolves toward better regions of the search space.

14.3.1 Encoding

In the partitioning problem, a solution is a set of cuts in some dimensions. It means that some dimensions can be cut many times while some are not at all. Therefore, strings are divided into blocs, each of them representing a cut in a dimension. The number of blocs in the strings is not limited so that the complexity of the partition can be dynamically evolved. Two strings with different lengths are shown in Figure 14.3.

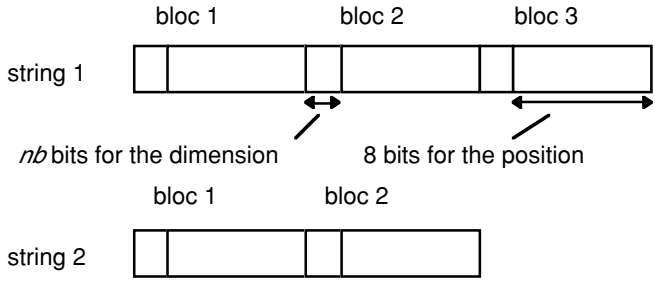

Figure 14.3: Strings and blocs.

In this figure, the *nb* first bits of a bloc encode the dimension that cuts and the 8 following bits encode the position of the cut in the dimension. The position of a bloc in a string is not important.

14.3.2 Genetic Operators

In addition to the widely used genetic operators mutation, crossover and deletion, authors also introduce "delete from one and insert in another" or theft. *mutation* — each bit in a string has a probability to be flipped crossover each bloc of a string has a probability to undergo a crossover. If so, a bloc of the same dimension has to be found in the second string chosen for reproduction, and a substring is exchanged. *deletion* — each bloc has a probability to be deleted. *insertion* — probability to insert a new bloc created at random. *theft* — probability for string 1 to steal a bloc at random from string 2 if both strings belong to a pair chosen for reproduction.

14.3.3 Fitness Evaluation

Defining the fitness function is the most important part of the method. Neither many cuts nor many rules are desirable. Both are interrelated but not the same. The number of subspaces must be as small as possible. For a given number of cuts, less subspaces will be generated if few dimensions are used. The upper limit for the number of subspaces *(ns)* is 2^{nc}, with *nc* the total number of cuts. Therefore, following terms are to be integrated in the fitness function:

$$\frac{1}{1+e^{(ns-ns_{th})}}$$

and

$$\frac{1}{1+e^{(nc-nc_{th})}}$$

The fitness falls when the number of subspaces or the number of cuts is above its thresholds ns_{th} and nc_{th} respectively.

Assuming clustered data with DVQ3 [HGG] and considering *gp* as the partition percentage, the percentage of points that are correctly separated to the hyper cuboids of appropriate classes:

$$gp = 100.\sum_{i=1}^{s}\sum_{j=1}^{v}\frac{\max_x\left(p\left(N_j,\vec{I}_x\right)\cdot V^s_{N_j,x}/V^t_{N_j}\right)}{\sum_{x=1}^{l}p\left(N_j,\vec{I}_x\right)\cdot V^s_{N_j,x}/V^t_{N_j}} \qquad (1)$$

$p(N_j, \vec{I}_x)$ is the density of probability that neuron N_j belongs to the class x of $\vec{I}_x$, s is the number of subspaces, v is the number of neurons (clusters), $V^s_{N_j,x}$ is the volume of neuron j belonging to class x, contained in subspace s and $V^s_{N_j}$ is the total volume of neuron N_j.

Considering the fact that probability density function (PDF) supplied by DVQ3 can be used to get the conditional probability $p(N_j|\vec{I}_x)$: given a data vector $\vec{I}_x$ of class x, it will activate neuron N_j:

$$p\left(N_j,\vec{I}_x\right) = p\left(N_j\middle|\vec{I}_x\right)\cdot p\left(\vec{I}_x\right) \qquad (2)$$

where $p(\vec{I}_x)$ is the density of probability that the input vector $\vec{I}_x$ is of class x. If all classes have the same probability, $p(\vec{I}_x) = 1/l$, where l is the total number of different classes.

The class that has the maximum of probability in one subspace determines its class. This maximum is divided by the total probability of this subspace (that is, the probability that a learning pattern happens to be found in this subspace, whatever its class) to calculate the ratio.

This ratio represents the "clarity of classification" for subspaces or the importance of subspaces for the corresponding classes. The goal is of course to get a high clarity of classification in all subspaces to prevent errors.

Since this procedure has to be made for all subspaces, it is the major time consuming part of the algorithm. The processing of every subspace is difficult due to the fact that the partition can be anything since none of its parameters are pre-determined. Therefore, a recursive procedure with pointers is used in simulation software. $p(N_j, \vec{I}_x)$ can be considered as a weight. Suppose 2 classes with the same probability, one of them occupying a much smaller volume than the other, which happens quite often when many dimensions are used.

One may wish to give their true probabilities to the different classes, with the risk that some classes could be neglected and considered as not important enough

if their probability is too low compared to the cost of making new segmentations. On the other hand, one can artificially increase the importance of one class, even if its probability is rather low, when, for instance, a particular class (e.g., meltdown in a nuclear plant) is more dangerous than the opposite. This method was implemented to solve a difficult case in section • There are many possibilities to define the fitness function which makes the method very flexible.

If input data are used instead of clusters generated by DVQ3, equation 1 is reduced to:

$$gp = 100.\sum_{i=1}^{s} \frac{\max_x\left(p\left(\vec{I}_x\right)\cdot V_x^s / V^t\right)}{\sum_{x=1}^{l} p\left(\vec{I}_x\right)\cdot V_x^s / V^t} \tag{3}$$

where V_x^s is volume of the part belonging to class x in subspace s, and V^t is the total volume. One more term was still added to fight back the strength of the two previous exponentials, setting another threshold for partition:

$$\frac{1}{1+e^{(gp_{th}-gp)/10}}$$

with gp_{th} a desired percentage of good partitioning. Note that gp_{th} can be set to values higher than 100%, even if gp will never get bigger than that. This can be done to move the equilibrium state to a higher number of partitioning without changing the goals regarding the number of cuts. It does not mean that a better quality can be achieved with the same amount of cuts since the number of segmentations increases, whenever the clarity of classification increases. It will just move the equilibrium toward more cuts while keeping a sharp cut in the fitness when reaching nc_{th}. If the desired clarity of classification cannot be achieved in this manner, *ncth* is also to be increased. Of course, if a high percentage of neurons are overlapping, this percentage will never be taken back by more segmentation.

The complete fitness function is:

$$\frac{gp}{\left(1+e^{(gp_{th}-gp)/10}\right)\left(1+e^{(ns-ns_{th})}\right)\left(1+e^{(nc-nc_{th})}\right)} \tag{4}$$

14.4 Results

14.4.1 Heuristic Method

Since the heuristic method is much faster, it is more interesting to use it for a large number of data, i.e., input/output learning vectors (even if its performance is at least as good when preprocessed hyper spheres or hyper cuboids are used). Two benchmarks are presented. The first one is an artificially created two-dimensional problem, where two classes made of 300 vectors with two input are

separated by a sinusoidal border. The second one is the well-known *Iris* data set [And35], containing 75 vectors in each training and test (recall) file, with 4 input divided into 3 classes. The result for the first benchmark is shown on Figure 14.4. With 5 cuts in dimension x and 3 cuts in dimension *y*, the partition percentage reaches 96%, which is quite good since the sinus has to be approximated by rectangles.

For the second benchmark, a 99% of partition was achieved for the normalized data set:

Dimension 1	0.33
Dimension 3	0.167 0.33 0.667
Dimension 4	0.33 0.667

For this problem, dimension 2 has been left out. Actually, dimension 1 and maybe dimension 4 could be removed from the partition and the separation of the different classes would still be satisfactory. It shows that the algorithm finds the relevant dimensions without removing from them the dimensions that are not strictly necessary.

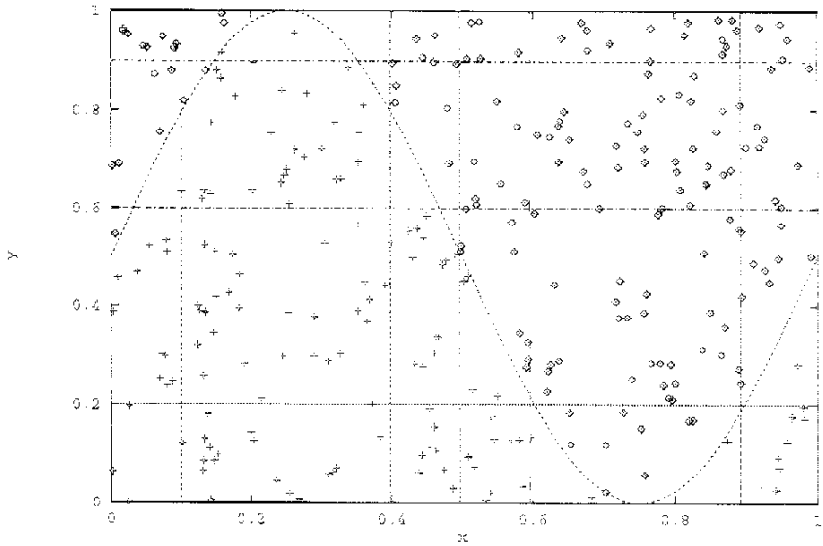

Figure 14.4: Sinusoidal boundary with heuristic method.

14.4.2 Genetic Algorithm Based Solutions

Since this algorithm is much slower — its order is exponential with the number of cuts — it can be interesting to use some data compression before the partitioning. Nevertheless, results shown here for comparison have been produced with 3 different types of input: the patterns themselves in all cases, clusters generated by RBFNs (RBF neurons) [PHS+94] for the benchmark *Artificial data* and the clusters generated by DVQ3 (DVQ3 neurons) for all the other problems.

The first benchmark is an artificial two dimensional case with 1097 training vectors where two classes are separated by one straight border at *xdimension1* = 0.4 [PHS+94]. The difference is that class 0 is separated in two disjoint areas by class 1 (see Figure 14.5).

This is a difficult case since the small class 0 area contains only about 2% of the 1097 points. If a cut is made at *xdimension1* = 0.4, a 98% of classification is already achieved with only one cut in one dimension. The heuristic method described will not recognize the smaller portion due to its approximation capability.

With usual parameters, the genetic algorithm will find the same approximation with one obvious cut. In a case where the class 0 can be of extreme importance, the genetic algorithm based solution allows the increase of importance in calculating the objective function as described in (section). So the probability of class 0 can be artificially increased, considering that the cost of not recognizing class 0 was higher than the cost of not recognizing class 1. With this safety measure two more cuts are made.

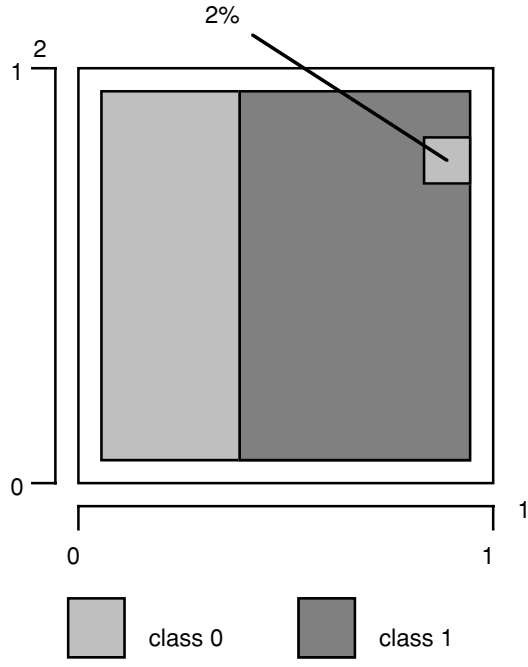

Figure 14.5: Benchmark *Artificial data.*

Figure 14.6 shows the different generations before and after making the correct 4 cuts partition for unclustered data.

Dimension 1	0.402 0.816
Dimension 2	0.707 0.872

Even if the number of data vectors is fairly large, 60 generations are produced in 30 minutes on a Sparc 10 station and 99% correct classification for both learning and recall sets was reached. The important parameters are: population = 21, gp_{th} = 140%, initial number of cuts = 7, limit for the number of cuts = 4 per dimension, probability of class 0 is twice higher than class 1's.

A high percentage could be already reached in the initial population. It is firstly due to the special strategy followed: the population starts with very "fat strings" (strings with many blocks) that are going to slim and lose their superfluous blocs. Secondly, this problem can easily be solved with one cut and the initialized population contains 147 cuts.

Making more generations would have finally made a 100% of classification, since it is possible to separate both classes totally and because the 4 cuts are already close to the optimum.

The next data for this problem were RBF nearest prototype neurons generated from the training data set [PHS+ 94]. With a population of 25 and a limit number of cuts by dimension set to 4, 99.5% for both learning and recall sets was made in 60 generations (30 seconds on a Sparc 10 station) (see Figure 14.7)

The first real world application is the *Solder* data file described in [HPG93] containing 184 data to be classified either as good or bad solder joints. There are 2 classes, and 23 dimensions (or features) extracted by a laser scanner. Clustered data with DVQ3 neurons are used first. The parameters are usual ones in the sense that it was not intended to find after many trials what values they should take in order to produce the best results.

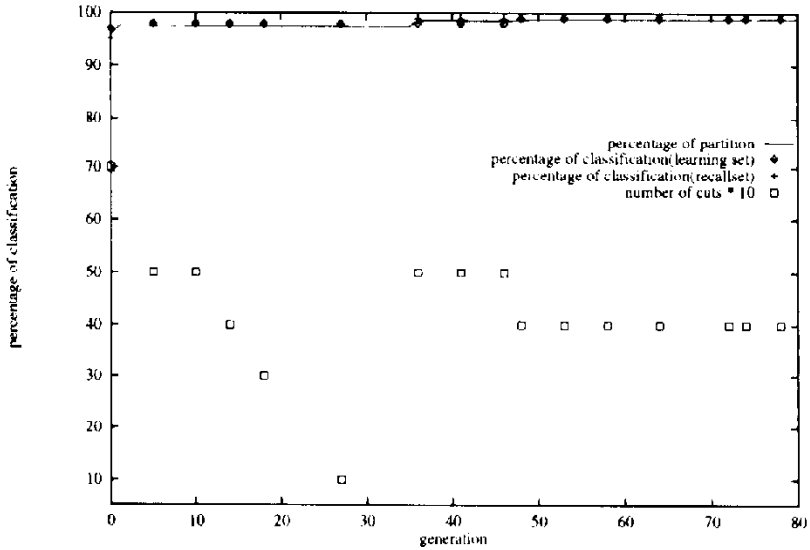

Figure 14.6: Performance using unclustered *Artificial data.*

The threshold gp_{th} = 120% is set with a maximum of around 6 cuts (0.26 cuts/dimension * 23 dimensions). The population was set to 15. The number of cuts is only 3 with the highest percentage of classification for the recall set too (96%) as shown in Figure 14.8. Ideally the program should have converged to this result instead of the (97%, 95%, 4 cuts) reached at the 151st generation. This is due to the fact that partition and classification don't exactly match. The fitness will improve if the number of cuts is increased from 3 to 4 in order to gain few percents in partition. This could have been probably avoided if the allowed number of cuts had been lower.

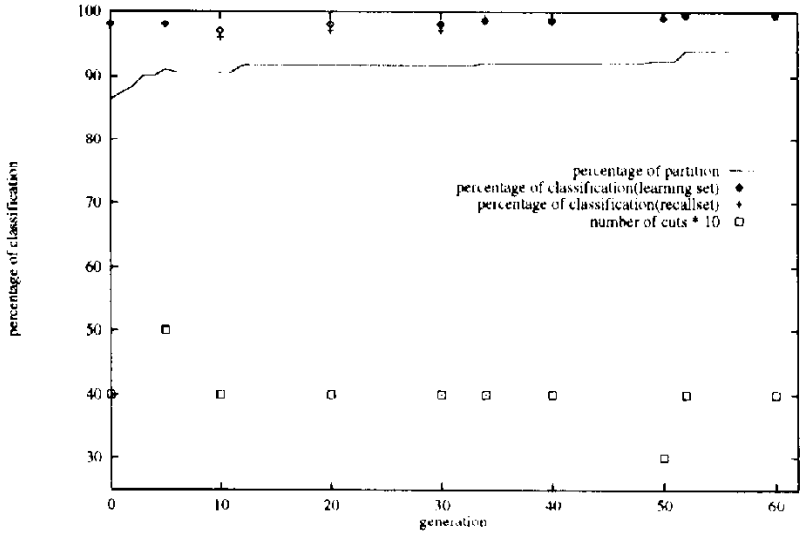

Figure 14.7: Performance using *Artificial data* clustered with RBFs.

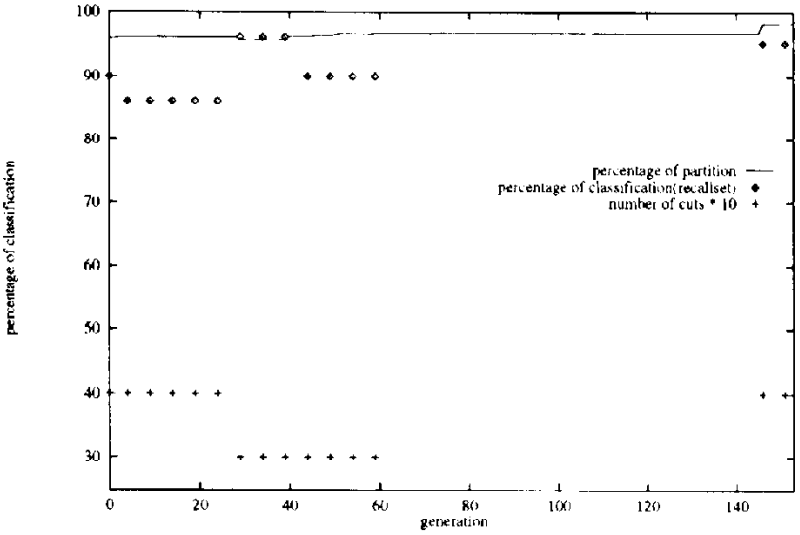

Figure 14.8: Performance with *Solder* data clustered by DVQ3.

If the patterns are used instead of neurons with the same parameters and a smaller (13) population, one may expect slightly better results since there is no loss of information due to the data compression and the partition almost reflects the real distribution of the patterns. It is to be seen in Figure 14.9 that classification results follow the partitions curve. The small difference is due to the small "noisy hyper volumes" that have been given around each data for generalization and calculation reasons. As a consequence the algorithm converges to the desired solution (98%, 98%, 3 cuts).

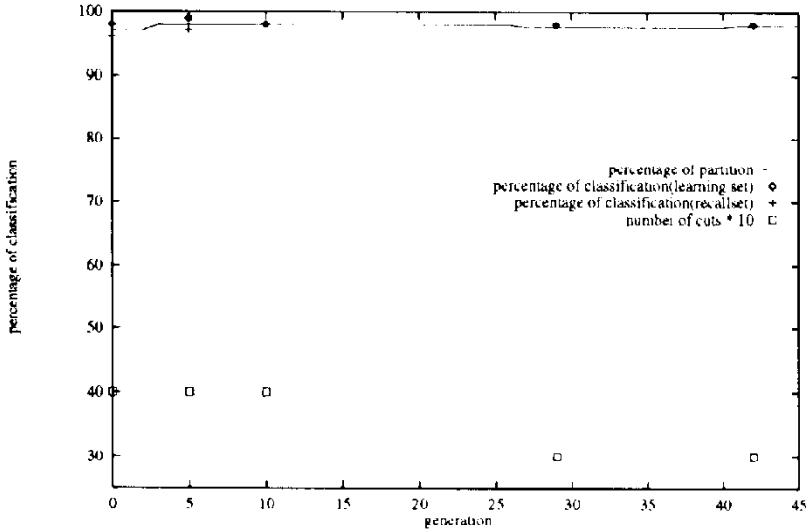

Figure 14.9: Performance with unclustered *Solder* data.

The second real world type application is more difficult: 10 different handwritten characters are to be distinguished in a 36-dimensional pattern space [HG94]. An initial unsuccessful effort is shown in Figure 14.10. The input data are DVQ neurons. All their radii are scaled by 1.3 to make a bit more certain that the patterns are contained by their hyper volumes. The different parameters were set in the normal range.

A good guess for the total number of cuts would be around 9 because there are 10 classes to separate. This parameter was set to 0.24 cuts/dimension*36 dimensions = 8.64 cuts. Since a high percentage is desired, gp_{th} = 120% is set as a goal.

The percentage of partitioning seems to settle to 90% and the number of cuts to 7. It seems to be harder to get higher than 90%, most probably because of overlappings. Those overlappings can be great either because of the use of many unnecessary dimensions or the inadequate scaling of neurons. The data themselves can be mixed too but more dimensions may result in less overlapping. Of course these reasons can all be there at the same time.

If few dimensions have to be used by setting the number of cuts allowed by dimension to 0.18 (for 36 dimensions, the fall in the fitness function is at 6.5 cuts) it takes about 10 minutes to obtain 110 generations. The radii have been

multiplied by 1.25 and the population size is 29. An offset between classification results and partitioning cannot be avoided due to the form of the neurons. The fact that the generalizing ability is very good for test set (99%) could show that neurons are adequately scaled.

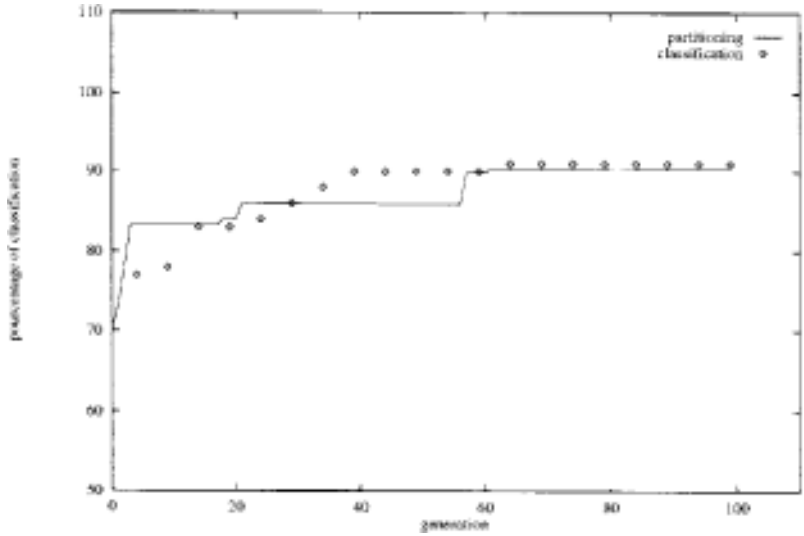

Figure 14.10: Unsuccessful trial with 9 cuts as a limit.

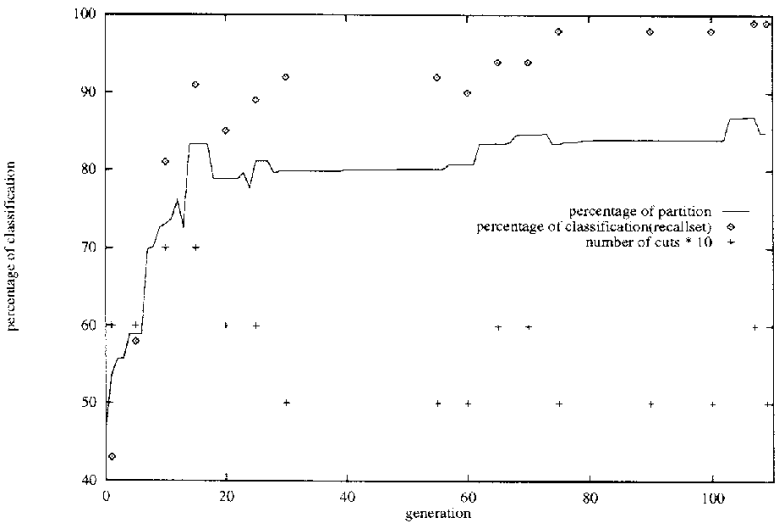

Figure 14.11: Performance with *Digit* data clustered by DVQ3.

The final solution needs only 5 dimensions and 5 cuts to achieve 99% and 94% of classification for training and recall sets, respectively. It must be said that if

the 1005 learning vectors are used instead of the few DVQ3 neurons, it took 9 hours on the same machine to achieve the same result.

If the data set with 1005 vectors are used, the program needs 9 hours on a Sparc 10 station to make 60 generations, with a population of 15. With the same parameters, the results shown in Figure 14.12 are obtained.

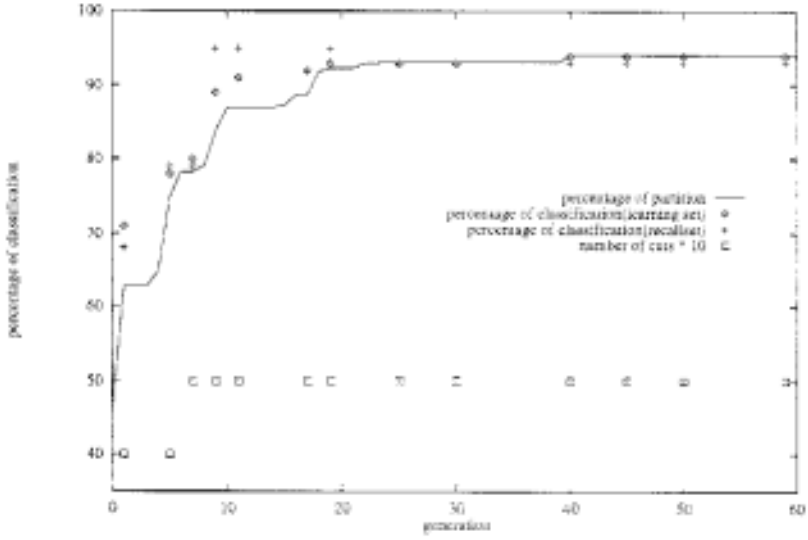

Figure 14.12: Performance with unclustered *Digit* data.

Discussion

In this paper, the importance of the partition of the pattern space has been stressed because it leads to efficient and compact classifiers at a very low cost if the number of cuts and of dimensions can be somehow reduced.

At this stage, the genetic algorithm, which is much slower than the heuristic method, could achieve the best partitions. Because the heuristic method uses projections of the space and has a discrete approach: it suffers from losses of information, lack of precision and is quite sensible to noisy variations of the vectors distribution in the space. However, its speed allows many iterations and some search strategy to get better results. Of course it cannot find the necessary dimensions among all the relevant dimensions and this problem has not been solved yet. Nevertheless, the heuristic method can be applied to a number of problems before moving to more global time consuming genetic algorithm based methods.

References

[And35] E. Anderson. The Irises of the Gaspe Peninsula. *Bull. Amer. Iris Soc.*, 59:2-5, 1935.

[BS93] Th. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for pargreeter optimization. *Evolutionary Computation,* 1(1):1-23, 1993.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[HG94] S.K. Halgamuge and M. Glesner. Neural Networks in Designing Fuzzy Systems for Real World Applications. *International Journal for Fuzzy Sets and Systems* (in press) (Editor: H.-J. Zimmermann), 1994.

[HGG] S.K. Halgamuge, C. Grimm, and M. Glesner. Functional Equivalence Between Fuzzy Classifiers and Dynamic Vector Quantisation Neural Networks. In *ACM Symposium on Applied Computing (SAC'95)* (Submitted), Nashville, USA.

[Ho175] J.H. Holland. *Adaptation in Natural and Artifiical Systems.* The University of Michigan Press, 1975.

[HPG93] S. K. Halgamuge, W. Poechmueller, and M. Glesner. A Rule based Prototype System for Automatic Classification in Industrial Quality Control. In *IEEE International Conference on Neural Networks' 93,* pages 238 243, San Francisco, U.S.A., March 1993. IEEE Service Center; Piscataway. ISBN 0-7803-0999-5.

[PF91] F. Poirier and A. Ferrieux. DVQ: Dynamic Vector Quantization — An Incremental LVQ. In *International Conference on Artificial Neural Networks'91,* pages 1333-1336. North Holland, 1991.

[PHS+ 94] W. Poechmueller, S.K. Halgamuge, P. Schweikeft, A. Pfeffermann, and M. Glesner. RBF and CBF Neural Network Learning Procedures. In *IEEE International Conference on Neural Networks' 94,* Orlando, U.S.A., June 1994.