

# Genetic Algorithm Selection of Features for Handwritten Character Identification<sup>1</sup>

Roger S. Gaborski<sup>2</sup>

Health Science Research Lab  
Eastman Kodak Company  
Rochester, New York 14653-2123 USA

David G. Tilley<sup>4</sup>

Imaging Research Laboratories  
Eastman Kodak Company  
Rochester, New York 14653-5722 USA

Peter G. Anderson<sup>3</sup>

Computer Science Department  
Rochester Institute of Technology,  
Rochester, New York 14623-0887

Christopher T. Asbury<sup>5</sup>

Computer Science Department  
Rochester Institute of Technology,  
Rochester, New York 14623-0887

<sup>1</sup>Presented at The International Conference on Artificial Neural Networks & Genetic Algorithms "ANNGA 93," Innsbruck, Austria, April 1993.

<sup>2</sup>[gaborski@kodak.com](mailto:gaborski@kodak.com)

<sup>3</sup>[pga@cs.rit.edu](mailto:pga@cs.rit.edu)

<sup>4</sup>[tilley@kodak.com](mailto:tilley@kodak.com)

<sup>5</sup>[cxa9194@cs.rit.edu](mailto:cxa9194@cs.rit.edu) (Mr. Asbury was a co-op student at Kodak.)

### **Abstract**

We have constructed a linear discriminator for hand-printed character recognition that uses a (binary) vector of 1,500 features based on an equidistributed collection of products of pixel pairs. This classifier is competitive with other techniques, but faster to train and to run for classification.

However, the 1,500-member feature set clearly contains many redundant (overlapping or useless) members, and a significantly smaller set would be very desirable (e.g., for faster training, a faster and smaller application program, and a smaller system suitable for hardware implementation). A system using the small set of features should also be better at generalization, since fewer features are less likely to allow a system to “memorize noise in the training data.”

Several approaches to using a genetic algorithm to search for effective small subsets of features have been tried, and we have successfully derived a 300-element set of features and built a classifier whose performance is as good on our training and testing set as the system using the full set.

## Introduction & Summary

Starting with a hand-written digit recognition algorithm, with a 97%–99% correct recognition rate, which functions as a linear discriminator based on a collection of 1,500 binary features extracted from a  $30 \times 20$ -pixel array, we determine some 300-element subsets of those 1,500 features and build a new classifier which also achieve the same recognition rate. (Similar experiments are reported in [2]. who studied *all* subsets of a set of 15 features, and in [7] who searched for an optimal subset of 30 features. Both of these papers used the size of the chosen features subset to penalize the GA fitness function. We froze the size of our feature subset at 300.)

Two separate experiments were performed. One involved using the features subset (i.e., the “individual” of the genetic algorithm) to train and test a classifier (the result of this testing is the genetic algorithm’s “fitness” value). The other involved using the  $10 \times 1,500$ -element discrimination matrix we had for the good classifier and extracting a  $10 \times 300$ -element submatrix according to the GA individual. This second experiment short-circuits any training; it goes directly to testing. The first experiment has been eminently successful, creating small classifiers with an error rate (i.e., wrong classification) of under 1%. The second experiment was never able to achieve better than 3% error rate; we will not pursue this topic further here.

The smaller feature set is desirable for runtime space and speed of the algorithm as well as for potential hardware implementation [5]. Additionally, a smaller feature set would not so easily over-fit the training exemplars (“memorize the noise in the training data”), thus providing expectation of better generalization.

We had tried to reduce the feature set manually during development of the original recog-

nizer, since another disadvantage of large feature sets is that an  $F$ -element feature set implies that the training algorithm must invert an  $F \times F$  array, a process whose time complexity is  $O(F^3)$ . The manual search was unsuccessful. As well as being a massive search space, it was also difficult to effectively represent the sets of features. A rule for features evenly spaced over a character’s image was easy to derive and state. Cleverly constructed subsets, perhaps problem dependent ones, were not.

Our search for good feature subsets was conducted using several versions of a genetic algorithm. We varied the strategies for population management, child creation, and parent selection, as well as the parameters for population size and mutation rate. The results were, probably, what one should have expected (except for the success of locating feature subsets whose performance matched that of the full 1,500 set); i.e., larger population sizes combined with more conservative parent selection rules resulted in higher quality results which took longer to achieve. But the exercise gave us quantitative statements of these principles (to the extent that a number of experiments could be run, each of which took the full power of a SUN SPARCstation 2 for several days).

As we experimented and learned, we, naturally, discarded some approaches and concentrated on others which appeared to work for our particular problem. In the directions which seemed the most fruitful, we continued to experiment with the structures. All of this is detailed below.

## The Polynomial Algorithm

We have been working with a learning system applied to the classification of hand-written alphabetic characters known as the Polynomial Method [6]. This system extracts a binary vec-

tor of features from a normalized character, and then classifies the character using a matrix multiplication and then selecting the index of the largest component of the product vector as the unknown character’s classification.

Srinivasan described a learning algorithm to determine the above matrix based upon a classical least squares error minimization using a data base of correctly labeled training exemplars and the associated target product vectors, which are simply standard unit vectors with the single non-zero entry in the labeling position.

The features used are, generally, the logical products (“and”) of two nearby pixels (hence “polynomial”). We have been able to create accurate classifiers using 1,500 features resembling dilations of the king and knight chess moves in a  $30 \times 20$  array of pixels. Each pixel could be the center of up to eight such chess moves, which gives nearly 4,800 possible features. The feature count gives the size of a matrix which must be inverted, so we chose a relatively equidistributed set of these features to hold the total to a manageable yet effective number.

Specifically, suppose we have  $N$  labelled character training exemplars for  $K$  character classes and  $F$  features. (We have  $N = 100,000$ ,  $K = 10$ , and  $F = 1,500$ .) Construct an  $F \times N$  matrix,  $X$ , whose  $N$  columns are the  $F$ -element feature vectors of the training exemplars, and a  $K \times N$  matrix,  $Y$ , whose  $N$  columns are the  $K$ -element “target vectors” corresponding to the correct classifications for the respective training exemplar. A target vector for a character of classification  $k$  has value 1 at subscript position  $k$  and 0’s elsewhere. We determine a “classification matrix,”  $A$ , which satisfies, in the least-squares-error sense,

$$AX = Y \quad (1)$$

This is achieved using the Moore-Penrose

pseudo inverse,

$$A = YX^T(XX^T)^{-1} \quad (2)$$

Character recognition is achieved using this classification matrix  $A$  by extracting a feature vector  $\bar{x}$  from an unknown character, calculating

$$\bar{y} = A\bar{x} \quad (3)$$

and assigning classification  $k$  to the unknown character, where

$$\bar{y}_k > \bar{y}_i, \forall i \neq k \quad (4)$$

We augmented the basic polynomial method, outlined above, using an iterative technique inspired by perceptron and adaline training (see [1]). Simply stated, our method strives to determine the training exemplars that are near the boundaries for their particular classification and builds the matrices  $X$  and  $Y$  shown in (1) and (2) with those boundary cases over-represented. Although the resulting classifier is constructed using a least-squares-error rule, it functions more like one whose goal is to achieve more correct classifications than one whose goal is to identify and separate classification clusters based on centers of mass. This approach improved the performance of the 1,500-feature classifier from 97.06% correct on the testing data to 98.72%. Or, in other words, the incorrect classification rate is reduced from 2.94% to 1.28%.

## GA Hill-Climbing

We have been experimenting with genetic search algorithms to locate suitable classifiers using  $F = 100, 300$ , and 500 features. The search space is gigantic (e.g., there are  $\binom{1500}{300} \approx 5 \times 10^{144}$  possibilities using 300

features), so genetic search seemed particularly appropriate. An “individual” in the gene pool is a subset of  $F$  of the 1,500 features we found to be already useful. The “fitness function” is the classification accuracy (percent correct) of a classifier built using Srinivasan’s one-shot learning technique, where the system is built using a training set consisting of 30,000 hand-written digits and a testing set of 30,000.

Fitness evaluation is the principal timing bottle-neck in this process. We tried various approaches to this problem; these are described below.

(See [3] and [4] for detailed treatments of the theory of genetic algorithms.)

## Creating Children

In our GA experiments, we represent individuals as a sequence of 1,500 1’s and 0’s in which the number of 1’s is exactly  $F$ . Two individuals are chosen to become parents according to their fitnesses (competition techniques for choosing individuals for parenthood are described below). We use a form of uniform crossover [4] to combine genetic material from the chosen parents to create the children. In this process, if both parents agree (0 or 1) at some position, the two children will both inherit the value that the parents agree on. If the parents disagree at a position, the two different values are assigned to the two children randomly. However, we do ensure that each child is created with exactly  $F$  1’s.

## Selecting Parents

Parents are selected from the current population of individuals according to their fitness. Fitness is, of course, the performance on a set of testing data of a classifier built using the

individual’s one bits to select the classifier’s features. We choose individuals to be parents with the probability of being chosen a monotonically increasing function of their fitness.

The first interpretation of “monotonically increasing function of fitness” is that individuals may be chosen to be parents *proportional to their rank* when rank-ordered by fitness. This could be implemented by sorting the population according to fitness, and then choosing individuals’ ranks by a simulated biased roulette wheel [4]. A simpler method, which achieves the same result is to pick two individuals from the population (picking with uniform distribution), and then choose the one with greater fitness.

In our initial experiments this rule took the following form. We use a random shuffling procedure to select four individuals. The fittest of the first pair and the fittest of the second pair are chosen as the two parents.

Later experiments were generalizations of this first one. We shuffle and select  $2n$  individuals; the fittest of the first  $n$  and the fittest of the second  $n$  are chosen as the two parents. (The probability theory involved here is this. Suppose that a random number from the interval  $[0, 1]$  has uniform distribution; i.e.,  $f(x) \equiv 1$ . Then, let  $x = \max(x_1, x_2)$ , where  $x_1$  and  $x_2$  are chosen independently from the uniform distribution; then the probability distribution for  $x$  is  $f(x) = 2x$ . In general, if  $x = \max(x_1, x_2, \dots, x_n)$ , then the probability distribution,  $f(x) = nx^{n-1}$ .) For very large  $n$ , this will degenerate to the choice of the two fittest individuals to serve as parents. This would cause a rapid loss of “genetic material” with a consequent failure to search large portions of the space of all individuals. As we expected, the quality of the solutions deteriorates with larger  $n$ , but, as a trade-off, the speed of reaching good solutions increases. Tables 1 and 2 give experimental results.

## A Sequence of Populations

We experimented with several different methods for maintaining populations. We used the notion of distinct generations as well as that of a single, evolving population. GA folklore seems to indicate that these algorithms are sufficiently robust, that almost any approach will give solutions to the optimization problems of equivalent quality, but that some methods may converge to the champion much faster than others. Some of our experiments can take weeks, so we wanted to search out the rapid techniques. The first four population methods we tried are:

- *Steady state.* This was the first technique that we put together. When the two children are created, they replace the two losers of the two two-way competitions that selected the two parents.
- *Replace two worst.* This method is similar to the first, in that there is a single evolving population. However, here we replace the two worst fit individuals of the population with the two new children. (We do assure that neither of the parents are one of the two worst performers.)  
Both the first and the second method assure that the current fitness champion stays in the population pool.
- *Simple generational.* In this method, we introduce the notion of “generation,” where the children of the current population are used to create an entire new population. Here, the current champion can be lost.
- *Keep best half.* This fourth method is a variation on the third. The children of parents in generation  $n$  are used to build generation  $n+1$ . Then, the union of these two generations is sorted by fitness, and

the top (fittest) half of becomes generation  $n+1$ . Again, we assure that the best performers are not lost from the population pool.

## Fitness Testing

In this section, we describe the sequence of experimental set-ups we used, and how we addressed problems of efficiency.

In order to save save training and testing time in the iterated polynomial algorithm, feature vectors (1,500 features per) were extracted from all the training and testing exemplars and cached in disk files. So, our genetic search program used these “features files” instead of actual characters. For every genetic individual to be fitness tested, we read 30,000 feature vectors from the training set, extracted the indicated features to form a training vector of the desired length (generally 300), and we formed the classification weights matrix  $A = YX^T(XX^T)^{-1}$ , (equation (2)), then processed 20,000 characters of the testing set, by similarly extracting the subset of the features and multiplying the smaller vector by  $A$ . The fitness for the individual in question was the fraction of correctly classified characters. We took that fraction, multiplied by 1,000, and converted to an integer, to represent the fitness. Thus, we have fitness values in the range 8000 to 9900.

It occurred to us that we could achieve the same testing if we were to pre-evaluate the two matrices,  $YX^T$  and  $XX^T$ , for the full set of 1,500 features, which are then used in the evaluation of  $A$ . The components for evaluation of the  $A$ -matrix corresponding to a selected subset of the full feature set is simply the obvious submatrices of these large  $YX^T$  and  $XX^T$  matrices. This pre-evaluation takes one or two hours, depending on the computer chosen and its current load, but it would double the speed

for individual fitness evaluation. When the development of this technique stabilized, we were able to cache these two arrays on disk, with only the I/O cost as overhead.

The one-shot training (Srinivasan’s algorithm) using 30,000 training exemplars and 1,500 features resulted in a classifier with 97% accuracy on the 20,000 character testing set. Our genetic search was able to find feature subsets of size 300 that performed slightly better on this testing set. (Caveat: genetic hill-climbing uses a fitness measure which is derived using what we had called the “testing set.” This set is now intimately involved with (genetic) training, although the individual elements do not directly affect the classifier we are building. To be totally fair, a third set of characters is needed for final testing.)

We had hoped that using the good small subset of features iteratively in the polynomial training algorithm we would be able to achieve an even better classifier. That hope did not materialize. The one-shot trained  $A$ -matrix was as good as we could get with that technique.

What had started out as simply a time saving measure—caching the large matrices—suggested a new approach which has led us to a better result. The iterative polynomial training technique develops the two matrices  $YX^T$  and  $XX^T$ , but had no plans to save them. Now they were valuable: we re-ran the training which gave us 98.71%, saved the two large matrices, and used them with our genetic search. This approach was successful. We were able to discover a set of 300 of the 1,500 features that yielded 98.8% (same caveat as above).

## Summary of Results

After preliminary experiments had been tried, we settled on the use of the data mentioned

above; namely the arrays  $YX^T$  and  $XX^T$  that were a byproduct of the training session of the polynomial algorithm which produced the digits classifier whose performance was 98.71%. We had determined that a continuous population evolution rather than explicitly separate generations was desirable (separate generations made children of good performers wait too long to have good children of their own). And, the strategy of always replacing the two least fit individuals whenever two new children were produced worked best.

Our goal was a classifier which used only 300 of the 1,500 features. The available parameters were the population size, the parent-choosing method, and the mutation rate.

We tried population sizes of 100, 200, and 300. A population of 100 converged rapidly, but we did not achieve the best we could; 200 was better than 100; and 300 worked marginally better than 200, but took a long time to find the performers we were seeking.

We selected parents using the “best of  $N$ ,” taking two disjoint subsets of  $N$  individuals and choosing the most fit of each subset to be the parents. Larger values of  $N$  heavily skew the distribution to the right, causing rapid convergence combined with a degradation of the fitness of the individual converged to.  $N = 3$  seems to be the optimal point to trade off these two concerns.

Figure 1 shows the hill-climbing achieved by the genetic algorithm as we evolved a population of 300 individuals. Parent selection was performed by locating the most fit in two disjoint three-member competitions (“best of three”). The two new children always replaced the two least fit individuals in the current population.

Tables 1 and 2 show the results of several experiments. They do not show monotonic results as we have described above. The specific results depend, to some extent, on the outcome of the pseudo-random-number used

(*drand48()*), and several runs with different random seeds for each parameter would presumably smooth out these artifacts. Our approach was to sample the parameter space as widely as we could rather than to repeat the experiments for fixed parameters.

	Population size		
competition	100	200	300
best of 2	3,700	16,000	25,000
best of 3	3,300	9,100	18,000
best of 4	1,450	4,850	10,900
best of 5	1,400	4,650	9,200

Table 1: Number of individuals evaluated until convergence. (The first column shows the size of the parent selection tournament.)

	Population size		
competition	100	200	300
best of 2	9837	9893	9901
best of 3	9841	9884	9894
best of 4	9793	9845	9872
best of 5	9802	9856	9876

Table 2: Fitness of the best individual discovered.

We experimented a little with mutation. Our interpretation of this operator was that, after two new children were created, we would *mutate* a specified number of times, which we denote by *MUTATE\_COUNT*. A single *mutate* step consists of randomly choosing two bits in the string of length 1,500 and interchanging their values. The probability that a single *mutate* will actually modify an individual is 0.32.

With a *MUTATE\_COUNT* of three to nine, we were able to discover slightly better individuals, and to discover high performing indi-

viduals in approximately the same time as it took to find the best (and converge) with no mutation (Table 3). (When we use mutation, our search procedure never ‘converges.’)

	MUTATE_COUNT			
competition	0	3	6	9
best of 4	9793	9809	9802	9800
best of 5	9802	9802	9827	9804

Table 3: Fitness of the best individual after 1,500 evaluations using various levels of mutation

A *MUTATE\_COUNT* of 30 completely spoils hill climbing. Any successes with such a high rate of mutation seems to be attributable to nothing more than luck.

Figure 2 shows the feature subset in the best we have found. Each feature is the logical product of two factors, where each factor is the logical sum of three pixels. The three pixels’ centers are a scaled up version of a chess king’s or knight’s move. The figure shows the chosen “chess moves” as line segments connecting the two centers.

## References

- [1] Peter G. Anderson and Roger S. Gaboriski, “The polynomial method augmented by supervised training for hand printed character recognition,” *Proceedings of the International Conference on Neural Networks and Genetic Algorithms* 1993.
- [2] Eric I. Chang, Richard P. Lippmann, and David W. Tong, “Using genetic algorithms to select and create features for pattern classification,” *Proceedings of the International Joint Conference on Neural Networks*, 1990.

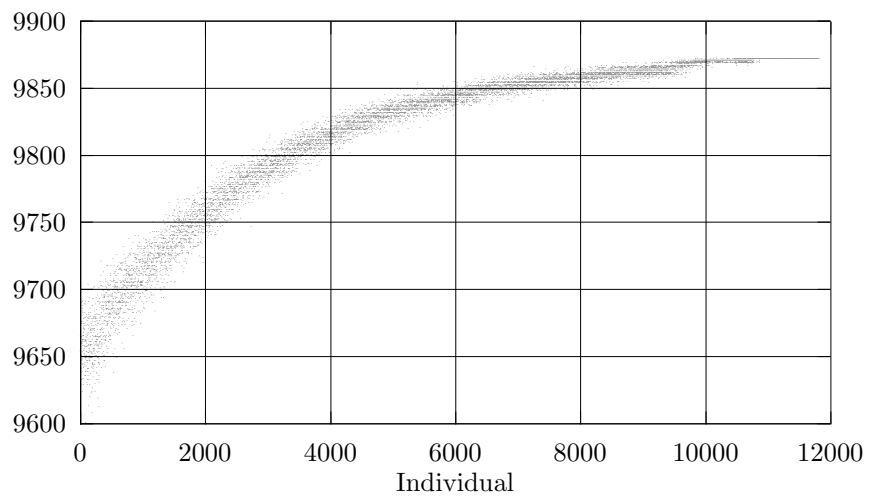


Figure 1: The fitness of each of the individuals encountered in our search for the best 300-element subset of features.

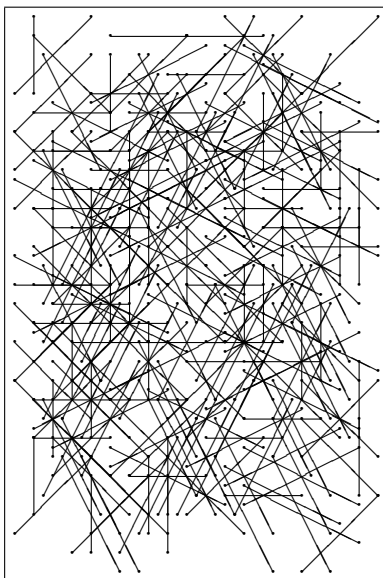


Figure 2: The 300 features of the best feature subset we have located.

- [3] Lawrence Davis (ed.), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.
- [4] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, New York, 1989.
- [5] A. Rao, P. G. Anderson, R. S. Gaborski, and K. S. Jaiswal, "A hardware polynomial feature net for handprinted digit recognition," *Proceedings of the Third IEE International Conference on Artificial Neural Networks*, 1993.
- [6] Uma Srinivasan, "Polynomial discriminant method for handwritten digit recognition," *SUNY Buffalo Technical Report*, December 14, 1989.
- [7] W. Siedlecki and J. Sklansky, "Constrained genetic optimization via reward-penalty balancing and its use in pattern recognition," *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.