USING GENETIC ALGORITHMS TO SELECT AND CREATE FEATURES FOR PATTERN CLASSIFICATION

by

Eric I-Chao Chang

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements
for the Degrees of

Bachelor of Science and Master of Science

at the

Massachusetts Institute of Technology

May 1990

Signature of Author Signature redacted
Department of Electrical Engineering and Computer Science
June 1990

Certified by Signature redacted
Dr. Richard P. Lippmann
Staff at Lincoln Laboratory
Thesis Supervisor

Certified by Signature redacted
Dr. David W. Tong
Electrical Engineer, GE Corporate Research and Development
Thesis Supervisor

Accepted by Signature redacted
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Acknowledgements

I would like to thank Dr. Richard Lippmann and Dr. David Tong for their enthusiastic support and helpful criticisms. Without their generous encouragement, this research would not have been completed. I am grateful for the knowledge and insight I received from them about the research topic, the proper way to conduct and present one's research, and life in general! I also enjoyed fruitful discussions with William Huang, Kenneth Ng, Dr. Ben Green, and Dr. Henzer Chen. Dr. Henzer Chen and William Huang also graciously provided data for me to conduct the experiments. I appreciate the good fortune of working with wonderful colleagues at GE Corporate Research Center and Lincoln Laboratory who made this research experience educational, enjoyable, and memorable. Finally, thanks to my parents, Mr. and Mrs. Ying-Fu Chang, for their faith in me. Although they may not understand the concepts in this thesis or what I learned at M.I.T., they supported my studies with understanding and trust. May the completion of this thesis give them as much pride and joy as they have given me.

USING GENETIC ALGORITHMS TO SELECT AND CREATE FEATURES FOR PATTERN CLASSIFICATION

by

Eric I-Chao Chang

Submitted to the Department of Electrical Engineering and Computer Science on June 1990 in partial fulfillment of the requirements for the Degrees of Bachelor of Science and Master of Science

Abstract

Genetic algorithms were used to select and create features and to select reference exemplar patterns for machine vision and speech pattern classification tasks. On a 15-feature machine-vision inspection task, it was found that genetic algorithms performed no better than conventional approaches to feature selection but required much more computation. For a speech recognition task, genetic algorithms required no more computation time than traditional approaches but reduced the number of features required by a factor of five (from 153 to 33 features). On a difficult artificial machine-vision task, genetic algorithms were able to create new features (polynomial functions of the original features) which reduced classification error rates from 19% to almost 0%. Neural net and nearest neighbor classifiers were unable to provide such low error rates using only the original features. Genetic algorithms were also used to reduce the number of reference exemplar patterns and to select the value of k for a k nearest neighbor classifier. On a 338 training pattern vowel-recognition problem with 10 classes, genetic algorithms simultaneously reduced the number of stored exemplars from 338 to 63 and selected k without significantly decreasing classification accuracy.

In all applications, genetic algorithms were easy to apply and found good solutions in many fewer trials than would be required by an exhaustive search. Run times were long, but not unreasonable. These results suggest that genetic algorithms may soon be practical for pattern classification problems as faster serial and parallel computers are developed.

Thesis Supervisor: Dr. Richard P. Lippmann
Title: Staff at Lincoln Laboratory

Thesis Supervisor: Dr. David W. Tong
Title: Electrical Engineer, GE Corporate Research and Development

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Feature selection and feature creation are two of the most important and difficult tasks in the field of pattern classification. Good features improve the performance of both conventional and neural network pattern classifiers. There is no good theory guiding the creation of good features, and only some theory in selecting good features. This thesis explores the application of genetic algorithms to both problems.

1.1 PATTERN CLASSIFICATION

The goal of pattern classification is to classify a set of patterns into different classes based on distinguishing characteristics. A pattern may be the outline of a fish, a bark from a dog, or the flag of a nation. Patterns from different classes are made up of features which differentiate the different classes. A feature can be any distinctive characteristic. For example, the U. S. flag has the following features: it has fifty stars, it has 13 alternating strips, and its colors are red, white, and blue.

In designing a pattern classification system, examples of the patterns in each class are typically used to "train" the pattern classifier. These patterns are called the training patterns. Features in patterns can be viewed as defining points in an input space. Providing more training patterns usually results in a better description of decision regions in the input space, resulting in a more accurate classifier. Decision regions are partitions of the input space into regions where patterns are classified as

one particular class. For example, the input space illustrated in the left of Figure 1.2 has two decision regions. Input patterns which fall into the white region are classified as □ class, while input patterns which fall into the shaded region are classified as the + class. After a classifier is trained, it must be tested with a different set of patterns called the testing patterns. When a classifier performs well on the testing set, generalization is high. Boundaries between different classes learned from the training patterns are accurate enough so that even a new set of patterns, the testing patterns, can be accurately classified.

It is important to have separate sets of training and testing patterns in order to confidently estimate the accuracy of a classifier. The performance of a classifier on training patterns can be overly optimistic since some classifiers, such as nearest neighbor classifiers, store all training patterns and can classify them perfectly. Accurate estimation of classification performance in real situations requires testing on patterns not used during training.

If the total number of patterns is too small to separate the patterns into training and testing sets that adequately characterize the classes, cross validation can be used[5]. In cross validation, a portion of the available patterns is randomly chosen to be used for training, with the remaining patterns used for testing. By averaging the classifier accuracy over different random partitions of training and testing data, a better estimation of the classifier's accuracy can be obtained.

With enough training patterns, similar low error rates can be provided using almost any type of neural net or conventional classifier[10]. However, the number of patterns available is often limited by the cost or the difficulty of obtaining more data. It is thus important to select and create good features that provide good performance with a limited number of training pattern.

Table 1.1: Possible Features of a Set of Shapes.

class-id	number of sides	area	x-proj	y-proj
T-1	3	0.78	1.56	1
R-1	4	0.54	1	0.54
R-2	4	0.18	0.18	1
T-2	3	0.32	1	0.64
T-3	3	0.97	1	1.94

1.2 WHAT'S A GOOD FEATURE?

Deriving a good set of features using genetic algorithms is one major goal of this study. An understanding of the relationship between features and classifier accuracy is thus essential. A good feature should make the task of distinguishing between different classes easier without requiring more training patterns.

A feature's usefulness depends on the classification task. For example, suppose the triangles and rectangles shown in Figure 1.1 need to be distinguished . A good feature is the number of sides in the object. Other possible features such as dimension and enclosed area provide no additional information for this task. The number of sides alone is the best feature, all other features are unnecessary and can be considered noise. Conversely, to separate out the objects by their size, the enclosed area of each object is the important feature.

A feature is good if it separates classes accurately with as few examples as possible. Going back to the object separation example, Table 1.1 lists some possible features of the objects.

If the number of sides is used as one feature, classes R and T (Rectangle and Triangle) can be specified with only two examples, one from class R and one from

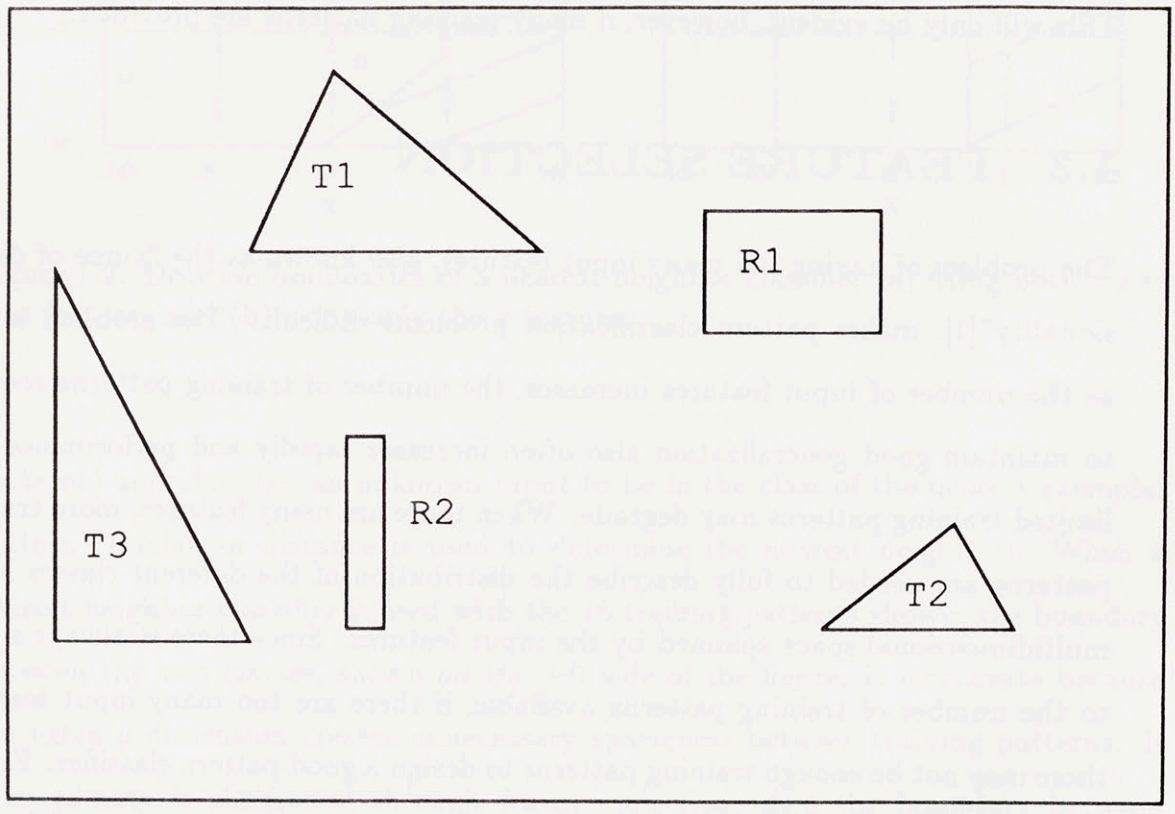

Figure 1.1: A set of rectangles and triangles that may be inputs to a pattern classifier.

class T. The number of sides is thus a very efficient feature to use for classifying objects into rectangles and triangles. On the other hand, if all features are considered, the area of each each object is also a reasonable feature, since for this given set, all objects with area greater than 0.6 are triangles and most objects with area below 0.6 are rectangles. However, since area is not fundamentally related to the shape of each object, the area feature would not provide reliable classification with new objects. This will only be evident, however, if many training patterns are provided.

1.3 FEATURE SELECTION

The problem of having too many input features, also known as the "curse of dimensionality"[1], makes pattern classification problems difficult. The problem is that as the number of input features increases, the number of training patterns required to maintain good generalization also often increases rapidly and performance with limited training patterns may degrade. When there are many features, more training patterns are needed to fully describe the distribution of the different classes in the multidimensional space spanned by the input features. Since there is always a limit to the number of training patterns available, if there are too many input features, there may not be enough training patterns to design a good pattern classifier. Feature selection (dimensionality reduction) is often required to select the subset of features that best separates classes.

Figure 1.2 demonstrates the effect of feature selection when training data is limited. In this problem, the first class consists of all points with x coordinate value greater than 3.5. The y coordinate value is random and is not important for classification. A nearest neighbor classifier was used to demonstrate the concept of decision regions. This classifier stores all reference training patterns (called exemplar

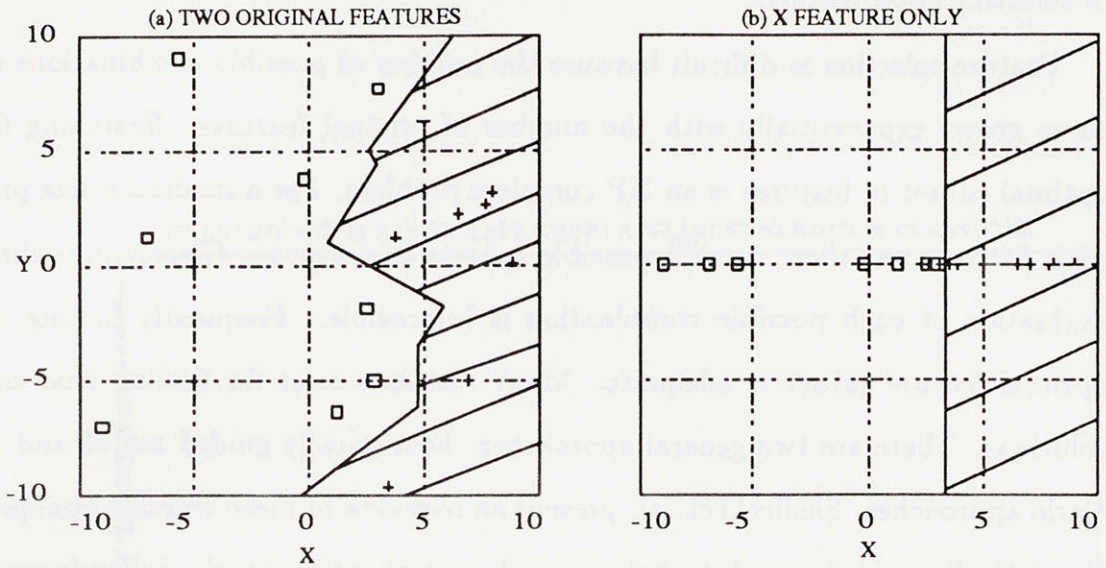

Figure 1.2: Decision boundaries of a nearest neighbor classifier (a) using both x and y as features and (b) using only the x feature.

patterns) and classifies an unknown input to be in the class of the nearest exemplar pattern (Euclidean distance is used to determine the nearest neighbor). When a nearest neighbor classifier is used with the 16 training patterns shown, the boundary between the two classes, shown on the left side of the figure, is inaccurate because the extra y dimension creates unnecessary sparseness between training patterns. If y coordinate is eliminated through feature selection, then the boundary between two classes becomes much more accurate as shown in the right side of Figure 1.2. Figure 1.3 plots the error rate of a nearest neighbor classifier for this problem as the number of training patterns varies from 0 to 500. Each curve is the average of 10 different random trials. Error rates are much lower when only the x feature is used, and an error rate of under 1% was achieved with less than 50 training patterns. On the other hand, using both features, the error rate is still above 1% even after the number of training patterns is increased to 500. This clearly demonstrates the benefit

of selecting good features.

Feature selection is difficult because the number of possible combinations of features grows exponentially with the number of original features. Searching for the optimal subset of features is an NP complete problem. For a moderate size problem with 64 features, there are 2^{64} possible subsets of features. Clearly an exhaustive evaluation of each possible combination is impossible. Frequently finding a near optimal feature subset is adequate. Many methods exist for finding near optimal solutions. There are two general approaches: heuristically guided search and Monte Carlo approaches. Siedlecki et. al. present an overview of these search techniques[17]. Heuristically guided search techniques such as sequential search, dynamic programming, and branch-and-bound search utilize heuristics to determine which solutions are to be examined. Monte Carlo approaches such as simulated annealing and genetic algorithms rely on selectively added randomness in the search to efficiently search for near optimal solutions.

1.3.1 Traditional Heuristically Guided Search Approaches

Sequential forward search and sequential backward search are the simplest and most widely used of the heuristically guided search techniques[17]. Sequential forward search starts with an empty feature subset, examines each feature's classifier accuracy individually, and puts the best performing feature into the current feature subset. Sequential forward search then looks at all the combinations that include the current feature subset and one of the remaining features and picks the best combination as the new current feature subset. At each cycle the number of features in the feature subset increases by one while the number of feature pairs examined reduces by one. The process repeats until the feature set grows to the original feature set size or a

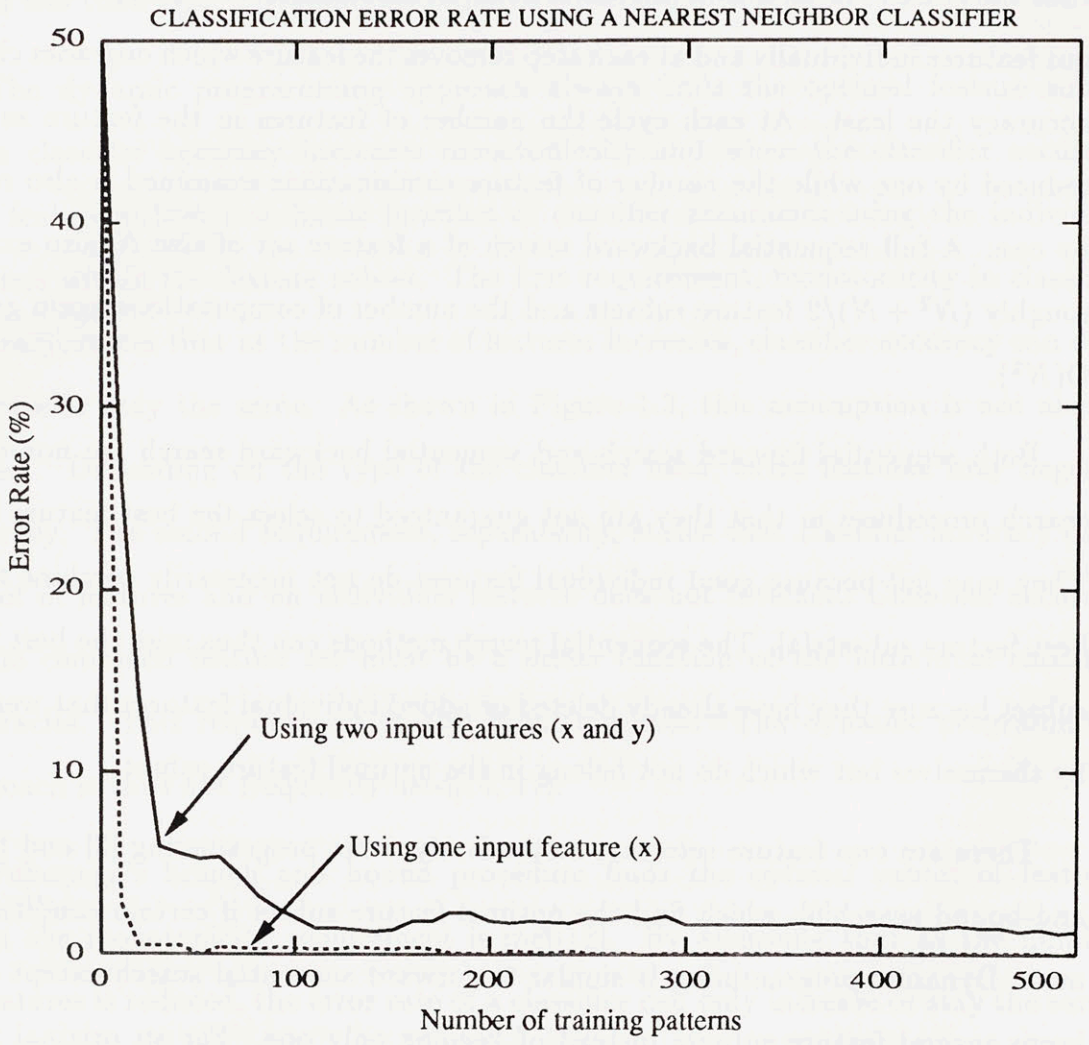

Figure 1.3: Classification error rate versus the number of training patterns used to train a nearest neighbor classifier for the feature set of (x, y) and the feature set of only x.

preset size. A full sequential forward search of a feature set of size N examines roughly $(N^2 + N)/2$ feature subsets and the number of computations grows as $O(N^2)$.

The backward sequential search is similar to the forward sequential search except that the search starts with a full feature set. Backward sequential search tries taking out features individually and at each step removes the feature which degrades classifier accuracy the least. At each cycle the number of features in the feature subset is reduced by one while the number of feature combinations examined is also reduced by one. A full sequential backward search of a feature set of size N also examines roughly $(N^2 + N)/2$ feature subsets and the number of computations again grows as $O(N^2)$.

Both sequential forward search and sequential backward search are non-optimal search procedures in that they are not guaranteed to select the best feature subset. They may fail because good individual features do not necessarily combine to form best feature subsets[4]. The sequential search methods can thus miss the best feature subset because they have already deleted or added individual features that were good by themselves but which do not belong in the optimal feature subset.

There are two feature selection methods, dynamic programming [3] and branch-and-bound search[6], which find the optimal feature subset if certain conditions are met. Dynamic programming is similar to forward sequential search except that it keeps several feature subsets instead of keeping only one. For an original feature set of size N, the dynamic programming approach starts with N feature subsets, each containing one original feature. Individual features are then combined with all feature subsets that do not contain the feature and assigned to the feature subset which performs the best with the individual feature. The process is repeated until all subsets have grown to the desired feature set size. For a feature set of size N,

the dynamic programming approach examines $N^2 * (N-1)/2$ feature subsets. The number of subsets examined grows as $O(N^3)$. The dynamic programming approach uses more than the $O(N^2)$ evaluations required by sequential search methods and much less than $O(2^N)$ evaluations required by exhaustive search.

The dynamic programming approach always finds the optimal feature subset when classifier accuracy increases monotonically and when the classifier accuracy of a feature subset is a linear function of classifier accuracies using the individual features within the feature subset. The first requirement, monotonicity in classifier accuracy, states that as the number of features increases, classifier accuracy can only increase or stay the same. As shown in Figure 1.3, this assumption is not always correct. Depending on the type of the classifier used, extra features may degrade accuracy. The second requirement, separability, states that classifier accuracy on a subset of features and on individual features does not interact. Classifier accuracy of the combined feature set must be a linear function of the individual classifier accuracies. This requirement is also often not met. The dynamic programming approach is thus not frequently used[14, 17].

Fukunaga's branch and bound procedure finds the optimal subset of features when the monotonicity requirement is met[12]. By assuming that as the number of features is reduced, the error rate of a classifier can only increase or stay the same, branches that have very high error rates in a search tree can be disregarded because reducing more features in that subset will not reduce the error rate. The algorithm can thus concentrate on only promising branches and find the optimal feature subset given sufficient time. In many real life problems, however, reducing the number of features may actually reduce the classifier's error rate, thus, these problems are not monotonic[4, 7]. This restriction and the complexity of the branch and bound

procedure with high input dimensionality again have limited its application.

1.3.2 Genetic Search

Genetic algorithms have recently been applied with good results to NP complete problems such as the traveling salesman problem and job scheduling problems[2]. They take advantage of "implicit parallelism" to efficiently search for good solutions[8] and depend on the generation-by-generation development of possible solutions, with selection eliminating bad solutions in future generations and allowing good solutions to be replicated and modified. It has also been shown that genetic algorithms are effective in optimizing multi-modal and noisy functions[8]. In these applications, each solution manipulated by genetic algorithms represents one possible location of the maxima of a complex function. Solutions specified by bit strings are first randomly generated, then evaluated, and finally manipulated to produce new strings. A suitable function needs to be found for evaluating the fitness of each solution. The selection and the search for a better solution is directly affected by the fitness function, thus the fitness function should be linked tightly to the eventual goal. In pattern classification problems, the usual criterion for success is the percentage of patterns classified correctly. It is thus logical to use the actual classification accuracy as the fitness function of a given subset of features instead of other possible functions, such as the variance of data with respect to the subset of features.

The "training on testing data" problem[5] may appear when the percentage of training patterns classified correctly is used as the fitness function. If the classifier accuracy on the training set is used as the fitness evaluation function, then as better subsets are created at each generation, the feature subsets will gradually be selected according to how they perform according to the testing patterns; in essence, testing

patterns have been used as training patterns. There is a danger of finding selections that are good for a particular set of testing patterns yet bad for the general distribution of patterns. This testing on training data problem can be delayed through using the "leaving one out" method and cross validation. A different portion of the training patterns is tested at each generation and the feature subsets' performance on this portion of patterns is used for fitness evaluation. However, a separate set of "pure" testing patterns is used to test the performance of feature subsets once feature selection is over. The feature subsets' performance on the pure test patterns is used to verify that feature subsets provide good generalization.

Siedlecki has recently successfully applied genetic algorithms to select features for high dimensional problems[19]. However, most of the studies he performed used artificially generated data. The only problem with real data consisted of 150 patterns with input dimension of 30. Because the number of training patterns was small, apparently the training set consisted of all 150 patterns and no testing set result was reported. As mentioned previously, without checking for the generalization ability of a given set of features, the result obtained on the training set by genetic algorithms may be highly misleading since the genetic feature selection method can overfit the training data.

This thesis compares the genetic search approach with the forward and backward sequential search approaches in efficiency and success of selecting features for several real problems. The sequential approaches were chosen as a basis of comparison because they are the most efficient of traditional approaches and they frequently perform well. Determining the practicality of genetic feature selection is the focus of this research.

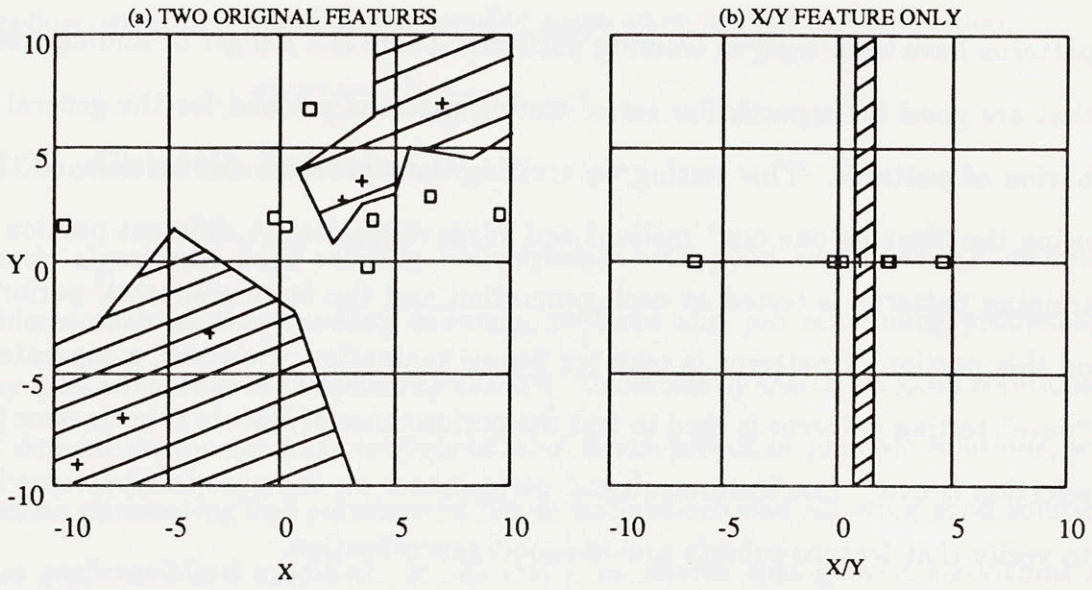

Figure 1.4: Decision boundaries of a nearest neighbor classifier (a) using both x and y features and (b) using the (x/y) feature to classify points on a diagonal line.

1.4 FEATURE CREATION

Feature creation is one of the most important and difficult parts of pattern classification. Created features are normally highly specific to the problem domain. For example, in speech recognition, the distribution of spectral energy may be useful features. In machine vision problems, corners or edges may be useful features. Finding the right features may demand extensive experimentation, yet without good features, it is impossible to provide high classification accuracy. Sometimes higher order functions of original features can dramatically reduce the number of training patterns needed and improve classification accuracy. One example is a pattern classification problem where all points having equal x and y coordinates form one class and all other points form another class, as illustrated in the left side of Figure 1.4.

The decision boundaries shown in the figure were formed using a nearest neighbor classifier as in Figure 1.2. Using the original features, shown in the left side of

Figure 1.4, provides good generalization only if there are many training patterns which cluster together on the diagonal line. However, since there are only a limited number of training patterns, there will very likely be gaps between points on the diagonal line so that a testing pattern in that gap will be classified incorrectly. Recognizing that higher order functions of the x and y features are more informative would provide better use of the limited training patterns. In this problem, the ratio between the x and y coordinates distinguishes between the classes. Suppose a new system, shown in the right side of Figure 1.4, is created where only the ratio of the x and y coordinates is used as the feature. In this case, many fewer training patterns are needed to provide good classification accuracy. All patterns which have a ratio of one belong in the diagonal class and are bounded within the narrow shadowed zone, while all other patterns belong in the non-diagonal class. In this case, it is impossible for a diagonal class pattern to be misclassified using a nearest neighbor classifier because all diagonal class patterns are on the same point. The chance of non-diagonal patterns being misclassified is greatly reduced as well.

Figure 1.5 plots the error rate of a nearest neighbor classifier versus the number of training patterns for this problem. It is clear that by using a good feature (the ratio of x/y) less than 20 training patterns are required to have an error rate of roughly 2%. On the other hand, using just the x and y features, even with 500 training patterns, the error rate of the classifier is still well above 2%.

Many feature creation techniques are available to create new features that are linear combinations of a given set of features. Fisher's linear discriminant approach creates new features in the direction of greatest intra-class variance. Features are thus generated with the hope that they will more clearly separate the classes[5]. Using a Karhunen-Lôeve expansion, the original features are transformed into a new set of

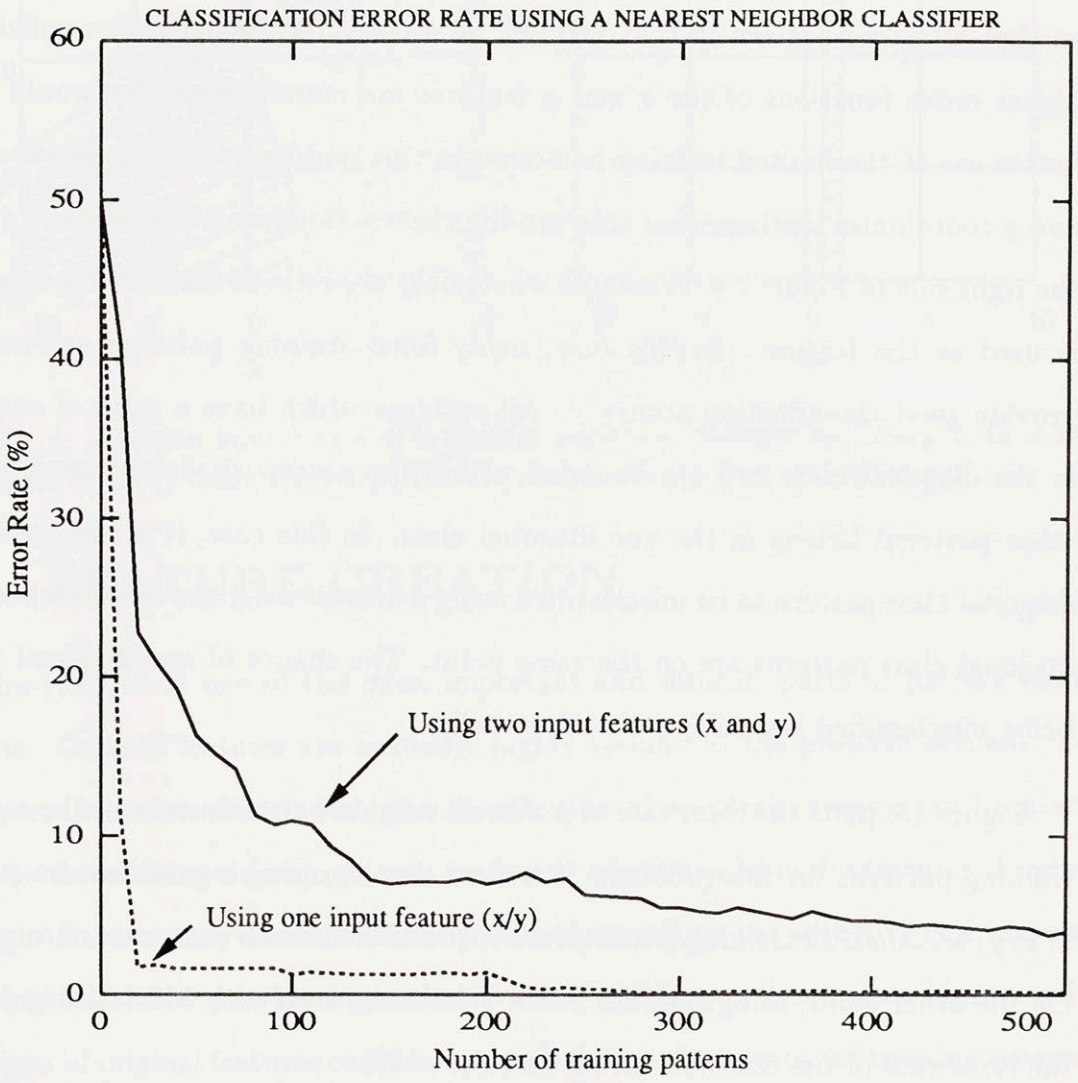

Figure 1.5: Classification error rate versus the number of training patterns used to train a nearest neighbor classifier for the diagonal line problem.

features with the aim of reducing the correlation between the new features as much as possible. By removing the correlation between features, it is hoped that the new features with the greatest variance will be the truly useful features. These approaches can reduce the number of input features to a classifier if only the most useful features are kept. They work well on problems where the patterns are linearly separable in the new feature space. In real problems, it is difficult to know *a priori* whether a set of patterns are linearly separable in some new feature space. Also, both of these methods examine the statistical properties of the training patterns which may not relate directly to the classifier accuracy.

Much research still needs to be done on creating higher order nonlinear features. For example, given a feature set of x and y, one can readily create a feature $z = ax+by$ with the coefficients a and b calculated using the methods described above. Few methods have been found, however, which effectively create higher order nonlinear features such as $z = x/y$ or $x * y$. The number of possible higher order features grows exponentially with the number of original features. For example, given a problem with three original features x_1, x_2, and x_3, the number of unique combinations of these three variables taken two at a time is 3. If the number of variables is increased to six, the number of unique combinations increases exponentially to 15. The exponentially growing characteristic of the problem makes an exhaustive search for even a moderate size problem computationally infeasible. Only searches of near optimal solutions can be attempted. Tenorio et. al. have applied simulated annealing and a tree-based approach to search for the near optimal solutions[20]. Other methods such as the Group Method of Data Handling (GMDH)[11] and non-linear regression can also be used to find higher order features. These methods rely on local pruning to reduce the number of combinations to be searched. For example, the GMDH approach builds

complex functions out of the original features by keeping function subblocks which fit the desired output function closely. It can encounter local minima and build very complicated functions when a simpler higher order function would be better because the local pruning falsely eliminates necessary function subblocks.

This thesis explores the use of genetic algorithms to search the space of possible features and find good new high order features that are nonlinear combinations of original input features. Such higher order features will improve the efficiency of pattern classification if the correct problem-specific feature can be found.

Higher order terms can take many forms, and the multiplicative form chosen here is only a small portion of all possible higher order terms. For example, the multiplicative forms can represent new features such as $x^2 * y$; however, other possible new features, such as $x * ln(y)$ and $(x + y)/(x - y)$, can not be found using only a polynomial representation. The type of higher order terms that provide best performance will be problem dependent. For example, in time series problem, the Fourier transform of inputs or the cepstrum may be the correct feature to use. In general, any nonlinear function can be chosen by the user to be created and searched using genetic algorithms.

1.5 THESIS OUTLINE

The remainder of this thesis is organized as follows: Chapter 2 describes the theory and practice of genetic algorithms. Experimental results of applying genetic algorithms on two simple problems are also presented. Chapter 3 describes the k nearest neighbor classifier and some improvements made to improve efficiency. Chapter 4 focuses on feature selection. Experimental results on real and artificial problems are presented. Chapter 5 describes methods and results of applying genetic algorithms

to feature creation. Chapter 6 describes enhancements that increase the complexity of features created. Chapter 7 presents results of experiments which used genetic algorithms to reduce the number of exemplars needed by k nearest neighbor classifiers. Finally, Chapter 8 provides an overview and discusses future research directions.

Chapter 2

GENETIC ALGORITHMS

2.1 INTRODUCTION

Genetic algorithms were first proposed by John Holland in 1967 to optimize functions. Since then, they have been applied to many different types of problems, such as pipeline control, computer aided design, and classifier design[8]. Genetic algorithms emulate Darwin's theory of evolution. A group of possible solutions are judged according to a "fitness" function, which is explicitly related to the objective function to be maximized. Better solutions are chosen and random elements are exchanged between two chosen solutions to generate new possible solutions. The new solutions then undergo mutation, where the bits of the solutions are randomly altered. Afterwards, these new solutions replace members of the old population. If the combination of two partially good solutions yields better solutions, then genetic algorithms will efficiently find near-optimal solutions.

2.1.1 Four Stages in Genetic Algorithms

There are four stages in the genetic search process: creation, selection, crossover, and mutation. In the creation stage, a group of possible solutions is randomly generated. In most genetic algorithm applications, each solution is a string of 0's and 1's. Each string is created by randomly placing 0's and 1's in the string.

After the creation stage, each solution is evaluated and assigned a fitness value.

This is followed by a selection stage, where the fitter solutions are given more chance to reproduce. This stage gives the fitter solutions more and more influence over the changes in the population so that eventually fitter solutions dominate.

A crossover operation occurs after two fitter solutions (called parent solutions) have been chosen to reproduce. During crossover, portions of the parent solutions are exchanged. This operation is performed in the hope of generating new solutions which will contain the useful parts of both parent solutions and be even better solutions. Crossover is responsible for generating most of the new solutions in genetic search.

When all the solutions are similar, the crossover operation loses the ability to generate new solutions since exchanging portions of same solutions generates the same solutions. Mutation is performed on each new solution to prevent the whole population from becoming similar. However, mutation does not generally improve solutions by itself. The combination of both crossover and mutation is required for good performance.

2.1.2 A Simple Example

A simple feature selection problem can be used to demonstrate the basic concepts of genetic algorithms. A string of 0's and 1's is used to indicate whether a given feature is used. Assume that for a given subset of features, fitness is the percentage correct score of a nearest neighbor classifier using this particular subset of features. The goal is to find the optimal feature subset with as few tries as possible. Suppose the ideal feature subset and its fitness value are:

10000 90%

To find a good feature subset, four procedures of genetic algorithms are performed, creation, selection, crossover, and mutation. In the creation stage, a finite set of

possible solutions is randomly generated. For example, in a particular experiment the following initial population may be created:

10110	74%	Initial set of solutions
01101	80%	
01011	80%	
01010	78%	
00011	78%	
11110	82%	

After this initial set has been generated and evaluated as shown above, two solutions are selected to create new strings. This is the process of selection. Just as in nature where the fittest tends to survive, fitter solutions are more likely to be selected. However, weak solutions still have a chance to become parents. Suppose the following two solutions are selected:

$\overline{01101}$	80%	Two selected solutions
$\underline{01011}$	80%	

Portions of these solutions are exchanged in order to create new solutions. By performing the crossover operation, new solutions can be created which retain the traits of the parents. A simple type of crossover operation is picking a random point along the solution string and exchanging the solutions' second portion. An example is shown below:

$\overline{011}\underline{11}$	78%	The new solutions after crossover
$\underline{010}\overline{01}$	82%	

The two selected solutions are crossed over to create two new solutions. The lines above and below the two strings are used to distinguish the string portion of one selected solution from that of the other.

After crossover, the newly created solutions are randomly mutated. This is necessary because sometimes all the solutions in the population are very similar. Even exchanging portions of different solutions creates no new solutions. With a small amount of mutation, however, new solutions are created to introduce more diversity into the population.

In our example, suppose the first new string is randomly changed at one position identified by the $\sim$ sign:

$$01\tilde{0}11 \quad 80\% \Big\} \text{Mutated solution}$$

In this case the mutation was fortuitous, because it improved the modified solution's fitness value from 78% to 80%. This does not always happen and is not the main purpose of performing mutation. Mutation's chief contribution is preventing premature convergence of the whole population of solutions, i.e, the whole population having similar solutions. Without mutation, once premature convergence occurs, genetic search stops. This process of selection, crossover, and mutation would normally be continued through many generations in this problem until an acceptable solution was found. As can be seen by this example, many new strings may need to be created because there is no guarantee that crossover and mutation will always lead to fitter strings.

2.2 METHODS

2.2.1 Introduction

There are many varieties of genetic algorithms. In the original simple models, all members of a population completely reproduce at every generation. In more complicated models, "niches" are formed in the population and "migration" is allowed within niches[8]. This study used an incremental approach that has been used successfully in other areas and was found to work well in initial exploratory experiments.

2.2.2 Static Population Model

The relatively new incremental static population model proposed by Darrel Whitley [22] was used in all experiments. In the regular genetic algorithm model, the whole population undergoes selection and reproduction, with a large portion of the strings replaced by new strings. It is thus possible for good strings to be deleted from the population. In the static population model, the population is ranked according to fitness. At each recombination cycle, two strings are picked as parents according to their fitness values, and two new strings are produced. These two new strings replace the lowest ranked strings in the original population. This model automatically protects the better strings in the population, so that the best string found so far always stays in the population. Also, since changes in the population are incremental, a large portion of the population is never replaced by worse strings (this can conceivably happen in the regular generation model).

2.2.3 Ranked Based Selection

Genetic algorithms rely on the fact that better strings are more likely to reproduce to increase search efficiency. In the experiments performed, a ranked based selection approach was used to select parents. In the typical genetic algorithm model, the probability of a string i becoming the next parent, $p(i)$, is calculated as follows: $p(i) \sim c * f(i)/f(avg)$. Here $f(i)$ is the fitness of the string i, $f(avg)$ is the average fitness of the whole population, and c is a constant selected by the user. If the fitness of string i is ten times greater than the average fitness, it would be ten times more likely than a average string to reproduce.

This selection scheme can fail due to the large differences between fitness values. For example, if the average fitness value is 10, yet two strings have the fitness values of 90, clearly these strings will be selected most of the time. When both of the parents are the same, the crossover operator has no effect and gradually the whole population becomes duplicates of the fittest strings. No new strings are created and the search halts. This phenomenon defeats the purpose of genetic algorithms search and is similar to the phenomenon of "inbreeding" noted by biologists.

To prevent superbly fit individuals from dominating a population, various scaling schemes have been proposed. For example, the value of c can be periodically adjusted when the probability of reproduction is assigned. In the beginning of the search, when a few strings are more likely to have a high fitness value relatively to the average fitness value, the constant c is adjusted lower. At the latter part of the search, when the difference between the best string's fitness value and the average string's fitness value is smaller, the constant c is increased to ensure that the best strings do get more chances to reproduce.

There are many problems in using the fitness values themselves to directly compute

the probability of reproduction. By using the rank-based selection scheme, all these problems can be avoided[22]. In a rank-based system, all the strings within the population are ranked according to their fitness value. A variable called *selective pressure* is used to determine how much the top strings are favored. The probability to reproduce is computed with the following iterative equations:

$$p(1) = 1.0 * \textit{selective pressure},$$

$$p(i+1) = (1.0 - \sum_{j=1}^{i} p(j)) * \textit{selective pressure},$$

where string 1 is the most fit solution and probabilities are assigned to the fittest solution first. This scheme gives exponentially more reproduction opportunity to fitter strings. Also, since only the relative rank of the strings affects the probability of reproduction, changes in the distribution of fitness values have no effect as genetic search progresses. An additional benefit is that bias toward good strings can be controlled by varying the *selective pressure* variable. For example, Figure 2.1 shows that the distribution of the chance to reproduce changes visibly when the *selective pressure* is changed from 0.05 to 0.25.

2.2.4 Crossover Operators

There are many different techniques that can be used to create new strings by crossing over old strings. A uniform crossover operator, one point crossover operator, two points crossover operator, or unit based operator can be used. With the uniform operator, random bits of each string are independently chosen to be crossed over.

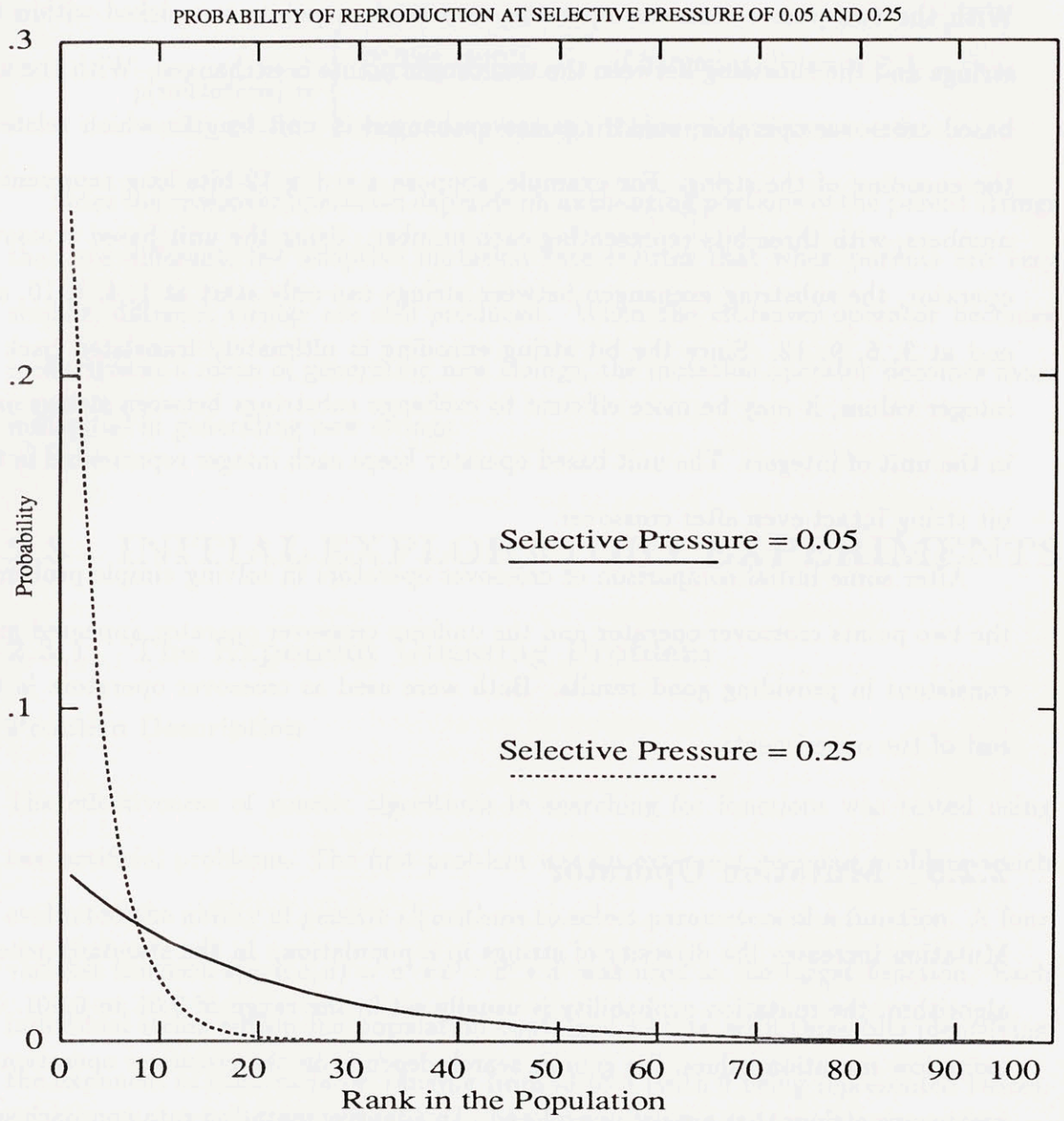

Figure 2.1: A comparison of the distribution of probability of reproduction with *selective pressure* values of 0.05 and 0.25 and population size of 100.

With the one point crossover operator, a random point is chosen in the string and the substring starting from the chosen point until the end of the string is exchanged. With the two points crossover operator, two random points are picked within the strings and the substring between the two chosen points is exchanged. With the unit based crossover operator, substrings are exchanged in unit lengths which relate to the encoding of the string. For example, suppose a string 12 bits long represents 4 numbers, with three bits representing each number. Using the unit based crossover operator, the substring exchanged between strings can only start at 1, 4, 7, 10, and end at 3, 6, 9, 12. Since the bit string encoding is ultimately translated back to integer values, it may be more efficient to exchange substrings between strings only in the unit of integers. The unit based operator keeps each integer represented in the bit string intact even after crossover.

After some initial comparison of crossover operators in solving simple problems, the two points crossover operator and the uniform crossover operator appeared most consistent in providing good results. Both were used as crossover operators in the rest of the experiments.

2.2.5 Mutation Operator

Mutation increases the diversity of strings in a population. In the standard genetic algorithm, the mutation probability is usually set in the range of 0.01 to 0.001. By using low mutation values, the genetic search depends on the crossover operation to create new strings that are yet unexplored. An adaptive mutation rate approach suggested by Whitley[23] was used in all experiments. This approach uses the hamming distance between the two parents as a measure of their similarity. If the hamming distance is large, then the mutation rate is reduced. If the hamming distance is small,

then the mutation rate is increased. More specifically, the mutation rate is calculated with the following equations:

$$p(mutation) = \begin{cases} \frac{mutative\ pressure}{hamming\ distance} & \text{if hamming distance} \geq 1 \\ mutative\ pressure & \text{if hamming distance} = 0 \end{cases}$$

Since the crossover operation depends on exchanging portions of the parent strings that are different, the adaptive mutation rate ensures that when parents are very similar, different strings are still produced. When the crossover operator becomes ineffective as a mean of generating new strings, the mutation operator becomes more influential in generating new strings.

2.3 INITIAL EXPLORATORY EXPERIMENTS

2.3.1 The Exponent Guessing Problem

Problem Description

The effectiveness of genetic algorithms in searching for functions was tested using two artificial problems. The first problem was an exponent guessing problem which evaluated the ability of genetic algorithms to select parameters of a function. A four variable function $t(a,b,c,d) = a^i * b^j * c^k * d^l$ was used as the target function. Each individual string within the population contained 12 bits, with three bits identifying the exponent of each variable, ranging from -3 to 3 (with 0 being represented twice). The difference, δ, between the desired function $t()$ and the function guessed by genetic algorithm was summed over the integer range ($0 \leq a \leq 5, 0 \leq b \leq 5, 0 \leq c \leq 5, 0 \leq d \leq 5$). The ratio $1/(\sum \delta + 0.001)$ was used as the fitness function to evaluate the fitness of each individual string. The small value 0.001 was added to the overall sum

to avoid dividing by zero when the correct function is found.

This experiment was performed to see whether genetic algorithms were able to find the correct function which maximizes fitness. When creating features, as described in Chapter 5, different high order functions of basic features are searched, it is thus important to test genetic algorithms' ability to create functions. There had been no previous results on using genetic algorithms to create functions, and it was uncertain whether genetic algorithms would work well on this problem.

Results

Sets of experiments using different crossover operators, mutative pressure, and selective pressure were run. The results are shown in Tables 2.1 to 2.4. Ten independent trials were run for each combination of selective pressure, mutative pressure, and crossover operator. The numbers shown in the tables are the average number of recombinations over 10 trials required before the perfect answer was found. Each trial was stopped once the number of recombinations reached 1001. The original population size was 100; therefore, 100 evaluations for the original population plus the number of recombinations equals the total number of evaluations used before the correct function was found.

The bit strings in this problem were 12 bits long. One of the exponents to be guessed was zero, which could be represented with two distinct combinations (000 and 100), so the total number of distinct strings was 2^{11}, or 2,048. An average of 1,024 evaluations would thus be needed in a random search procedure. Tables 2.1 to 2.4 show that as long as a selective pressure of 0.0 or a mutative pressure of 0.0 was not used, genetic algorithms always required fewer evaluations than the random search procedure. Furthermore, except when mutative pressure or selective pressure

Table 2.1: Number of Recombinations until the Correct Solution Was Found with a Uniform Crossover Operator for the Exponent Problem.

Mutative Pressure / Selective Pressure	0.0	0.25	0.5	0.75	1.0
0.00	387	247	452	515	862
0.05	387	109	265	320	244
0.10	713	70	219	423	449
0.15	705	282	137	300	169
0.20	804	143	293	374	305
0.25	902	212	172	225	373

Table 2.2: Number of Recombinations until the Correct Solution Was Found with a Two Points Crossover Operator for the Exponent Problem.

Mutative Pressure / Selective Pressure	0.0	0.25	0.5	0.75	1.0
0.00	447	432	648	664	442
0.05	902	374	252	281	374
0.10	809	159	334	293	322
0.15	901	160	134	397	446
0.20	901	216	348	334	310
0.25	1001	246	589	389	285

Table 2.3: Number of Recombinations until the Correct Solution Was Found with a One Point Crossover Operator for the Exponent Problem.

Mutative Pressure / Selective Pressure	0.0	0.25	0.5	0.75	1.0
0.00	406	451	373	507	453
0.05	708	266	327	374	385
0.10	906	174	324	250	180
0.15	710	297	177	280	342
0.20	1001	106	248	540	377
0.25	901	378	179	358	474

Table 2.4: Number of Recombinations until the Correct Solution Was Found with a Unit Based Crossover Operator for the Exponent Problem.

Mutative Pressure / Selective Pressure	0.0	0.25	0.5	0.75	1.0
0.00	504	459	436	509	603
0.05	721	312	125	319	371
0.10	704	165	316	342	373
0.15	1001	226	195	244	462
0.20	902	236	297	412	333
0.25	701	278	266	358	343

Table 2.5: The Average and Median Number of Recombinations until the Correct Solution Was Found Using Different Operators for the Exponent Problem.

Operator	Average	Median
Uniform	236.3	219
Two Points	300.4	281
One Point	285.2	266
Unit Based	272.7	266

equaled 0.0, there was no general trend in the number of recombinations required for different selective and mutative pressure values. Genetic algorithm search appeared to be robust to the values chosen and worked better than a random search for a wide range of values.

Table 2.5 lists the average and the median number of recombinations for a mutative pressure range of 0.25 to 0.75 and a selective pressure range of 0.05 to 0.25 required by each crossover operator. The uniform operator required fewer recombinations than the two points crossover operator, however, the difference was small. No crossover operator performed an order of magnitude better than other operators. All operators required more than a factor of three less evaluations than would be required by an average random search.

Another set of experiments was performed using the traditional generation model on the same exponent guessing problem. Three sets of experiments, each consisting of ten independent trials, were performed. The population size was also 100, the probability of reproduction was 60%, and the mutation rates were 0.001, 0.01, and 0.10 respectively. In any given generation, 60% of the strings were likely to be reproduced. The expected number of recombinations per generation was thus 60. The average number of recombinations required for the three sets of experiments is

Table 2.6: Number of Recombinations until the Correct Solution Was Found with a Two Points Crossover Operator for the Exponent Problem Using a Traditional Approach.

Mutative Pressure	Avg. # Recomb.	St. Dev.
0.001	540	180
0.01	1020	300
0.10	840	180

listed in Table 2.6.

The generation approach was not as efficient as the static population model approach. For example, the average number of recombinations with the two points crossover operator and the static population model was 300, yet all three trials of the generational model required more than 500 recombinations. This difference in efficiency is due to the fact that many new strings are generated in each generation, so a good string can reproduce many times in one generation. This excessive reproduction results in homogeneity in the population and premature convergence. In this experiment, the best string was always kept in the population, further increasing the chance of premature convergence.

2.3.2 The Linear Combination Guessing Problem

Problem Description

This problem tested the ability of genetic algorithms to find a combination of functions simultaneously. In feature creation, a set of features may need to be found at the same time in order for the classifier to benefit from them. To test the effectiveness of genetic algorithms in guessing a set of functions, the second problem was designed to

be a search of a linear combination of functions.

In this problem, genetic algorithms were used to find the variables and exponents of the function $t() = x_1^{i1} * y_1^{j1} + x_2^{i2} * y_2^{j2} + x_3^{i3} * y_3^{j3}$. The x and y variables could be any of the four variables a, b, c, and d. The exponent of each variable was either 1 or -1. Two bits were used to identify the variable and one bit was used to indicate the exponent of the variable. Each of the terms in the equation thus required 6 bits to represent. The whole equation required a total of 18 bits.

The actual function to be guessed was $t() = a * b + a * c + b/d$. Again the fitness function used was $1/(\sum \delta + 0.001)$, the inverse of the difference δ between the function guessed by genetic algorithms and the actual function over the range of 0 to 5 for each variable. To prevent dividing by zero when the perfect solution was found, a very small value (0.001) was again added to the sum of differences. The fitness value of the perfect string is thus 1,000.

Results

Experiments were again performed using different selective pressure, mutative pressure, and crossover operators. The average numbers of recombinations required before the correct solution was found are listed in Tables 2.7 to 2.11. The total number of evaluations used was the initial population size (100) plus the number of recombinations. This linear combination problem had an encoding length of 18 bits. However, there are some redundant solutions, for example, the term $a * b$ is the same as the term $b * a$. The actual number of distinct solution is $(2^5)^3$, or 32,768. Out of the 32,768 solutions, there are 3! correct solutions because the three terms in the target equation can be rearranged. The probability of successfully guessing the correct solution in a random trial is then 6/32768. The expected number of trials

required in a random search procedure is $1/2 * 6/32768 = 2730$.

Table 2.7 lists the number of recombinations required when the original two points crossover operator is used. As long as a mutative pressure of 0.0 or 1.0 and a selective pressure of 0.0 is not used, the average number of recombinations is 446. After adding 100, the number of evaluations required for the starting population, the total of 556 is still much less than 2,730. Genetic algorithms performed better than random search for all four types of crossover operators.

The operator tested in Table 2.7 first uniformly picked a point, then picked a second point uniformly between the first point and the end of the string. The portion between the first point and the second point was exchanged. This operator differed from an alternative form of two points crossover operator, where both points were picked uniformly along the length of the string, with the smaller number becoming the starting point.

The experiment with the second type of two points operator is shown in Table 2.8. There is no significant difference between Table 2.7 and Table 2.8. There was also no clear advantage in using the one point crossover operator (Table 2.9), the uniform operator (Table 2.10), or the unit based operator (Table 2.11). An operator might have been better at a particular setting of selective and mutative pressures, but no operator was consistently better than all the others.

Table 2.12 shows the average and the median number of recombinations for the mutative pressure range of 0.25 to 0.75 and selective pressure range of 0.05 to 0.25. These ranges were chosen to remove the results of using extreme mutative and selective pressure values. Within the ranges, the two points crossover operator performed the best and the uniform crossover operator performed the worst. The results differ with the results shown in Table 2.5, when the uniform operator performed the best

Table 2.7: Number of Recombinations until the Correct Solution Was Found with a Two Points Crossover Operator for the Linear Problem.

Mutative Pressure / Selective Pressure	0.0	0.25	0.5	0.75	1.0
0.00	1106	478	285	735	961
0.05	2413	449	317	336	599
0.10	2711	313	337	634	400
0.15	2704	355	544	493	945
0.20	2405	686	561	419	965
0.25	3001	309	558	382	866

Table 2.8: Number of Recombinations until the Correct Solution Was Found with a New Two Points Crossover Operator for the Linear Problem.

Mutative Pressure / Selective Pressure	0.0	0.25	0.5	0.75	1.0
0.00	680	698	562	804	948
0.05	1558	392	415	585	405
0.10	2403	614	506	511	705
0.15	2703	303	453	691	520
0.20	3001	369	572	640	510
0.25	2402	583	517	392	615

Table 2.9: Number of Recombinations until the Correct Solution Was Found with a One Point Crossover Operator for the Linear Problem.

Mutative Pressure / Selective Pressure	0.0	0.25	0.5	0.75	1.0
0.00	1088	564	671	522	1095
0.05	1855	282	306	808	867
0.10	2125	341	423	745	552
0.15	3001	480	258	582	523
0.20	2702	271	334	763	628
0.25	3001	496	797	650	668

Table 2.10: Number of Recombinations until the Correct Solution Was Found with a Uniform Crossover Operator for the Linear Problem.

Mutative Pressure / Selective Pressure	0.0	0.25	0.5	0.75	1.0
0.00	846	488	721	950	801
0.05	760	271	552	762	1044
0.10	2130	298	533	703	1480
0.15	2707	567	346	876	1068
0.20	2703	399	491	569	1031
0.25	2402	291	965	1142	1083

Table 2.11: Number of Recombinations until the Correct Solution Was Found with a Unit Based Crossover Operator for the Linear Problem.

Mutative Pressure / Selective Pressure	0.0	0.25	0.5	0.75	1.0
0.00	864	458	462	385	907
0.05	3001	416	364	505	528
0.10	2117	353	492	465	699
0.15	2406	244	453	438	765
0.20	2406	371	507	546	949
0.25	2403	328	815	990	902

Table 2.12: The Average and Median Number of Recombinations Until the Correct Solution Was Found Using Different Operators for the Linear Problem.

Operator	Average	Median
Uniform	584.3	533
Two Points	446.2	382
New Two Points	502.9	506
One Point	502.4	480
Unit Based	556.5	438

and the two points crossover operator performed the worst.

Table 2.13 shows the average and the median number of recombinations when the mutative pressure is 0.25 and the selective pressure range is 0.05 to 0.25. In this table, the new two points operator turned out to be the best in terms of average, while the uniform crossover operator was the best in terms of median.

Depending on the selective and mutative pressure range, one operator may be superior to another. However, no result convincingly shows one crossover operator to

Table 2.13: The Average and Median Number of Recombinations until the Correct Solution Was Found Required by Different Operators for the Linear Problem using Mutative Pressure of 0.25.

Operator	Average	Median
Uniform	365.2	298
Two Points	422.4	355
New Two Points	252.2	392
One Point	374.0	341
Unit Based	342.4	353

be better than other operators. In this thesis, the two points crossover operator was used for feature selection and feature creation problems for consistency. On problems with longer strings such as the experiments described in Chapter 7, the uniform crossover operator was used. These choices were made based on the experience that no one operator performed an order of magnitude better than the others. Performing an complete analysis of the performance of all crossover operators for the type of problems studied in this thesis would have taken too long, so no further studies on the effect of using different operators were pursued.

2.4 SUMMARY

This chapter describes the genetic algorithm approach selected for this study and the effect of using different mutative pressure, selective pressures, and crossover operators. Although small differences exist between the performance of different crossover operators, they are not large enough to favor selecting one operator over another. No specific set of selective and mutative pressure values was good for all

experiments but mutative pressures of 0.25 and 0.5 and selective pressures from 0.05 to 0.25 appeared to work well in all problems. The standard two points operator and the uniform operator were used for further experiments with parameters chosen from this range. With choices from these ranges, genetic algorithms on average found the correct function for two artificial function selection problems in 1/5 to 1/2 as many trials as would be required on average by a random search procedure.

Chapter 3

NEAREST NEIGHBOR PATTERN CLASSIFICATION

3.1 INTRODUCTION

Genetic algorithms for feature selection require a fitness function to estimate the usefulness of a set of features. The most direct way of determining the usefulness of a set of features is to actually use the features with a pattern classifier. In this thesis, the percentage correct from a k nearest neighbor classifier was used as the evaluation function. This chapter briefly describes the k nearest neighbor classifier and the enhancements made to reduce computation requirements.

3.2 THE K NEAREST NEIGHBOR CLASSIFIER

3.2.1 History

The k nearest neighbor classifier has been used as a reference classifier by many researchers in the neural networks field[10]. It has the advantage of not requiring a training phase and providing good accuracy when there are sufficient training patterns which are representative of the overall pattern distribution. It has been proven that the error of the nearest neighbor classifier is bounded by twice of the Bayes error when the number of training patterns is large[5]. Previous work has also demonstrated that

the relative performance of a feature set found using a k nearest neighbor classifier is closely related to the relative performance of other classifiers[7, 10].

3.2.2 Description

A k nearest neighbor classifier compares the Euclidean distance between a pattern to be classified and all stored training exemplar patterns. The unknown pattern is assigned the class label which occurs most frequently among the k nearest training patterns. The only training required is storing all the training patterns in memory. The procedures of training and using a k nearest neighbor classifier are:

- **Training:**
  1. Store all training patterns in memory along with their class labels.
- **Classification:**
  1. An unknown pattern is presented to be classified.
  2. Find the k nearest neighbors of an unknown pattern by calculating the Euclidean distance between the unknown pattern and the stored exemplars.
  3. Assign to the unknown pattern the class label which occurs most frequently among the k nearest neighbors. In case of a tie between classes, break the tie randomly.
  4. Ready to accept a new pattern to be classified.

Since genetic algorithms rely on trying out many possible solutions, the fact that the k nearest neighbor classifier requires little training makes it attractive. A different type of pattern classifier, the radial basis function classifier[11, 16], was also used as

a pattern classifier in initial experiments. However, using the radial basis function classifier took considerably longer and did not always provide large improvements in accuracy. Thus, the k nearest neighbor classifier was used in all experiments. Since computation requirements increase as k increases, k was set to 1 for all experiments except when genetic algorithms were used to choose exemplars as described in Chapter 7.

3.2.3 Efficiency Improvements

Although the k nearest neighbor classifier requires little training, the amount of time required to classify input patterns is considerable, especially when there are many example patterns and the input data has high dimensionality. When using "leave-one-out" cross validation to estimate error rates[5], the Euclidean distance between every exemplar pattern taken one at a time and other exemplar patterns must be calculated. Computation thus increases as $O(N^2)$ where N is the number of exemplar patterns. Two improvements were made to the basic nearest neighbor classifier to reduce computation requirements.

Comparing Against the Shortest Squared Distance

Any method which can cut down the number of operations performed in distance calculations will reduce the execution time of a k nearest neighbor classifier. One such method which does not require much memory or time is to terminate distance calculation on an exemplar if it can not be closer than the kth nearest neighbor found so far. An algorithmic description of the method follows:

In calculating the distance to each exemplar pattern with d as the input dimension and *sum* as the total squared distance to the exemplar,

1. $Sum = 0, i = 1$

2. Calculate d_i^2, the squared difference between feature i of the exemplar pattern and that of the input pattern.

3. $Sum = Sum + d_i^2$

4. If $Sum >$ minimum squared distance to the kth nearest neighbor found so far then done for this exemplar.

5. Else $i = i + 1$.

6. If $i \leq d$ then go to 2.

7. Else done for this exemplar.

This distance calculation proceeds by iteratively summing the squared distances between features in the input and the exemplar patterns, one feature at a time. When the sum is greater than the squared distance to the kth nearest neighbor found so far, then clearly the current exemplar can not be one of the k nearest neighbors and the extra computation for the remaining features can be eliminated.

This method requires an extra comparison at each dimension. For low dimension problems on a Sun 3/110 workstation with a Floating Point Accelerator, the time used for performing the comparison may be larger than the amount of time saved by avoiding more calculations in further dimensions. However, for problems with large number of features, as the sum of squared differences is progressively increased, it becomes more and more likely to be greater than the kth nearest squared distance found so far. The chance of avoiding many calculations is high and the likely saving in time is large.

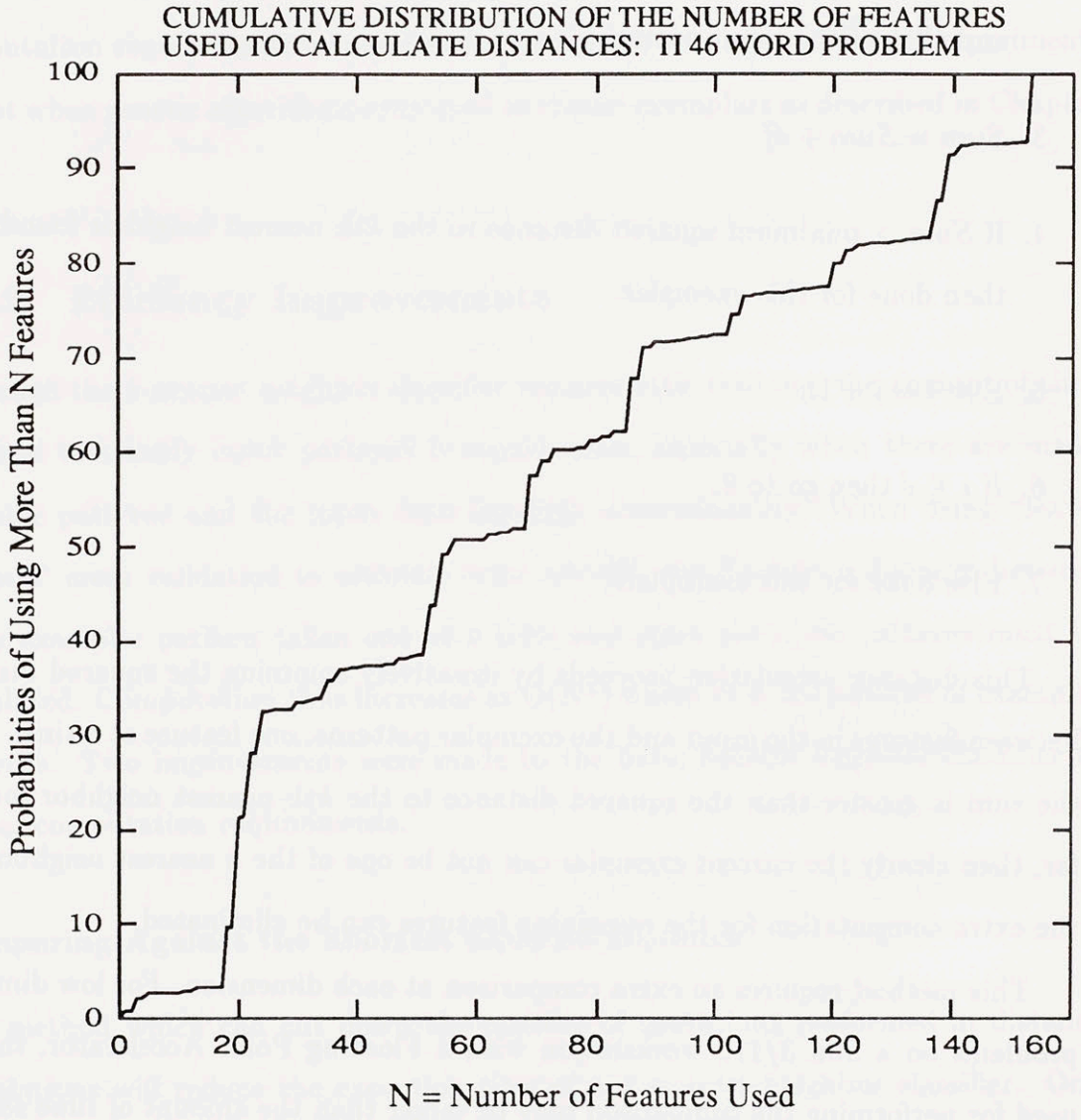

Figure 3.1: Cumulative distribution of the number of features out of 153 used to calculate Euclidean distances in a k-nn classifier with 90 input and exemplar patterns from the TI 46 word problem.

For example, for the Texas Instruments (TI) 46 word database problem described in next chapter, there are 153 input features. If a k nearest neighbor classifier with $k = 1$ is used with 90 training patterns, each distance calculation requires 305 additions plus subtractions and 153 multiplications per input pattern. With the modification, the cumulative distribution of the number of features used to calculate distances is shown in Figure 3.1. As can be seen, more than 50% of the distance calculations were completed before calculation proceeded to 80 features. The average time used by the modified version of the k nearest neighbor classifier in one experiment was 1.81 seconds. This is 44% less than the 3.21 seconds used by the unmodified version. All times were measured on a Sun 3/110 workstation with a Floating Point Accelerator in an experiment with 90 input patterns.

Conditionally Terminating Evaluation

While the above modification to the k nearest neighbor classifier does not introduce new errors, another modification (conditionally terminating evaluations) can statistically create new errors. However, the amount of time saved makes it worthwhile to accept the small risk of not accurately estimating the classifier accuracy.

Before introducing the approach, the reason for its use needs to be explained. In the experiments performed in this report, there were two classifier accuracies: the accuracy on training set and the accuracy on testing set. The accuracy on training set was the fitness function value used in the genetic search, while the accuracy on the testing set was not used in the genetic search. Through separating the patterns into the training set and the testing set, the generalization ability of the genetic search's solution can be checked. However, the k nearest neighbor classifier with $k = 1$ has an accuracy of 100% on the training patterns because the nearest neighbor of a pattern

is itself. To avoid this problem, the "leave-one-out" approach was used[5]. In this approach, every pattern in the training set is classified using the training set with itself removed. The overall accuracy on the training set is the number of correct classifications divided by the total number of patterns in the training set.

In many problems the size of the training set is large, thus calculating the training set accuracy using the "leave-one-out" approach takes a long time. It requires classifying each exemplar pattern using $N(N-1)$ Euclidean distance computations. Conditionally terminating evaluation is based on the idea that the eventual classification accuracy on the training set can be adequately estimated using only a small proportion of all the training patterns. At intervals of 50 training patterns, the program estimates the error rate and calculates the expected deviation from the true error rate with the following equation:

$$\sigma = \sqrt{\frac{(1-p)*p}{n}}$$

In this equation p is the estimate of probability of correct classification obtained so far and n is the number of samples tested so far. The value p is calculated with the following equation:

$$p = \frac{Number\ of\ training\ patterns\ correctly\ classified\ so\ far}{Number\ of\ training\ patterns\ classified\ so\ far}$$

Every 50 patterns, the sum of p and $2*\sigma$ (two times the standard deviation) is compared with the best performance obtained so far. If the sum of p and $2*\sigma$ is less than the best accuracy so far, then testing is stopped and the program accepts $p*100$ as the percentage correct. This procedure works because the percentage correct usually stabilizes quickly as one goes through the patterns in the training set. Thus the final percentage correct can be estimated from the percentage correct calculated

with a smaller sample set size. For example, Figure 3.2 shows that as the number of samples evaluated increases for the vector problem discussed in Chapter 4, the percentage correct gradually stabilizes and the final percentage correct is not far from the percentage correct calculated at $n = 100$. When $n = 50$, the estimated percentage correct plus two standard deviations is greater than the best percentage correct found so far, so the calculation continues. At $n = 100$, the estimated percentage correct plus two standard deviations is below the best percentage correct found so far, so the program would stop and accept the current estimated percentage correct as the final value. Since the percentage correct is far from the best percentage correct found so far, slight inaccuracy in classifier accuracy does not affect the genetic search.

Assuming that the deviation of p from the true percentage correct is a Gaussian random value with a mean of 0, it can be shown that there is only a 2.3% chance of having the true percentage to be greater than $p + 2 * \sigma$. There is thus only a one in forty chance that the accepted percentage correct would be lower than the percentage correct obtained when all patterns in the training set are classified.

When genetic algorithms are used for feature creation, the best feature subsets are frequently outstanding compared to the average feature. The feature subsets that have low percentage correct compared to the best feature subset encountered so far thus do not require accurate estimations of their accuracy. By using this approach and the above rule, the amount of computation required by the Parallel Vector problem, described in the next chapter, was reduced by a factor of 6. Since genetic algorithms took days to run, a reduction from 6 days to 1 day was significant. Accepting a 2.3% probability of a large estimation error resulted in much shorter computation time and made this research possible.

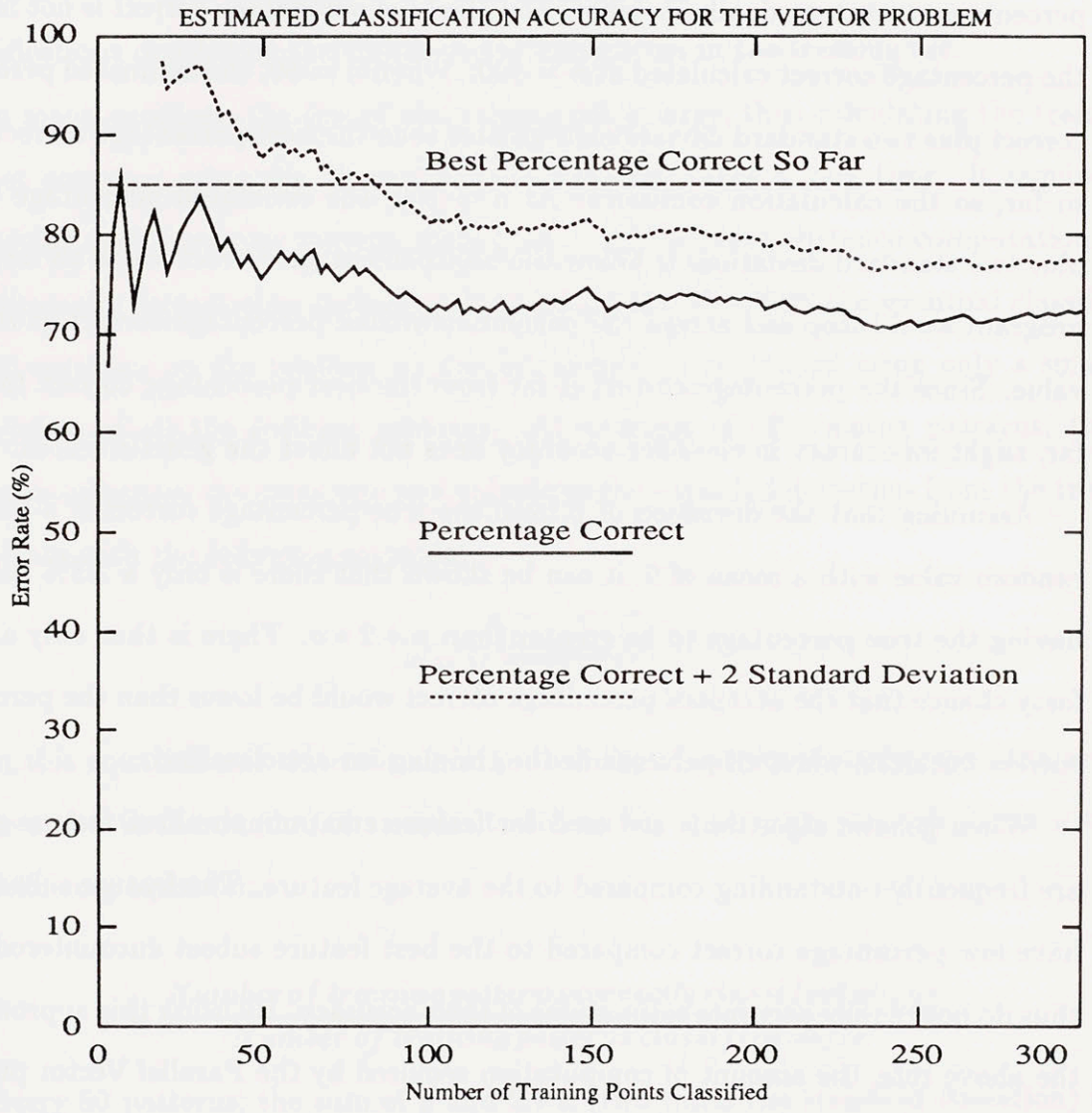

Figure 3.2: Variation in estimated accuracy (percentage correct) of a nearest neighbor classifier as more training exemplars are classified using a leave-one-out for the parallel vector problem.

Chapter 4

FEATURE SELECTION

4.1 INTRODUCTION

The problem of feature selection is to select that subset of features from a large initial set that provides best classification performance. As mentioned previously, for a problem with high dimensionality, conducting a full search through all the possible feature subsets is infeasible. This thesis focuses on genetic algorithms and uses forward and backward sequential search as a reference for comparisons.

4.2 METHODS

4.2.1 Representation and Evaluation

Every feature set or possible solution is represented by a bit string with d bits, where d is the maximum input dimension. Each bit of a solution represents a feature. If the bit is 1, then the feature is used, otherwise the feature is not used. Many different evaluation functions have been suggested to differentiate between good and bad feature subsets. Most are statistical and indirect and make strong assumptions on data distributions[5, 14]. The accuracy of a nearest neighbor classifier was used as the evaluation function in this study. The nearest neighbor classifier has the advantages of requiring no training time and providing results directly related to performance.

4.2.2 Feature Reduction

The goals of feature reduction are two fold: feature reduction and classifier performance improvement. These two goals may conflict with each other, as in cases where the full feature subset is actually the feature set to use if best classifier performance is required. There are cases, however, when reducing the number of features is more important than finding the best performing feature set possible. For example, the storage of the extra features may be very expensive and the collection of additional features may be time consuming. By reducing the number of features required, one can directly reduce memory requirements and the expense of collecting training data.

If classifier performance is used as the fitness function, genetic search will find a set of features with the best classifier performance and not necessarily reduce the number of features used. There must be an incentive for feature reduction to take place.

In the experiments performed in this study, feature reduction was an option that could either be used or not used. When the feature reduction option was used, an incentive in the form of a "bonus" was given to good strings which did not use many features. The bonus was a constant multiplied by the number of features not used and it was added to the classifier performance to derive the fitness value of the string.

Two different methods of giving the bonus were tried. In one policy called "only-the-best", only the string that performed better or equal to the best string so far would receive a bonus. This policy had the effect of ensuring that classification accuracy was not compromised for the sake of reducing the number of features used. If the full feature subset turned out to be the best set, then it would still have higher fitness value than other subsets which used fewer features but had lower classification accuracy.

Another policy of giving the bonus was called "bonus-above-the-threshold." With this policy, the user specified an acceptable percent correct threshold. All strings equaling or surpassing the standard would get a bonus proportional to the number of features they did not use. By using this policy, the user could specify that feature reduction was the more important goal once a certain minimum performance standard had been achieved. With this policy, it was possible for a string with a lower classification accuracy to have higher fitness value because it used fewer features. This policy is useful if an acceptable threshold can be found. It may be difficult to use if it is difficult to select a minimum performance threshold.

4.2.3 Reshuffling

The "only-the-best" policy has a problem in that the fitness function is time varying because it depends on the best performance measured so far. This causes strings evaluated earlier in the population to have an advantage over latter strings. Since the first string to be evaluated is compared against a classification accuracy of zero, it will definitely receive a bonus. It will be more difficult for latter strings to receive the bonus. There will be cases where two identical strings, one evaluated at the beginning and one evaluated near the end, have different fitness values.

To avoid biasing the population by favoring the strings evaluated earlier, a procedure called "reshuffling" took place periodically. When reshuffling was performed, the classification accuracy of all the strings was compared with the best classification accuracy obtained thus far. Reshuffling periodically was essential to rank all solutions using the same fitness function.

4.3 EXPERIMENTS

4.3.1 The NMR Problem

Problem Description

The first data used for feature selection was from a GE Corporate Research and Development database. The database contained 387 patterns, each with 15 features, for four different classes of patterns. The 15 features were derived from the intensity histogram of nuclear magnetic resonance (NMR) test images that were generated for calibration and test purposes using a standard glass sphere. Within the data were 27 normal patterns, 72 chopped phase shift patterns, 144 spike patterns, and 144 phase shift patterns. The latter three classes were defective images that would be obtained from hardware faults in the image processor. With this database, the goal was to distinguish between the patterns of the normal case and that of the three defective cases.

Results

The first 300 samples were used as the training set. The remaining 87 samples were used as the testing set. A population size of 100, with selective pressure of 0.05, mutative pressure of 0.50, and "only-the-best" policy (weight per feature = 5) were used with genetic search. An exhaustive search of all 32,768 possible subset combinations of 15 features was performed for this problem to find the optimal solution. This solution contained 8 features and had a classification error rate of 2.3% on the training set.

The progress of the genetic algorithm in searching for good feature subsets is shown in in Figure 4.1. The maximum fitness increased in stages, while the average fitness

of the population increased gradually from recombination 1 to 1,100. The effect of reshuffling can be seen at recombination cycle 1,200 of Figure 4.1, where maximum fitness dropped abruptly. The best string up to that point was no longer the best performing string, so it lost the bonus portion of its fitness function. The average fitness of the population also dropped because many strings within the population lost their bonus.

The amount of feature reduction achieved is shown in Figure 4.2. In this figure, the best classification accuracy obtained thus far and the number of features used by the best subset are shown. The top plot of Figure 4.2 shows that at recombination 1,200, the training set accuracy increased slightly. This increase in accuracy was gained at the expense of using three more features, as shown in the bottom plot. The tendency of the search was to reduce the number of features as much as possible without decreasing classifier accuracy. The search procedure favored low classification error rate over feature reduction. The final best string is the optimal feature subset with the classification error rate of 2.3% on the training set and 8 features. As a comparison, the classifier error rate with the full set of 15 features was 4.3%. Experiments with "only-the-best" policy using a weight per feature of 1 instead of 5 found the same features but with roughly half the number of recombinations.

In a second experiment, the "bonus-above-the-threshold" policy was used. The minimum acceptable classifier accuracy was chosen to be 92%. Once the minimum performance standard was reached, a string using fewer features would have higher fitness value. A string that used very few features would then be expected. The best string found by this method used only three features. Its classification accuracy was 92.3% on the training set, just above the minimum standard.

The results of genetic algorithms search in the two cases demonstrates that genetic

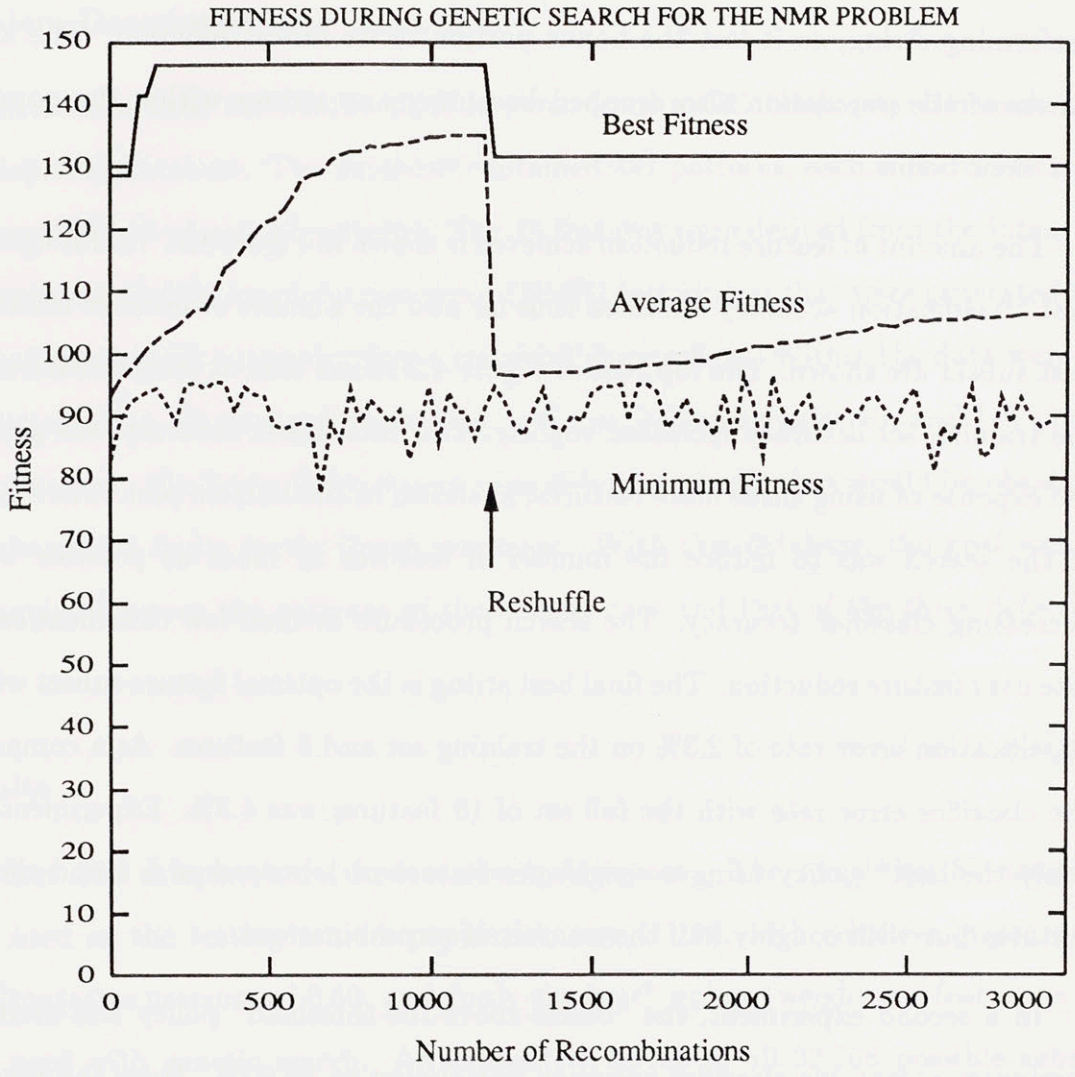

Figure 4.1: Fitness (percentage correct plus a bonus of 5 for every feature not used) versus the number of recombinations for the NMR problem.

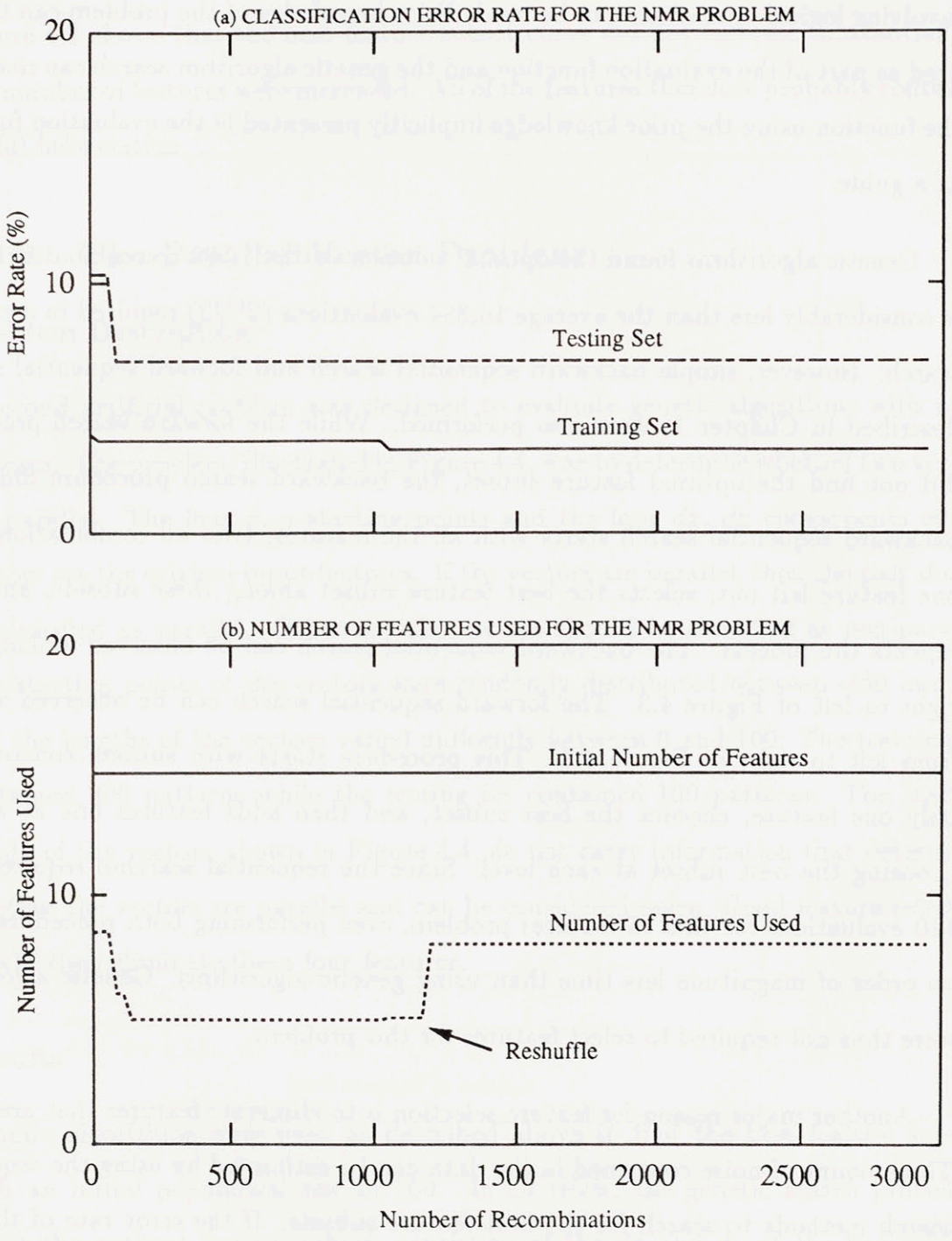

Figure 4.2: Genetic algorithms' progress in searching feature subsets for the NMR problem, (a) lowest error rate and (b) minimum number of features.

algorithms do find a good solution. In fact, even complicated evaluation functions involving logical operations can be used. Prior knowledge of the problem can thus be used as part of the evaluation function and the genetic algorithm search can maximize the function using the prior knowledge implicitly presented in the evaluation function as a guide.

Genetic algorithms found the optimal solution within 1,500 recombinations. This is considerably less than the average 16,384 evaluations ($2^{15}/2$) required in a random search. However, simple backward sequential search and forward sequential search, described in Chapter 1, were also performed. While the forward search procedure did not find the optimal feature subset, the backward search procedure did. The backward sequential search starts with all the features, tries all combinations with one feature left out, selects the best feature subset among these subsets, and then repeats the process. The backward sequential search can be observed looking from right to left of Figure 4.3. The forward sequential search can be observed looking from left to right of Figure 4.3. This procedure starts with subsets consisting of only one feature, chooses the best subset, and then adds features one at a time, choosing the best subset at each level. Since the sequential searches required only 120 evaluations for this 15 features problem, even performing both procedures took an order of magnitude less time than using genetic algorithms. Genetic algorithms were thus not required to select features for this problem.

Another major reason for feature selection is to eliminate features that are noisy. The amount of noise contained in the data can be estimated by using the sequential search methods to search for the best feature subsets. If the error rate of the best feature subset curve does not have a deep minimum, then the amount of noise in the data is not very large. On the other hand, if the error rate of the best feature subset

curve dips when not all the features are used, then certain features are probably noise. Figure 4.3 shows that the best feature subset curve did not increase substantially as the number of features were increased. All of the features therefore probably contained useful information.

4.3.2 The Parallel Vector Problem

Problem Description

A second artificial problem was designed to evaluate genetic algorithms with noisy features. The problem, illustrated in Figure 4.4, was to determine whether two vectors are parallel. The four x, y starting points and the four dx, dy components of two vectors are the original input features. If the vectors are parallel, then the pair should be classified as parallel. Otherwise the vectors should be classified as non-parallel. The starting points of the vectors were randomly distributed between -150 and 150 and the lengths of the vectors varied uniformly between 0 and 100. The training set contained 300 patterns while the testing set contained 100 patterns. The starting points of the vectors, shown in Figure 4.4, do not carry information that determines whether the vectors are parallel and can be considered noise. Good feature selection should thus eliminate these four features.

Results

Genetic algorithms were used as described above to find the best feature subsets with an initial population size of 100. In all trials, the genetic search procedure found the optimal feature subset, consisting of the dx and dy of the two vectors. In order to evaluate how effective genetic algorithms were in searching for the best feature, a collection of experiments were performed. Each experiment consisted of

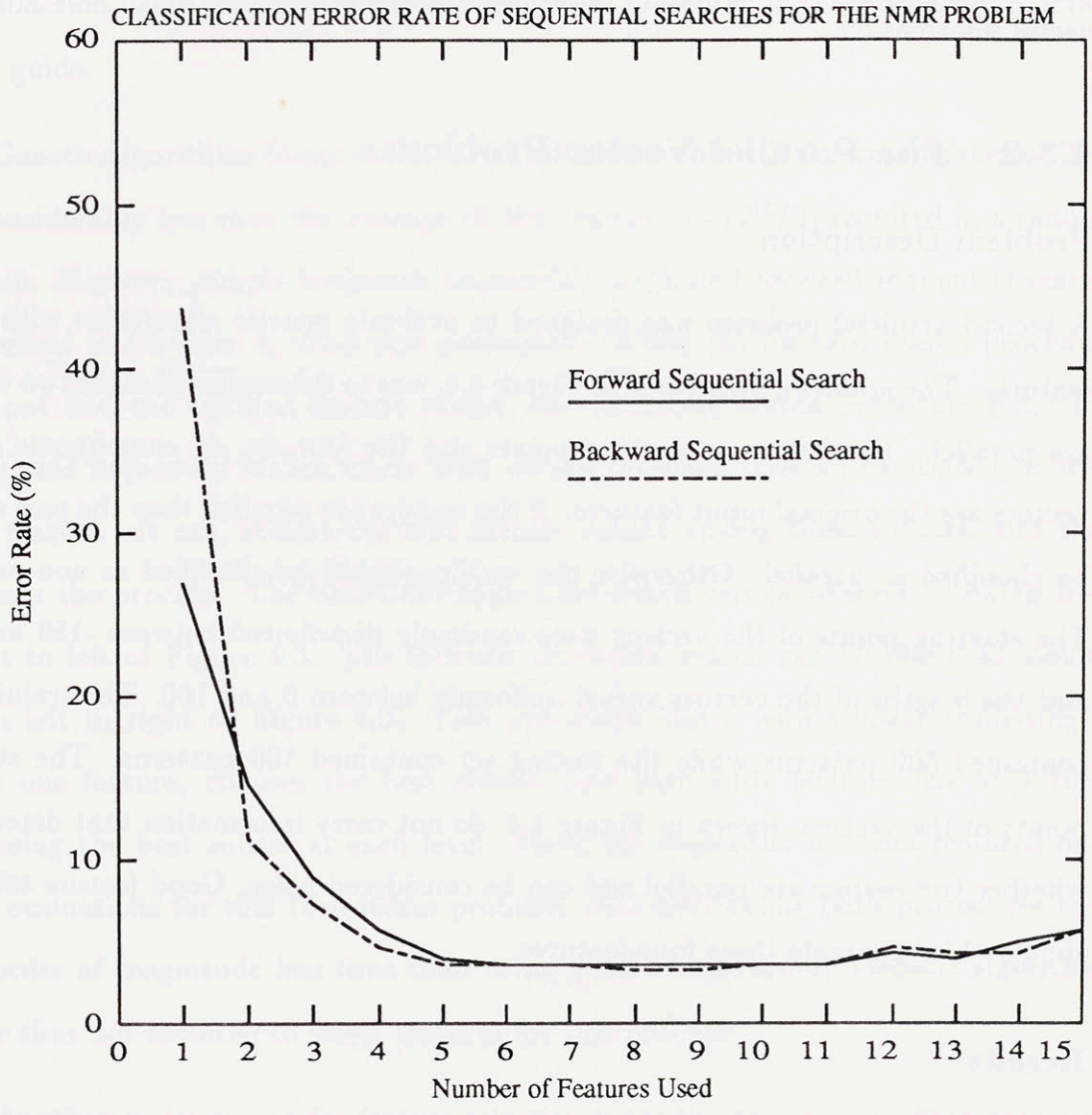

Figure 4.3: Forward and backward sequential search results for the NMR problem.

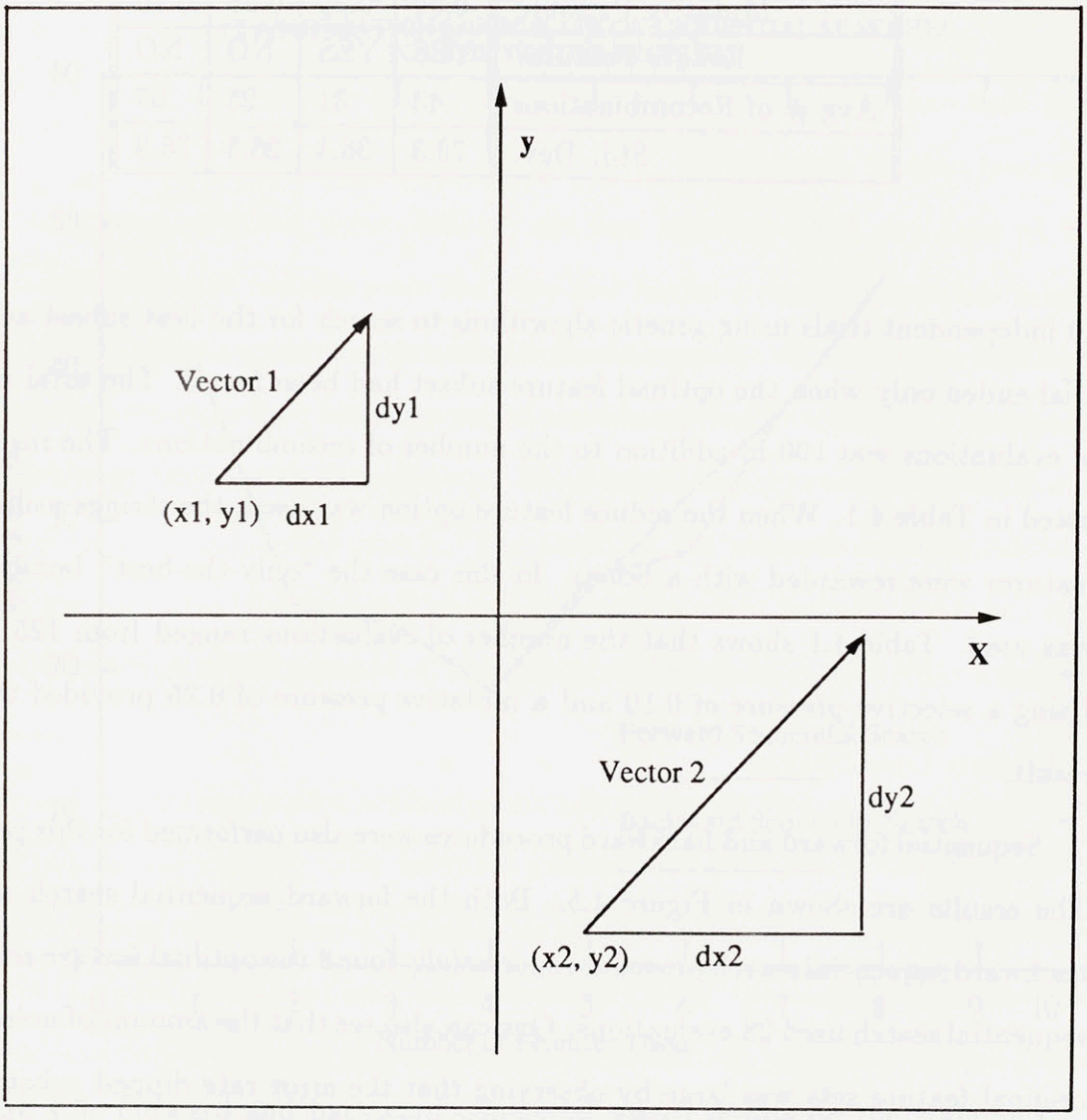

Figure 4.4: The parallel vector problem.

Table 4.1: Average Number of Recombinations Needed to Find the Best Feature Subset for the Vector Problem.

Experiment	I	II	III	IV
Selective Pressure	0.05	0.10	0.10	0.05
Mutative Pressure	0.50	0.25	0.25	0.50
Reduce Feature?	YES	YES	NO	NO
Avg # of Recombinations	43	31	25	57
Std. Dev.	70.3	36.3	26.5	76.9

10 independent trials using genetic algorithms to search for the best subset and each trial ended only when the optimal feature subset had been found. The total number of evaluations was 100 in addition to the number of recombinations. The results are listed in Table 4.1. When the reduce feature option was used, the strings using fewer features were rewarded with a bonus. In this case the "only-the-best" bonus policy was used. Table 4.1 shows that the number of evaluations ranged from 125 to 157. Using a selective pressure of 0.10 and a mutative pressure of 0.25 provided the best result.

Sequential forward and backward procedures were also performed for this problem. The results are shown in Figure 4.5. Both the forward sequential search and the backward sequential search procedure successfully found the optimal feature set. Each sequential search used 28 evaluations. One can also see that the amount of noise in the original feature sets was large by observing that the error rate dipped substantially when the number of features used was four. Error rates increased with the starting point features because these four features are noise. As a comparison, the error rate curves in Figure 4.3 have no significant dip in the middle because there were no purely noisy features in the NMR problem.

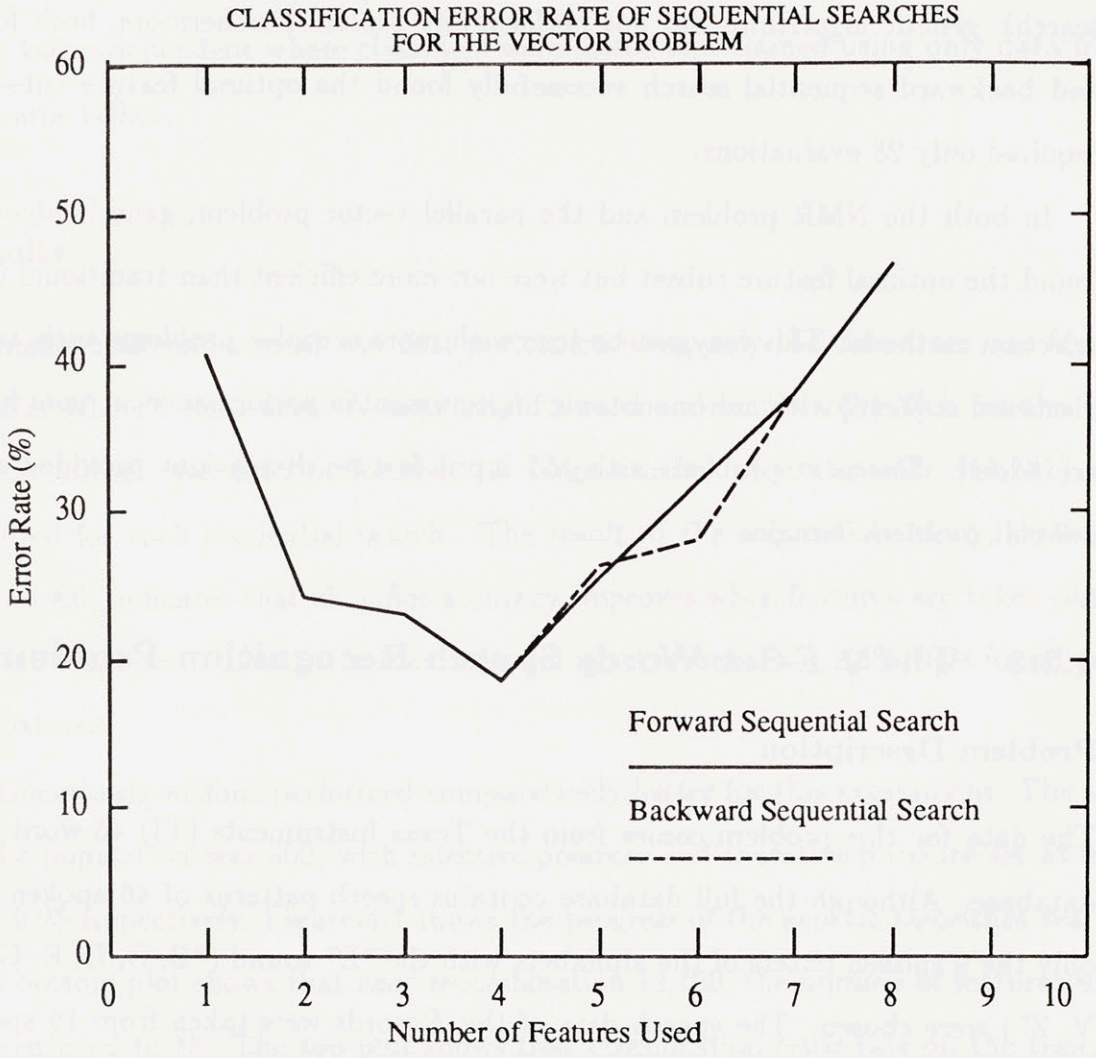

Figure 4.5: Forward and backward sequential search results on the training set for the vector problem.

When the number of evaluations required by the different approaches are compared, the performance of genetic algorithms is disappointing. Even choosing the best experiment, the average number of evaluations was 125, with a standard deviation of 26. Compared to 128 (the expected number of evaluations required in a random search), genetic algorithms did not perform any better. Furthermore, both forward and backward sequential search successfully found the optimal feature subset and required only 28 evaluations.

In both the NMR problem and the parallel vector problem, genetic algorithms found the optimal feature subset but were not more efficient than traditional feature selection methods. This may not be true with more complex problems such as those discussed in [7, 18] with non-monotonic improvement in performance as more features are added. The next problem with 153 input feature dimensions provided a more difficult problem domain.

4.3.3 The 9 *E*-Set Words Speech Recognition Problem

Problem Description

The data for this problem comes from the Texas Instruments (TI) 46-word speech database. Although the full database contains speech patterns of 46 spoken words, only the 9 spoken letters of the alphabets with the "E" sound ("B, C, D, E, G, P, T, V, Z") were chosen. The speech data of the *E* words were taken from 16 speakers, eight male and eight female. Waveforms were spectrally analyzed and encoded with a hidden Markov Model speech recognizer as described in [9]. The features were the average log likelihood distance and duration from all the hidden Markov nodes determined using Viterbi decoding. There were a total of 8 nodes with the duration and the average distance values from each node counting as two input features. The

final output of the hidden Markov model was also included in the feature set. This resulted in 17 features per word class. Since there were 9 different word classes, the total number of features was 153 (17 * 9). For each talker there were 10 patterns in the training set and 16 patterns in the testing set per word class. All experiments were talker dependent where classifiers were tested and trained using only data from the same talker.

Results

A smaller scale experiment was first performed using only the data from one female speaker (the F1 data set). A sequential forward and backward search for the best feature subset was first performed. A large number of evaluations (11,781) was required for each sequential search. The result of the sequential search, shown in Figure 4.6, indicates that classifier accuracy improves when features are taken out of the full feature set. Best performance (training set error rate = 2.2%) was found with 33 features.

Genetic algorithms performed comparatively better for this experiment. The size of the population was 500, with selective pressure and mutative pressure set at 0.05 and 0.25 respectively. Figure 4.7 shows the progress of the genetic algorithm search. The bottom plot shows that near recombination 12,100, the number of features used was reduced to 15. The top plot shows that classification error rate on the training set was 3.3% and on the testing set was 17.4%.

This testing set error rate is lower than the error rate of 18.8% provided by the original HMM recognizer used to segment input speech tokens[9]. Genetic algorithm feature selection thus improved the discrimination of the HMM recognizer. The features selected by the genetic search are listed in Table 4.3. Features 1, 3, 5, 7,

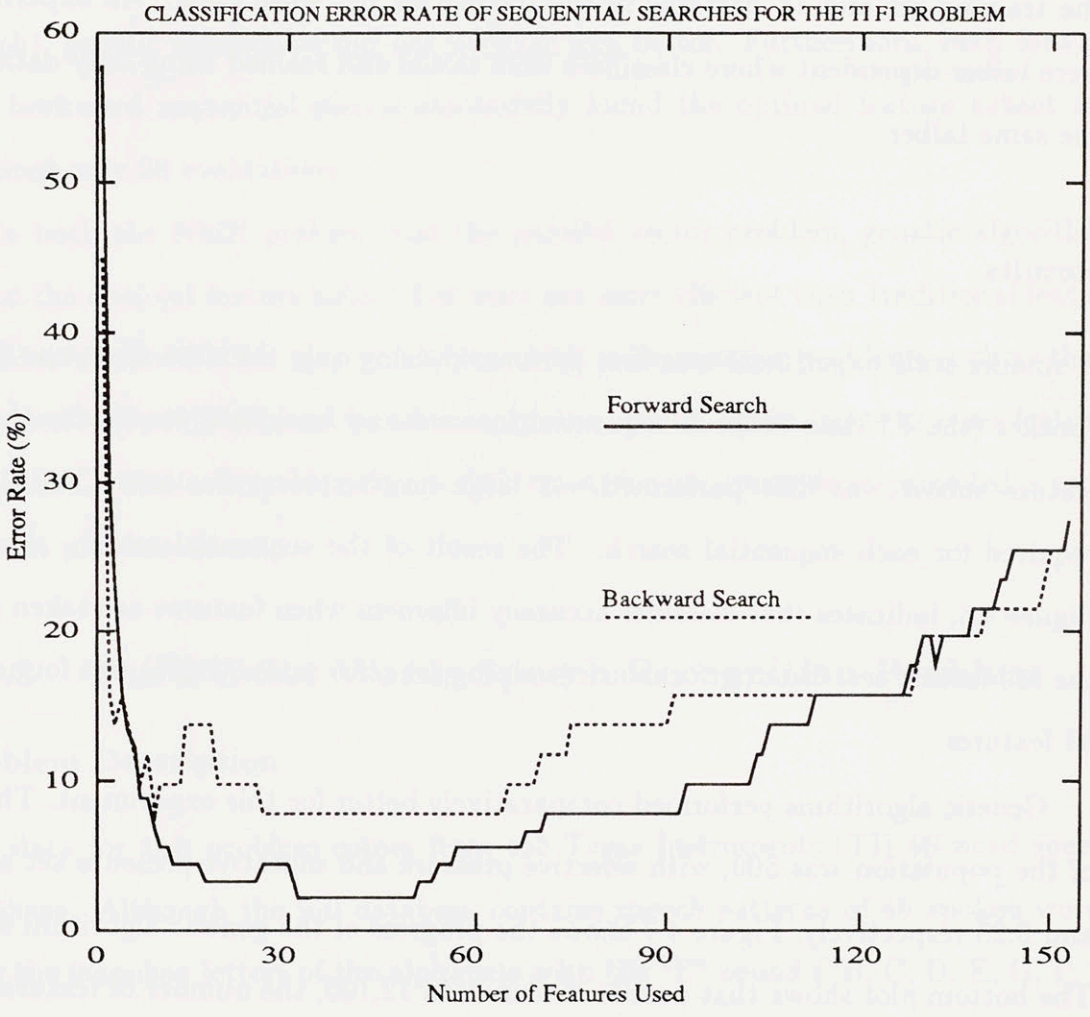

Figure 4.6: Forward and backward sequential search results on the training set for the TI F1 problem.

Table 4.2: Comparison between Sequential Search and Genetic Algorithms for the TI F1 Problem.

Method	Error Rate		# of Features	# of Evaluations
	Train	Test		
Genetic Search	3.3%	17.4%	15	20,000
Sequential Search	2.2%	18.5%	33	11,781

9, 11, 13, and 15 are the durations from hidden Markov nodes 1, 2, 3, 4, 5, 6, 7, and 8 respectively. Features 2, 4, 6, 8, 10, 12, 14, and 16 are log likelihood from hidden Markov nodes 1, 2, 3, 4, 5, 6, 7, and 8 respectively. Feature 17 is the hidden Markov model's final output. Table 4.3 shows that features were selected near the beginning of the words where spectral differences are greatest. Table 4.2 summarizes the comparison between the best feature subsets found by genetic search and sequential searches. Genetic algorithms found a feature subset which not only used fewer features but also had better generalization performance.

A second multi-talker experiment was performed using the data from four female talkers (F1, F2, F3, and F4) by combining the data from all talkers as if it came from one talker. A sequential forward and backward search for the best feature subset was first performed. Since the amount of data was four times the amount of data in the first experiment, this experiment took approximately 16 times longer to perform. Each sequential search still required 11,781 evaluations. Results from the sequential search are shown in Figure 4.8. This figure shows a rather drastic example of a problem with the forward sequential search. As the number of features increased, the classification error rate actually increased to about 60% before dipping back to about 20%. The feature subset at the 60% point consisted of features 1, 3, 5, 7, 9, 11, 13,

Table 4.3: Features Selected by Genetic Algorithms for the TI F1 Problem ("Y" means the feature is used, a blank means the feature is not used).

Word Class / Features	1	2	3	4	5	6	7	8	9
1. Duration of node 1									
2. Strength of node 1	Y	Y	Y	Y	Y	Y		Y	
3. Duration of node 2									
4. Strength of node 2	Y					Y	Y		
5. Duration of node 3									
6. Strength of node 3	Y	Y							Y
7. Duration of node 4									
8. Strength of node 4									Y
9. Duration of node 5									
10. Strength of node 5									
11. Duration of node 6									
12. Strength of node 6						Y			
13. Duration of node 7									
14. Strength of node 7									
15. Duration of node 8									
16. Strength of node 8									
17. Output of HMM									

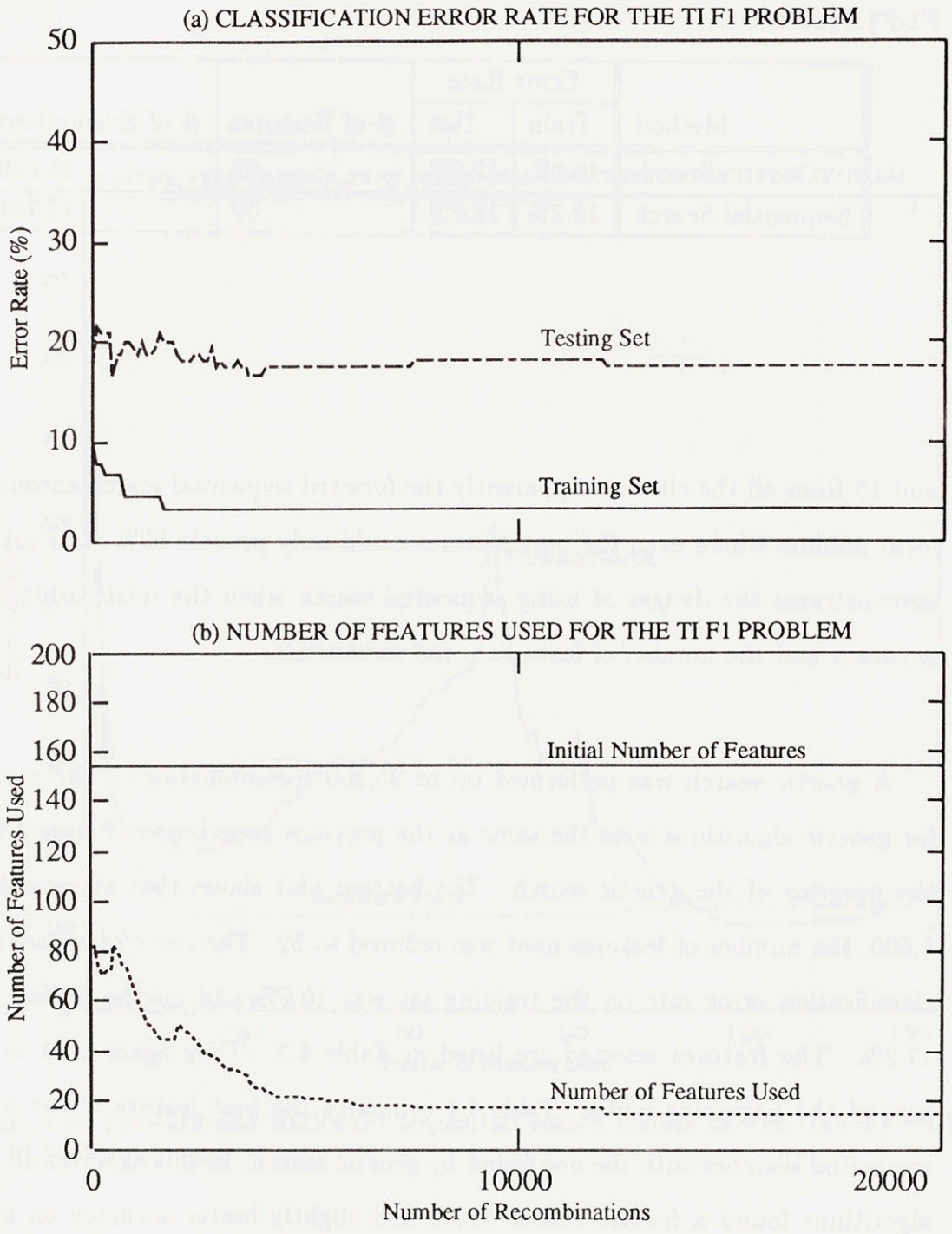

Figure 4.7: GA's progress in searching for feature subsets with high classification accuracy and few features for the TI F1 problem.

Table 4.4: Comparison between Sequential Search and Genetic Algorithms for the TI F1-F4 Problem.

Method	Error Rate		# of Features	# of Evaluations
	Train	Test		
Genetic Search	10.0%	17.2%	32	25,000
Sequential Search	12.2%	18.6%	19	11,781

and 15 from all the classes. Apparently the forward sequential search encountered a local minima where even the best features could only provide 60% error rate. This demonstrates the danger of using sequential search when the relationship between accuracy and the number of features is not monotonic.

A genetic search was performed up to 25,000 recombinations. The parameters for genetic algorithms were the same as the previous experiment. Figure 4.9 shows the progress of the genetic search. The bottom plot shows that at recombination 9,600, the number of features used was reduced to 32. The top plot shows that the classification error rate on the training set was 10.0% and on the testing set was 17.2%. The features selected are listed in Table 4.5. They again tend to cluster around the beginning nodes. Table 4.4 compares the best feature subset found by sequential searches with the one found by genetic search. In this experiment, genetic algorithms found a feature subset which had slightly better accuracy on both the testing set and the training set, however, this feature subset used 13 more features than the feature subset found by sequential search.

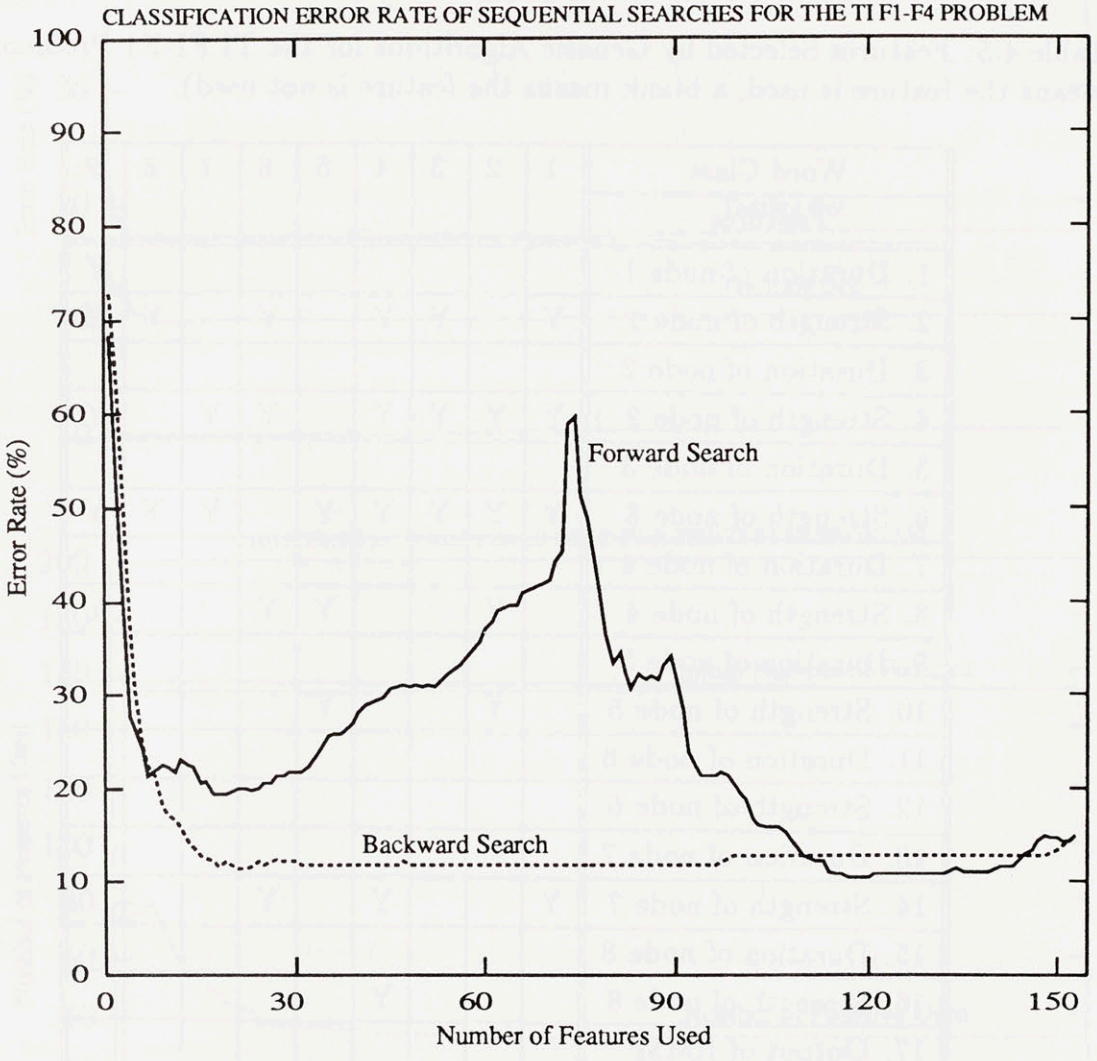

Figure 4.8: Forward and backward sequential search results on the training set for the TI F1-F4 problem.

Table 4.5: Features Selected by Genetic Algorithms for the TI F1-F4 Problem ("Y" means the feature is used, a blank means the feature is not used).

Word Class / Features	1	2	3	4	5	6	7	8	9
1. Duration of node 1									Y
2. Strength of node 1	Y		Y	Y		Y		Y	Y
3. Duration of node 2									
4. Strength of node 2	Y	Y	Y	Y		Y	Y		Y
5. Duration of node 3									
6. Strength of node 3	Y	Y	Y	Y	Y		Y	Y	Y
7. Duration of node 4									
8. Strength of node 4		Y			Y	Y			Y
9. Duration of node 5									
10. Strength of node 5		Y			Y				
11. Duration of node 6									
12. Strength of node 6									
13. Duration of node 7									
14. Strength of node 7	Y			Y		Y			
15. Duration of node 8									
16. Strength of node 8				Y					
17. Output of HMM									

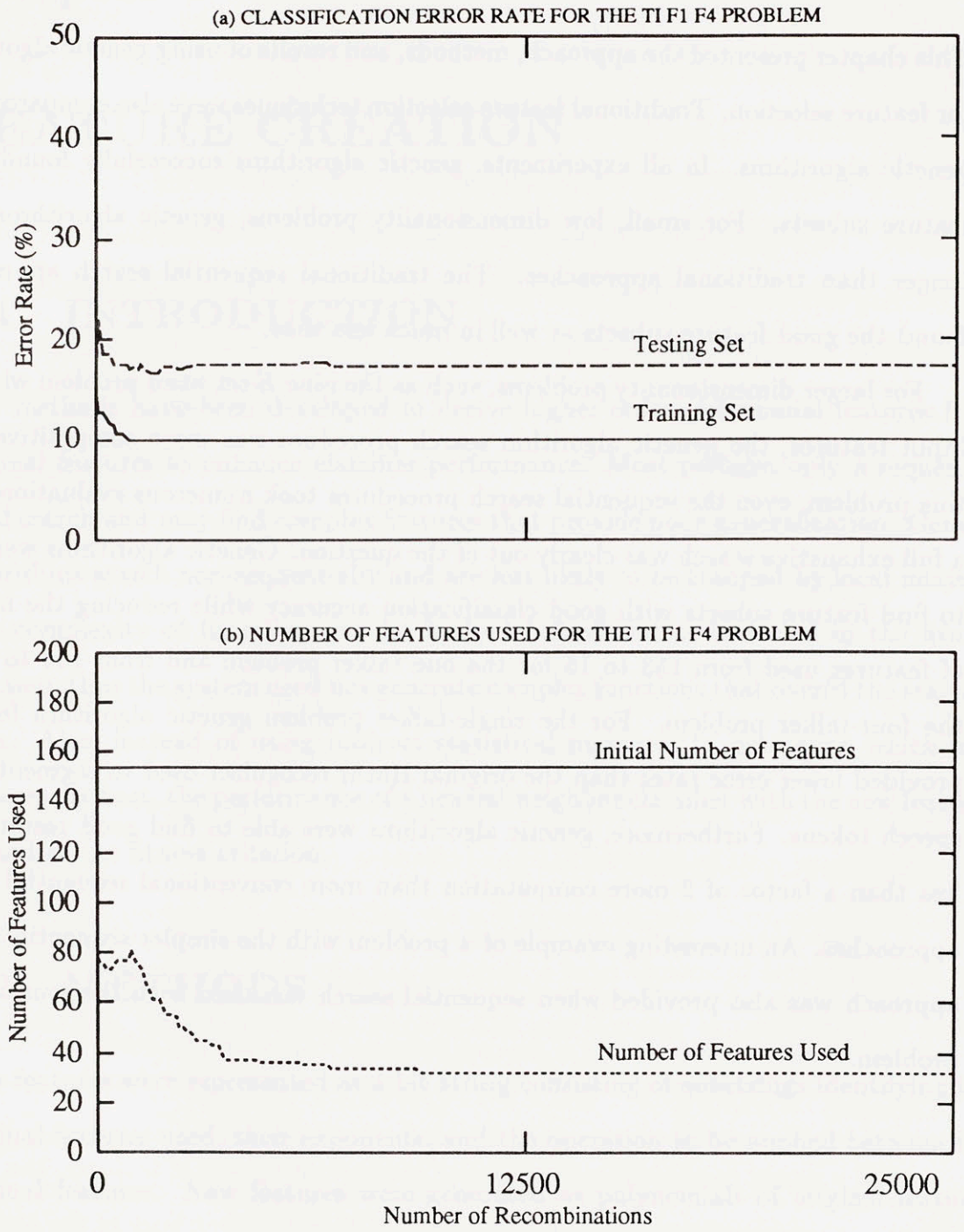

Figure 4.9: GA's progress in searching for feature subsets with high classification accuracy and few features for the TI F1-F4 problem.

4.4 SUMMARY

This chapter presented the approach, methods, and results of using genetic algorithms for feature selection. Traditional feature selection techniques were also compared with genetic algorithms. In all experiments, genetic algorithms successfully found good feature subsets. For small, low dimensionality problems, genetic algorithms took longer than traditional approaches. The traditional sequential search approaches found the good feature subsets as well in much less time.

For larger dimensionality problems, such as the nine *E*-set word problem with 153 input features, the genetic algorithm search procedure was more competitive. For this problem, even the sequential search procedures took numerous evaluations, and a full exhaustive search was clearly out of the question. Genetic algorithms were able to find feature subsets with good classification accuracy while reducing the number of features used from 153 to 15 for the one talker problem and from 153 to 32 for the four-talker problem. For the single-talker problem genetic algorithm features provided lower error rates than the original HMM recognizer used to segment input speech tokens. Furthermore, genetic algorithms were able to find good results with less than a factor of 2 more computation than more conventional sequential search approaches. An interesting example of a problem with the simpler sequential search approach was also provided when sequential search was used with the four speaker problem.

Chapter 5

FEATURE CREATION

5.1 INTRODUCTION

Few methods have been developed to derive higher order polynomial features from original features to enhance classifier performance. Most perform only a sequential local search and may find complex features that provide poor generalization. Genetic algorithms search non-sequentially and are less likely to be trapped by local minima. The complexity of functions can be limited beforehand or included in the fitness function, thus the system need not generate complex functions that overfit the training data. Also, instead of using indirect statistical measures to determine which new features are best, the performance of a nearest neighbor classifier with the new features is used as the fitness criterion.

5.2 METHODS

New features were represented as a bit string consisting of substrings identifying the original features used, their exponents, and the operation to be applied between the original features. New features were generated as polynomials of original features taken two at a time. This form was chosen admittedly with the experiment problem in mind, because the parallel vector problem to be described below required new features which were polynomials of two original features.

Each of the original features had a identifying bit string. The power of each original feature and the operator to be used between two original features were also identified with bit strings. For example, if feature 1 (f_1) had identification bit string 00 and feature 2 (f_2) had identification bit string 01, the string 00 1 01 1 01 represented the new feature ($f_1^2 - f_2^2$). The first two bits (00) of the string identified the first feature. Then one bit (1) was used to indicate the power of the feature. The same decoding mechanism was used for the second feature. Finally, the last two bits (01) identified the operator (-). The power of the original features was limited to either 1 or 2. The operators that could be used between original features were addition, subtraction, multiplication, and division. It was hoped that with these limits on the complexity of the created features, a more general set of features would be derived. In actual problems, if specific knowledge on the likely form of a useful feature is available, the operator set can be modified to take advantage of that knowledge.

The number of new features created was fixed *a priori* and the fitness function was classification accuracy (% correct). Initially, a population of new features was randomly generated. Each member of the population and was then evaluated according to the following procedure:

1. New features and original features encoded in each member were searched together using a forward sequential search to find the best subset of features. The features were evaluated using a nearest neighbor classifier and leave-one-out cross-validation.

2. If a newly created feature was used in the best subset of features, the part of bit string representing it was left untouched. Otherwise that part of the string was randomly scrambled. Since feature creation was highly random, randomly mutating the part of the bit string representing useless new features increased the size of search

space and avoided premature convergence.

Figure 5.1 illustrates the whole process of creating new features and deciding whether to keep the sub-string describing the new feature.

After the whole population had been evaluated, members of the population were selected, recombined, and mutated using the same genetic algorithm that was used for feature selection. In all experiments two new features were used. For many problems, often only one new feature does not enhance classification accuracy, and multiple new features must be used simultaneously.

Using a nearest neighbor classifier created a problem. The nearest neighbor classifier accuracy was easily affected by noise and any original or new feature that was essentially noise would decrease classification accuracy. For example, in a set of three newly created features, suppose two of them are exactly the needed features, but the third one is a random function of the original features. This third feature would be noise and distort the distance measurements used by the nearest neighbor classifier. After considering other classifiers that might be more noise tolerant (radial-basis function and back-propagation networks) and the amount of time required to train them, the nearest neighbor classifier was still chosen because it required no training. However, to avoid the effect of noisy features, a local sequential forward search was conducted on the set of original features and new features to select the best subset of original and new features during the evaluation of every string. The effect of noisy features could thus be eliminated and the search space was reduced. Without sequential search, both the correct new features and the correct set of features must be found at the same time using genetic algorithms, otherwise the usefulness of the newly created features would not be apparent. With sequential search, good features can be found one at a time and still have their usefulness not distorted by other

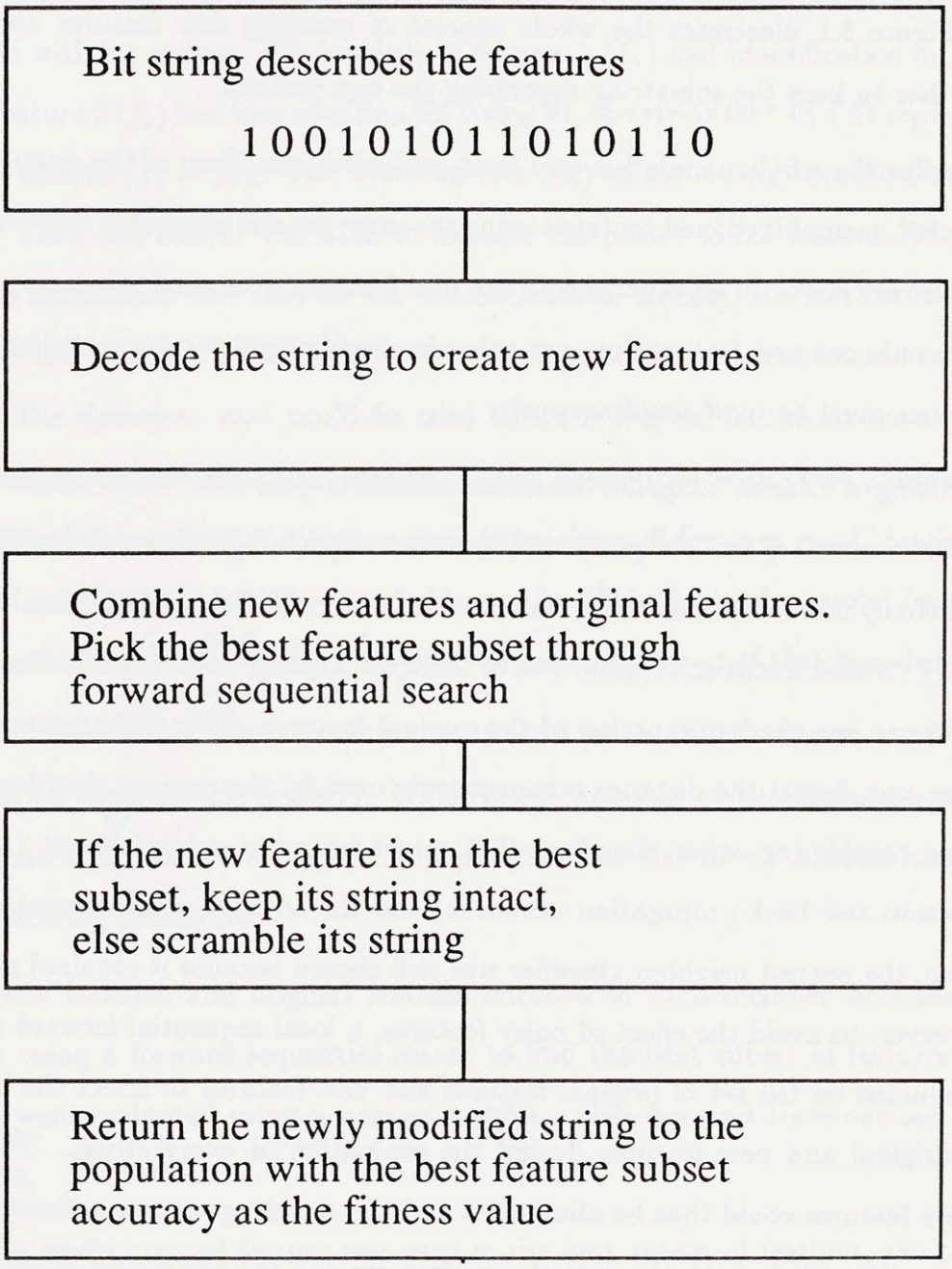

Figure 5.1: A block diagram of the feature creation process in which local search is used to eliminate features that are noise.

noisy features. In cases where local search fails due to non-monotonicity, then the correct features and correct set of features must be found simultaneously by genetic algorithms.

5.3 EXPERIMENTS

5.3.1 The Parallel Vector Problem

Problem Description and Results

The artificial parallel vector problem described in the previous chapter was first used as a test problem because higher order functions (slopes) are known to improve performance. The starting point coordinates of the vectors were taken out of the data set to reduce the size of the search space. Original features consisted of the four dx and dy components of the vectors. In an experiment using the (+, -, *, /) operator set, a population size of 200, and 300 training patterns, genetic algorithms successfully found a useful set of features. In fact, genetic algorithms found a better set of features than the slopes (dy_1/dx_1 and dy_2/dx_2) that should be very good features. These slope features with the nearest neighbor classifier provided a 6.7% error rate on the training data. This compared favorably with the 18.7% error rate using only the four original dx and dy features. It was also much better than the error rate obtained using carefully tuned GMDH (13.3%) and radial-basis-function classifiers (8.3%) [11]. It was clear that by having the correct features, the performance of classifiers could be improved a great deal. Genetic algorithms, however, found the following set of new features: dy_2/dy_1 and dx_2/dx_1.

The classification error rate using this set of features was 2.7% on the training data. The reason for improved performance with these features is demonstrated in

Figure 5.2 and Figure 5.3. Genetic search found regularity in the data which was not at first apparent. The original data set was created with $dx1$ and $dy1$ ranging uniformly between -50 and 50, and then having $dx_2 = k * dx_1, dy_2 = k * dy_1$, k ranging uniformly between 0.0 and 2.0. The slope can range from $-\infty$ to ∞, with most vectors clustering between -1 and 1. However, the features dy_2/dy_1 and dx_2/dx_1 were bounded to be between 0.0 and 2.0. The parallel input patterns were thus clustered more densely when dy_2/dy_1 and dx_2/dx_1 were used as features (Figure 5.3) instead of using the slopes (Figure 5.2). This clustering improved the performance of the nearest neighbor classifier.

Figure 5.4 shows the progress of genetic algorithms on the vector problem. The curves shown are the average of 5 independent runs. The testing set performance is the percentage correct rate on 100 test patterns that were not used by genetic algorithms in finding new features. Good performance was achieved near recombination 1,500. The total number of evaluations was (1,500 + 200) * 28 = 47,600, where 28 is the number of evaluations required by forward sequential search. The number of possible distinct solutions was 357,760 $((K^3 + 3 * K^2 + 2 * K)/6, K = 128)$. The expected number of evaluations that were needed in a random search was half of that, 178,880. Genetic algorithms took roughly 1/4 of the average number of evaluations required in a random search. These 47,600 evaluations took 31 hours on a Sun 3/80.

5.4 SUMMARY

Genetic algorithms were used to create new high order polynomial features. Genetic algorithms successfully found new features which reduced the error rate from 19% to 3% on one artificial problem and were shown to be a promising search procedure for finding higher order features. However, the amount of time required to utilize

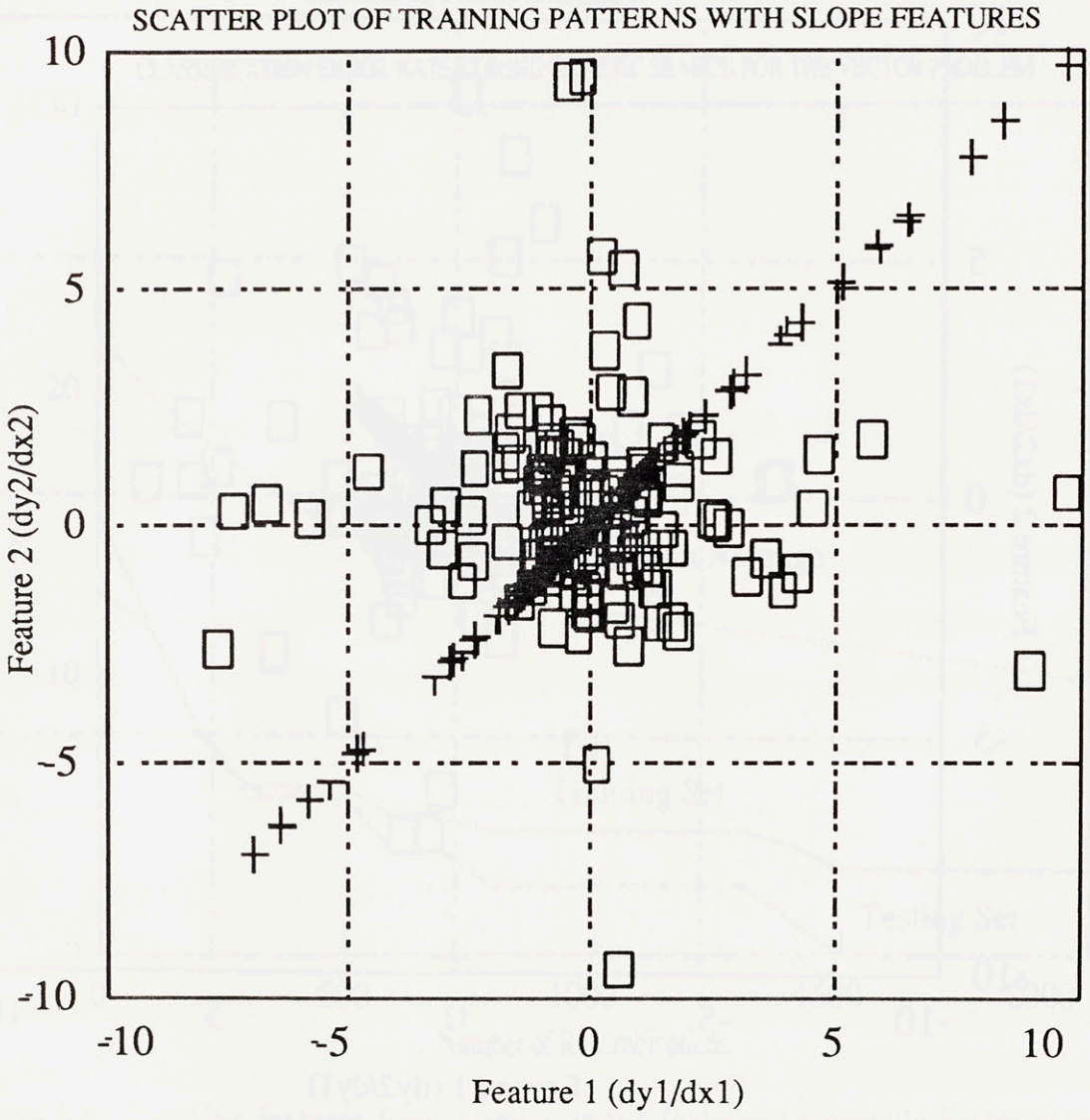

Figure 5.2: Scatter plot of training patterns for parallel (+) and non-parallel (□) vectors using slope features for the vector problem.

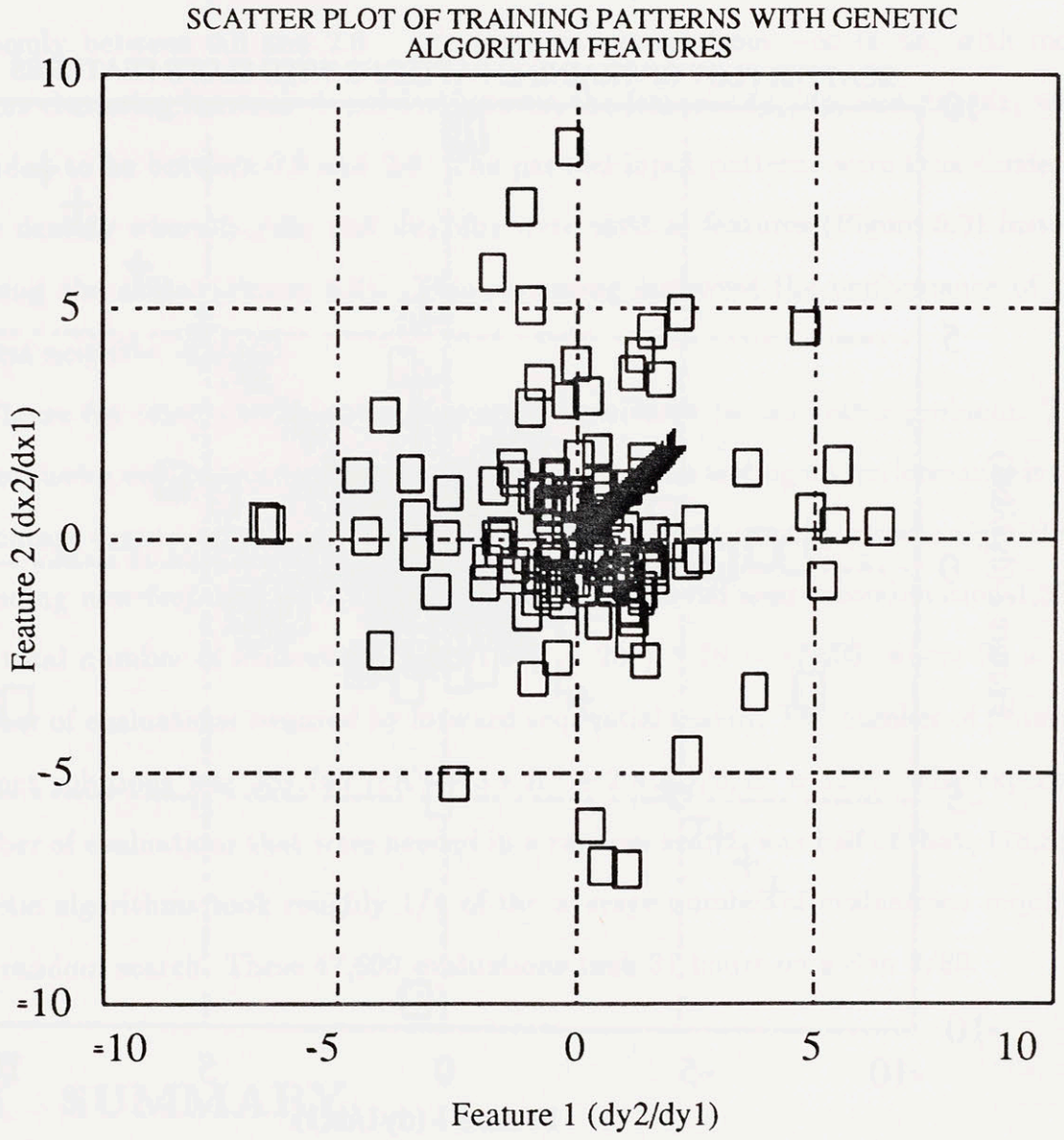

Figure 5.3: Scatter plot of training patterns for parallel (+) and non-parallel (□) vectors using genetic algorithm features for the vector problem.

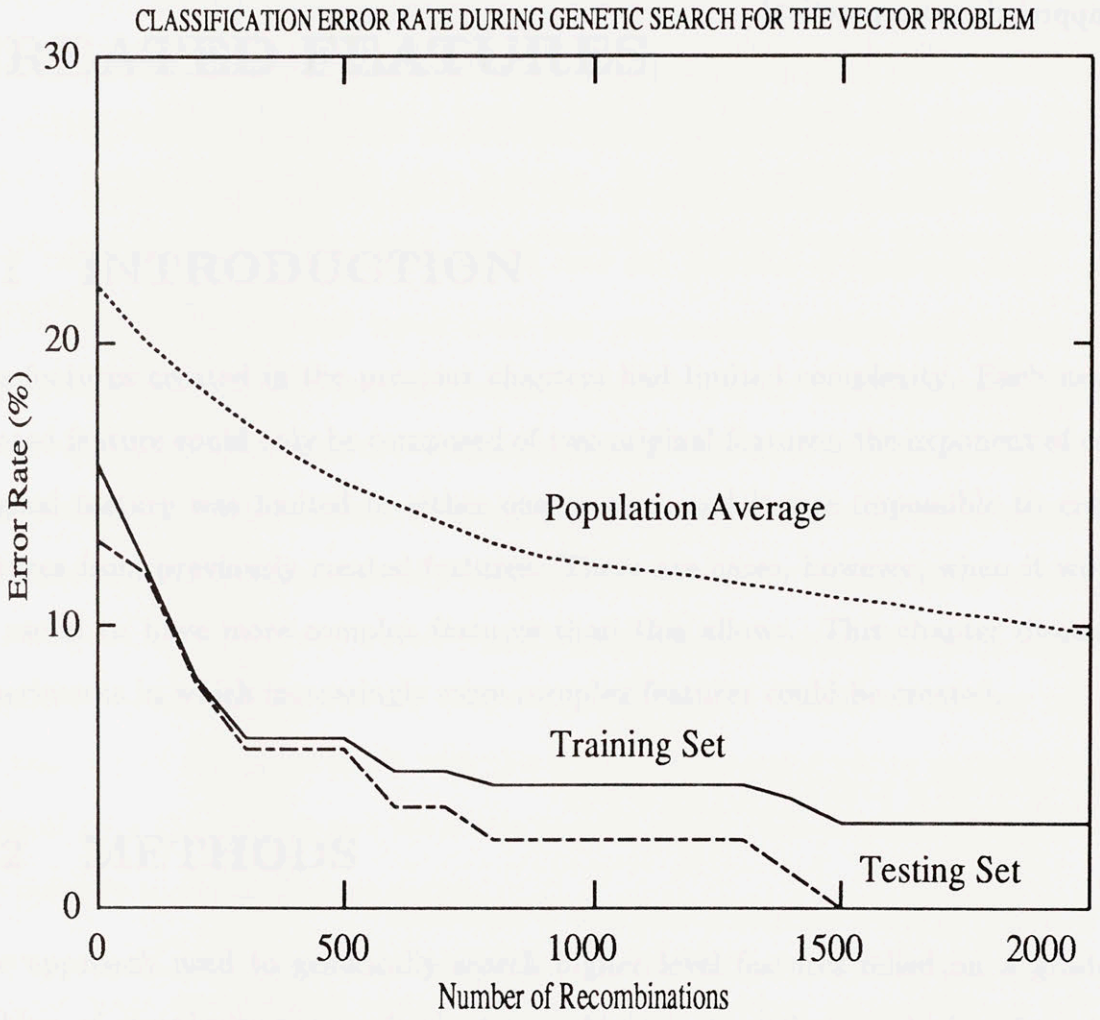

Figure 5.4: Creating features from original input features to provide better classification accuracy for the parallel vector problem.

this approach grows very quickly with the number of features. Even for the simple problem presented in this chapter, run times were more than a day. Such times are practical only for difficult problems when a good solution is essential. Shorter run times achieved through parallelism or more powerful computers could make this approach more practical.

Chapter 6

INCREASING THE COMPLEXITY OF CREATED FEATURES

6.1 INTRODUCTION

The features created in the previous chapters had limited complexity. Each newly created feature could only be composed of two original features, the exponent of each original feature was limited to either one or two, and it was impossible to create features from previously created features. There are cases, however, when it would be useful to have more complex features than this allows. This chapter discusses experiments in which increasingly more complex features could be created.

6.2 METHODS

The approach used to genetically search higher level features relied on a gradual buildup of complexity over multiple stages. At each stage the complexity of created features was limited. Once the accuracy of the classifier had converged at one stage, another stage was begun where more complex high order features were allowed. This improves generalization by creating simple features first and by creating more complicated features only when simpler features are not satisfactory.

It is difficult to define a condition that positively signals the successful completion

of searching for good features of a limited complexity at one stage. The number of recombinations since the last improvement in classifier accuracy was used as an estimate of how well the search space has been explored during one stage. In the beginning of the genetic search process, it is easy to find features which improve the classifier accuracy, thus the number of recombinations since the last improvement is frequently reset back to zero. After very good features have been found, it is increasingly more difficult to find even better features, so the number of recombinations since last improvement increases steadily, and eventually reaches a preset limit.

Once the limit is reached, the best features that were created thus far become a part of the original feature set, and even newer features are created using the original features and these previously created features. The complexity, or order, of the created features can thus increase steadily. For example, suppose a necessary feature is the sum of two created features. With this process, it is possible for this feature to be created. Previously, with the limit on the complexity of the created features, such features could not be created.

6.3 EXPERIMENTS

6.3.1 The Parallel Vector Problem

The parallel vector problem used to explore feature creation was again used. Figure 5.3 shows that with the right features, the parallel vector patterns are closely clustered in the diagonal line from (0,0) to (2,2) in the feature space. The denser one class of patterns clusters together, the less likely a nearest neighbor classifier will make a mistake. It is therefore reasonable to assume that creating a new feature consisting of the differences between the ratios may be a good feature. With this

feature, all parallel patterns would have a value of zero and cluster on one point.

Experiments were performed with a complexity interval limit set to 2,000, i.e. the complexity of features created increased when there hadn't been improvement in classifier accuracy for 2,000 recombinations. The value of 2,000 was chosen since in previous experiments, the ratio features were found within 2,000 recombinations. It was therefore likely that when the complexity level is increased, the already created features would be the correct ratio features. In other problems where the correct setting is unknown, there would be a tradeoff between search thoroughness and computation time.

Figure 6.1 shows the progress of the genetic search in finding more complicated features. The ratio features were first found near recombination 700. After the error rate had not changed for 2,000 recombinations, the complexity of the created features was increased at recombination 2,700. At this point the two ratio features and the four original features were treated as if they were six original features. The final feature found after this point was $(dx_2 * dy_2)/(dx_1 * dy_1)$. Classification error rate for the training set decreased to 0% with this feature. All parallel vector patterns have a value of 1 for this feature and the whole parallel vector class has been clustered into one point in the input feature space.

The classification error rate using the final created feature alone was 0% for for both the training patterns and testing patterns. The original best classification error rate was 3% for the training patterns and 0% for the testing patterns. This improvement in training pattern performance was 3%, which is statistically significant. The utility of the new feature can also be seen by observing the distribution of classes provided by this feature as shown in Figure 6.2. Here it can be seen that all the parallel vector patterns are on one point with feature value of one while most of

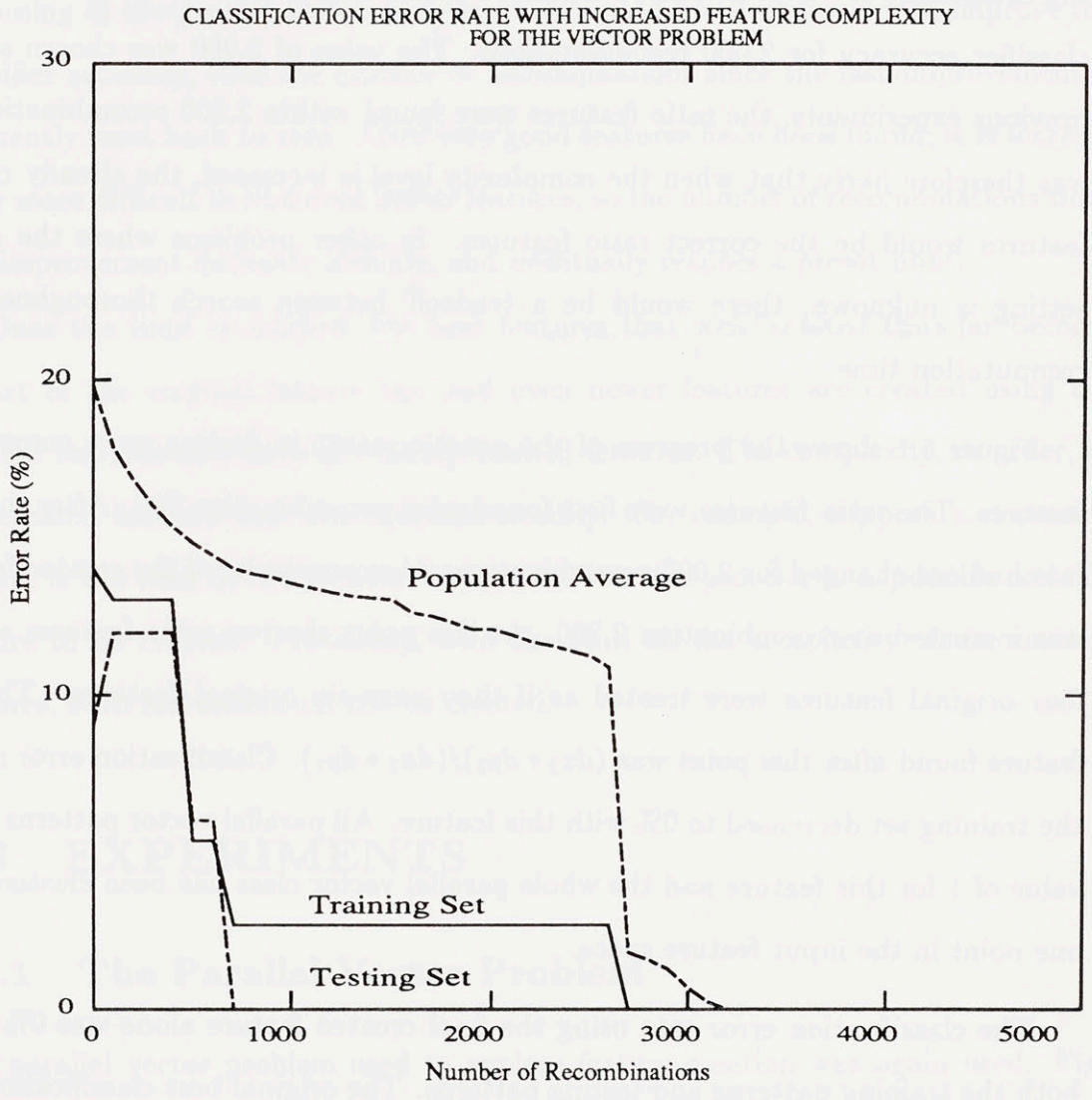

Figure 6.1: Creating features out of created features to improve classification accuracy for the parallel vector problem.

the non-parallel vectors are around 0. The separation between classes is thus very distinct.

6.3.2 The Vowel Problem

The vowel database, originally collected by Peterson and Barney[15], was also used to test feature creation. There were ten classes, each class being a word starting with "h" and ending with "d", with a vowel in between ("head", "hid", "hod", "had", "hawed", "heard", "heed", "hud", "who'd", and "hood"). The patterns were collected from 67 speakers, including men, women, and children. A total of 338 patterns was used as the training set and 333 patterns were used as the testing set. Each pattern consisted of two features which were the two formant frequencies of the vowel determined by spectrographic analysis. A formant is a resonant frequency of the speaker's vocal tract. The two lowest formants were used in the database. Yuchen Lee and Kenney Ng have also used this set of data to perform experiments on other types of classifiers[10, 13].

A population of 200 was again used with selective pressure of 0.05, mutative pressure of 0.25, and k of 5. In this experiment, all features were normalized within the range of 0 to 1 by transforming the highest number within the feature to 1 and the lowest number within the feature to 0. Results presented in Table 6.1 and Figure 6.3 show that classification error rates on both training and testing sets decreased as genetic algorithms created more complex features. Three features ($f_2 * f_1$, $1/f_2$, and f_1^2/f_2) created near recombination 700 in addition to the second original feature (f_2) provided the lowest classification error rate during the first stage of creating features (24.6% for the training set and 17.4% for the testing set). As a comparison, classification error rates using only the original features are 27.2% for the training set

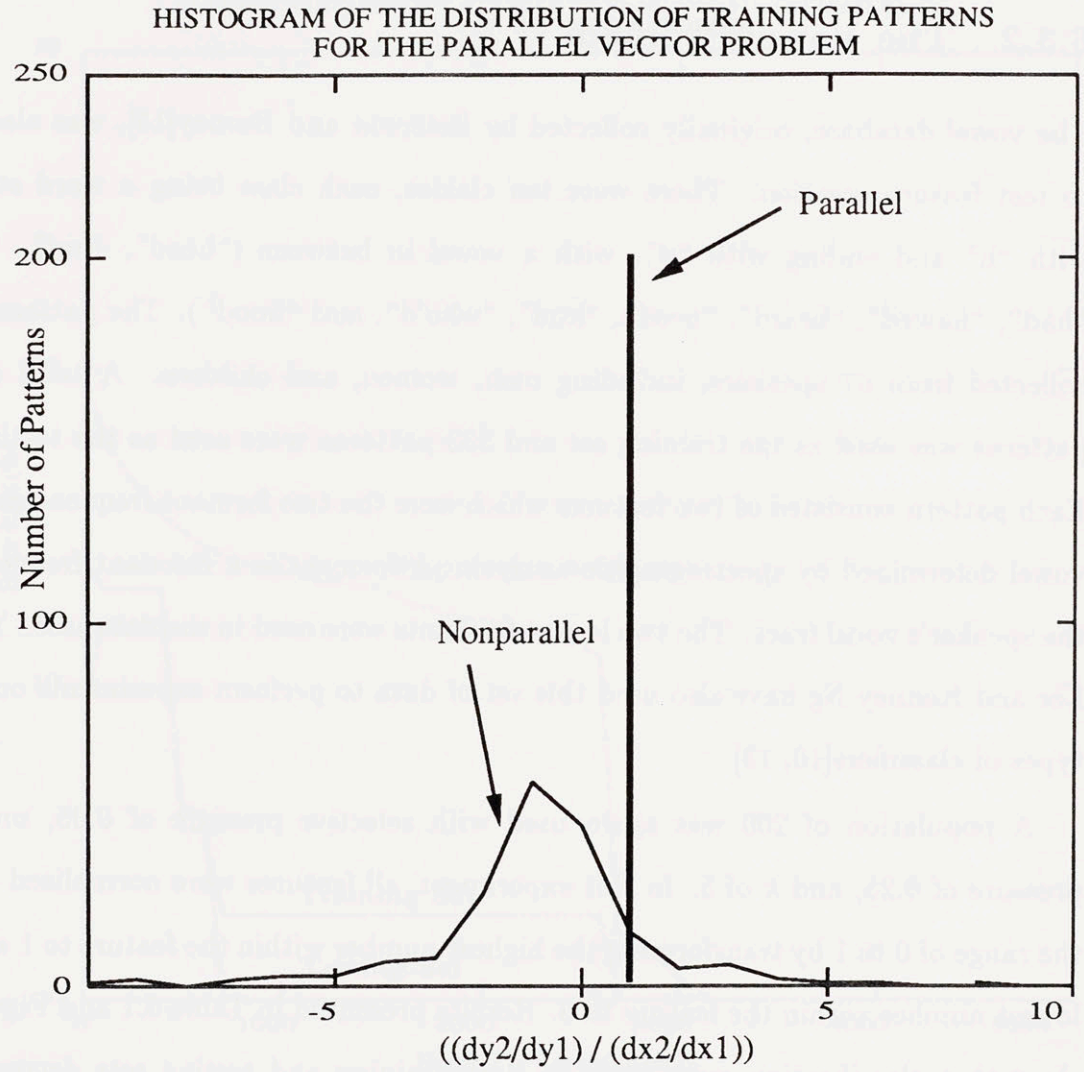

Figure 6.2: Distribution of the training patterns belonging to parallel and non-parallel classes when the feature $((dy_2/dy_1)/(dx_2/dx_1))$ is used for the parallel vector problem.

Table 6.1: Classification Error Rate for the Vowel Problem as New Features Are Created.

Condition	Error Rate		# of Features
	Train	Test	
Original	27.2%	20%	2
After First Stage	24.6%	17.4%	4
After Second Stage	23.6%	17.1%	6

and 20% for the testing set.

When additional complexity was allowed at recombination 3,100, two additional higher order features ($f_3^2 * f_3^2$ and f_6^2/f_1^2) were created where f_3 is the first created feature stored in the previous stage ($f_2 * f_1$) and f_6 is another previously created feature (f_2/f_1^2). Using these two newly created features and the four features used at recombination 700 results in the training set classification error rate of 23.6% and the testing set classification error rate of 17.1%.

6.4 Summary

This chapter presented the results of using genetic search to gradually create more complicated features after the created feature performance stabilizes. In the parallel vector problem, a reduction of 3% in error rate was achieved. Genetic algorithms also reduced classification error rate for the vowel problem. The gradual increase of complexity allows the feature creation approach to be used on real problems. The simpler fixed complexity approach was limited because the complexity of useful features is often not known *a priori* and might be too limited. Gradually increasing complexity allows an incremental, albeit still limited, enlargement of the search space.

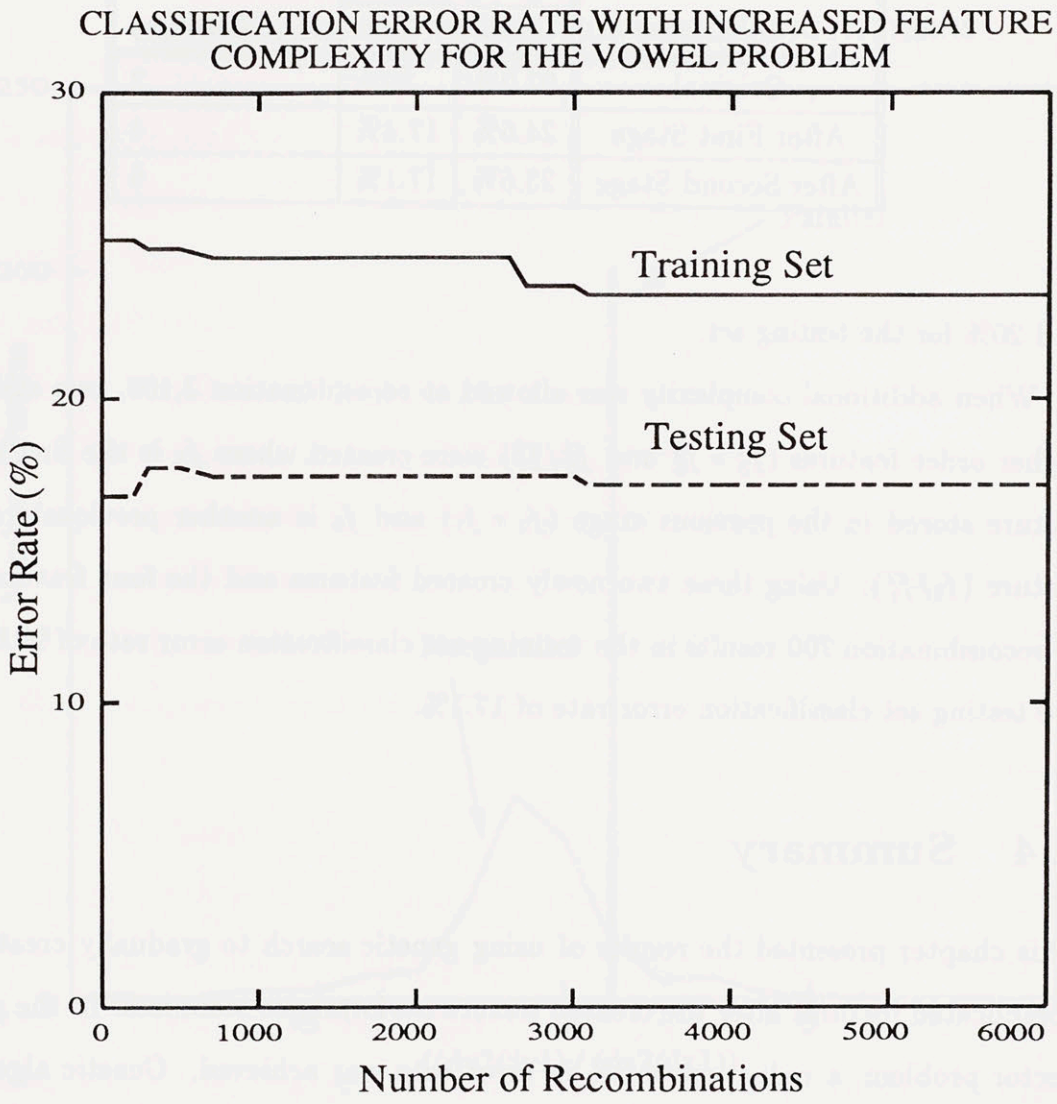

Figure 6.3: Creating higher order polynomial features to reduce classification error rate for the vowel problem.

Chapter 7

EXEMPLAR SELECTION

7.1 INTRODUCTION

The performance of a k nearest neighbor classifier typically improves as more training patterns are stored. This characteristic often makes a k nearest neighbor classifier impractical because both classification time and memory requirements increase linearly with the number of training patterns[10]. A k nearest neighbor classifier with many training patterns may thus need too much time or memory to provide real time response.

Previous approaches to reducing the classification time of nearest neighbor classifiers include using kd-trees and the Metric-Space Search Algorithm(AESA)[21]. Kd-trees partition the input space so that not all training pattern distances need to be calculated. The AESA creates a look-up table of intra-training-pattern distances which also reduces the number of distance calculations required. Unfortunately, the benefits of these approaches diminish dramatically when the input dimensionality is high. A better way to reduce classification time of a k nearest neighbor classifier is to directly reduce the number of training patterns stored. With fewer training patterns, both classification time and memory requirement are reduced.

The number of exemplars stored can frequently be reduced a great deal without sacrificing classification accuracy. For example, in Figure 6.2, there are actually over 100 training patterns all with the value of 1 when only one would suffice. By

carefully selecting training patterns, the number of training patterns can often be reduced dramatically.

One sequential approach to selecting training patterns is called the condensed nearest neighbor classifier[5]. This approach is outlined below:

1. Start with an empty set of training exemplars.

2. Select each individual training pattern sequentially.

3. Classify the selected training pattern using the current set of exemplar patterns. If the pattern is classified incorrectly, keep the training pattern as an exemplar, otherwise discard the training pattern.

4. Repeat steps 2 and 3 until all patterns have been selected.

One problem in using this approach is that it is biased. Training patterns evaluated earlier are more likely to be included as exemplars. An alternative approach is to use genetic algorithms to select the whole training set simultaneously. With this approach the order of the training patterns does not affect the selection of training patterns. Furthermore, if a given number of training patterns needs to be used simultaneously in combination to improve classifier accuracy, the genetic algorithm search procedure is more likely to find the group than a simple sequential search. This is similar to the feature selection problem when two individually good features would not necessarily be a good set of features.

7.2 METHODS

7.2.1 Using Bonus to Reduce the Number of Patterns

The original feature selection and creation program was enhanced with the capability of selecting training set patterns. Each bit in a string identified whether a particular training pattern was used. To reduce the number of training patterns used, a "bonus" system similar to the one described in section 4.2.2 was again used. The bonus was calculated according to how many patterns were not used. A string received a bonus of one point for each pattern that it did not use. There were two bonus policies, "only-the-best" and "bonus-above-the-threshold." With the "only-the-best" policy, bonus was only given to the training set with the highest classifier accuracy. This policy ensured that classifier accuracy was not sacrificed to reduce the number of training patterns. With the "bonus-above-the-threshold" policy, all training sets with classifier accuracy above a user-preset threshold received bonus points. The choice between these two policies depends on the tradeoff desired between classification accuracy and number of training patterns used. In general, the "bonus-above-the-threshold" policy results in a smaller number of stored exemplars, but does not provide the highest classification accuracy.

7.2.2 Using Genetic Algorithms to Select k

The number of training patterns also depends on the value k, the number of nearest neighbors that are polled. For example, if k is 1, then a testing pattern would be classified correctly as long as its nearest neighbor belongs in the correct class. On the other hand, if k is 7, then a testing pattern would be correctly classified only when a majority of its nearest seven neighbors are in the right class. As k increases, the need

for training data also increases. But there are cases when it's worthwhile to have a large k. It has been shown that with infinite amount of training patterns, accuracy increases as k increases[5].

There is thus a tradeoff between classifier accuracy and the number of training patterns required when the value k is chosen. For a given problem, there is generally no way of determining the proper value of k besides performing experiments. Since the choice of k affects both the number of training patterns required and classifier accuracy, the option of using genetic algorithms to select the value for k was added to the genetic search algorithm.

The k value was encoded with three bits, thus k could vary from 1 to 8. These three bits were attached to the end of each string and were separate from the other bits that identified the usage of the training patterns. The whole combined string was manipulated as before. Besides separate treatment in decoding, there was no differentiation between the k bits and the pattern status bits when the string was processed.

7.3 EXPERIMENTS

7.3.1 The Vowel Problem

Exemplar selection was tested using the vowel database described in Section 6.3.2. This database was chosen because it was a real problem with complicated boundaries between classes, it was thus more challenging to reduce the number of training patterns without sacrificing classification accuracy. Also, there were only 338 training patterns, the number of bits managed by genetic algorithms was therefore manageable.

The number of strings used in the population was set to 500. A uniform crossover operator was used with the mutative pressure value set to 0.25 and the selective pressure value set to 0.10. The uniform crossover operator performed quite well at this setting for the linear guessing problem (Table 2.10), so it was decided to try using it for this problem. Since each run of the program took about one day of computation time, it was impossible to perform a significant comparison between different settings of operators, selective pressure, and mutative pressure. Judging from the experience gained in performing initial exploratory experiments described in Chapter 2, it is unlikely for any setting to be an order of magnitude better than other settings.

Figure 7.1 shows the progress of genetic reduction of exemplars with $k = 1$ and "only-the-best" policy. After the initial evaluation of the 500 strings, the number of exemplars is already reduced to 160. Then, by recombination 20,000, the number of exemplars is reduced to 101. The training set classification error rate decreased to 12.3% while the testing set classification error rate stayed above 24.3%.

In previous studies[10] best recognition accuracy was provided when k was set to 8. Experiments were performed with k set to 8 as shown in Figure 7.2. As mentioned previously, a k nearest neighbor classifier with large k should require more data to perform well, so the number of exemplars probably can't be reduced as much. Figure 7.2 illustrates this point. At recombination 30,000, the number of exemplars was reduced to 147. This is 46 more exemplars than in the last experiment. The training set classification error rate was 14.5% while the testing set classification error rate was lower, 20.7%.

The other fitness policy, "bonus-above-the-threshold", was also tried. With this policy, once the performance above the threshold had been achieved, the goal of the genetic search became reducing the number of exemplars as much as possible.

A greater reduction in exemplars should be expected. Figure 7.3 illustrates the reduction of stored exemplars with the bonus above the threshold policy, k of 1, and the threshold of 80%. The number of stored exemplars was reduced to 43 by the end of the experiment. Using a higher k would result in a higher value, since more exemplars are needed to correctly identify each input pattern. As shown in Figure 7.4, such results were indeed obtained with $k = 8$. While the classification error rate on the training set fluctuated below 20%, the preset threshold, the number of exemplars was steadily reduced to 79.

Section 7.2.2 described the process of using genetic algorithms to select both k and the exemplars. Two experiments were performed using this approach, one with the "only-the-best" policy and the other with the "bonus-above-the-threshold" policy. Figure 7.5 shows the result of the experiment with the "only-the-best" bonus policy. At recombination 30,000, the number of exemplars was reduced to 146, k was selected to be 7, and the training and testing classification error rate were 14.5% and 21.3% respectively.

The number of exemplars was again greatly reduced by using the "bonus-above-the-threshold" policy with the threshold of 80%. Figure 7.6 illustrates the search. The number of exemplars was reduced to 63 while 6 was chosen for k. The training set classification error rate of 18.9% stayed barely below 20% as expected. The testing set classification error rate was 20.1%.

Figures 7.7 and 7.8 illustrate the difference in decision boundaries when the 43 stored exemplars chosen with genetic algorithms were used instead of all 338 exemplars. The decision boundaries in Figure 7.8 are smoother and generalize well with new data. On the other hand, using all 338 exemplars creates decision boundaries (Figure 7.7) that provided perfect performance on the training patterns but performed

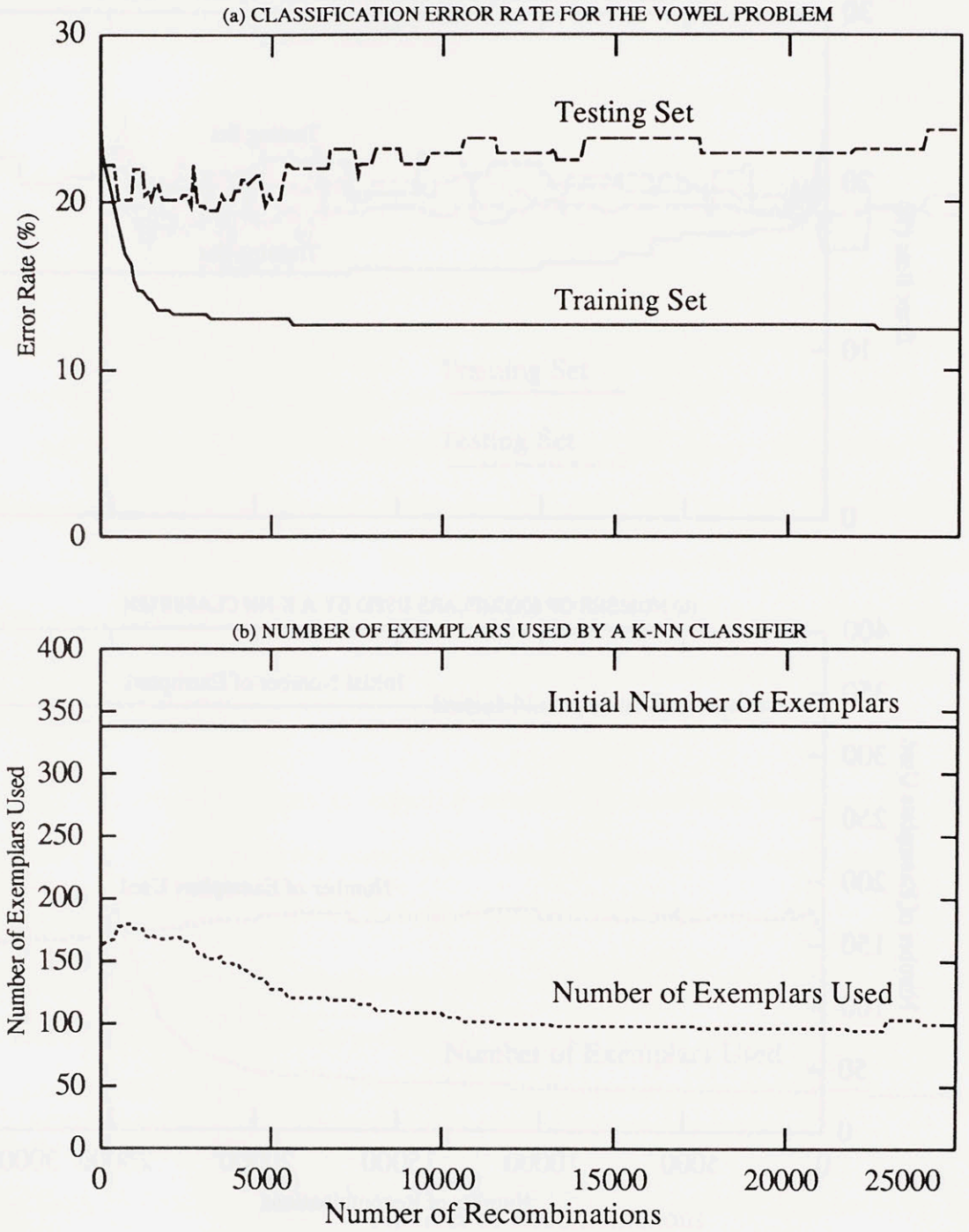

Figure 7.1: Progress of genetic reduction of exemplars for the vowel problem with $k = 1$ and "only-the-best" bonus policy, (a) classification error rate, and (b) the number of exemplars used.

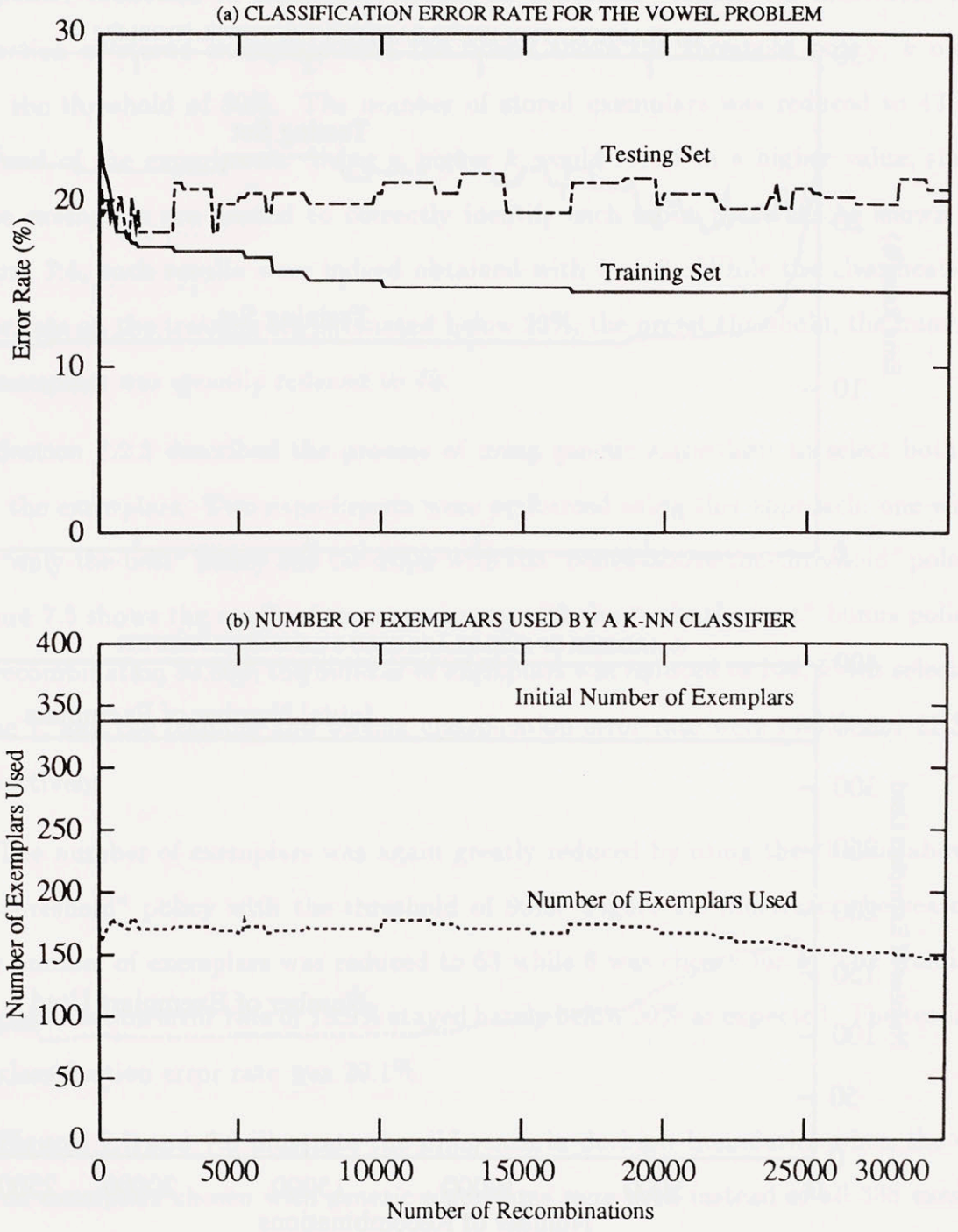

Figure 7.2: Progress of genetic reduction of exemplars for the vowel problem with $k = 8$ and "only-the-best" bonus policy, (a) classification error rate, and (b) the number of exemplars used.

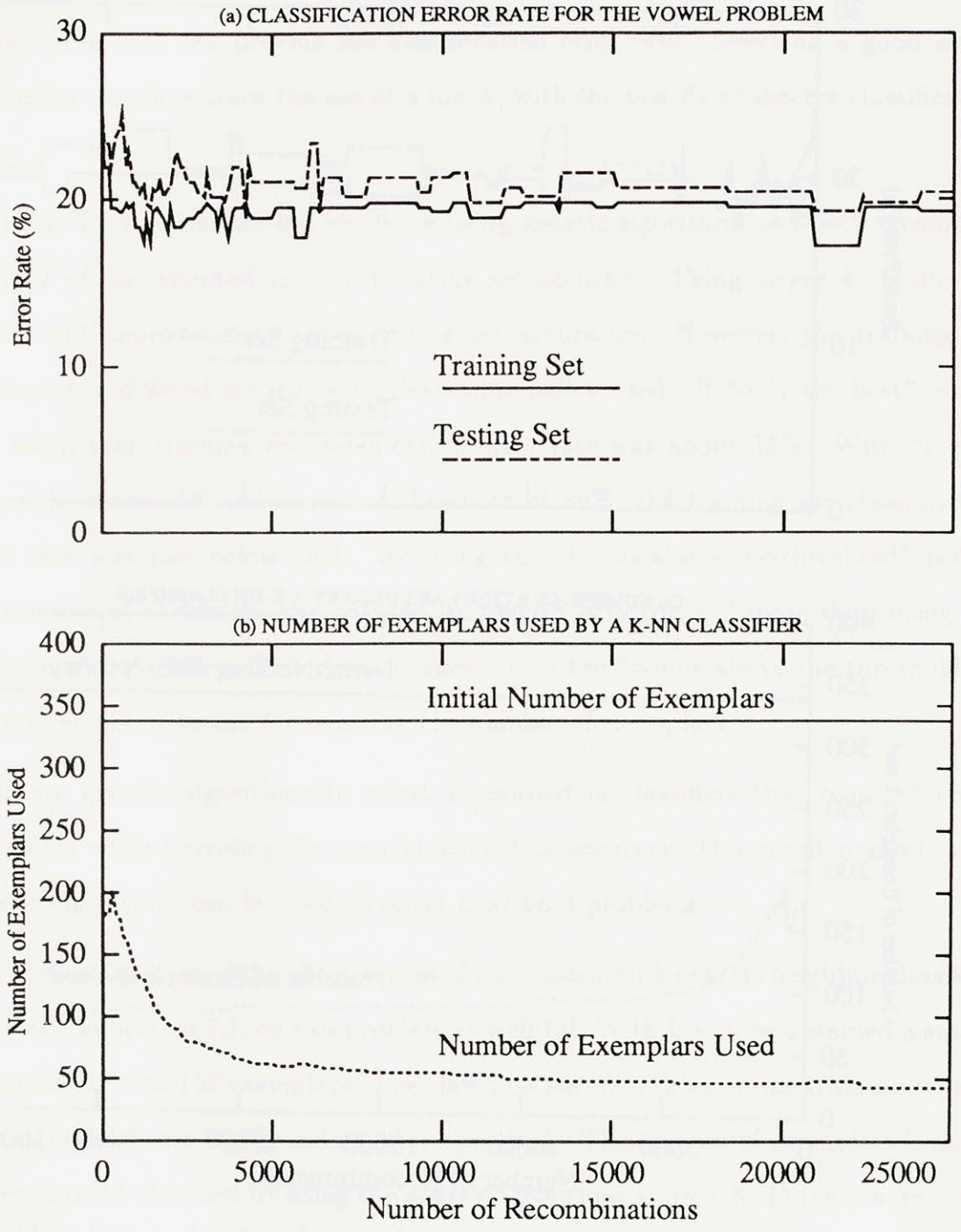

Figure 7.3: Progress of genetic reduction of exemplars for the vowel problem with $k = 1$ and "bonus-above-the-threshold" policy, (a) classification error rate, and (b) the number of exemplars used.

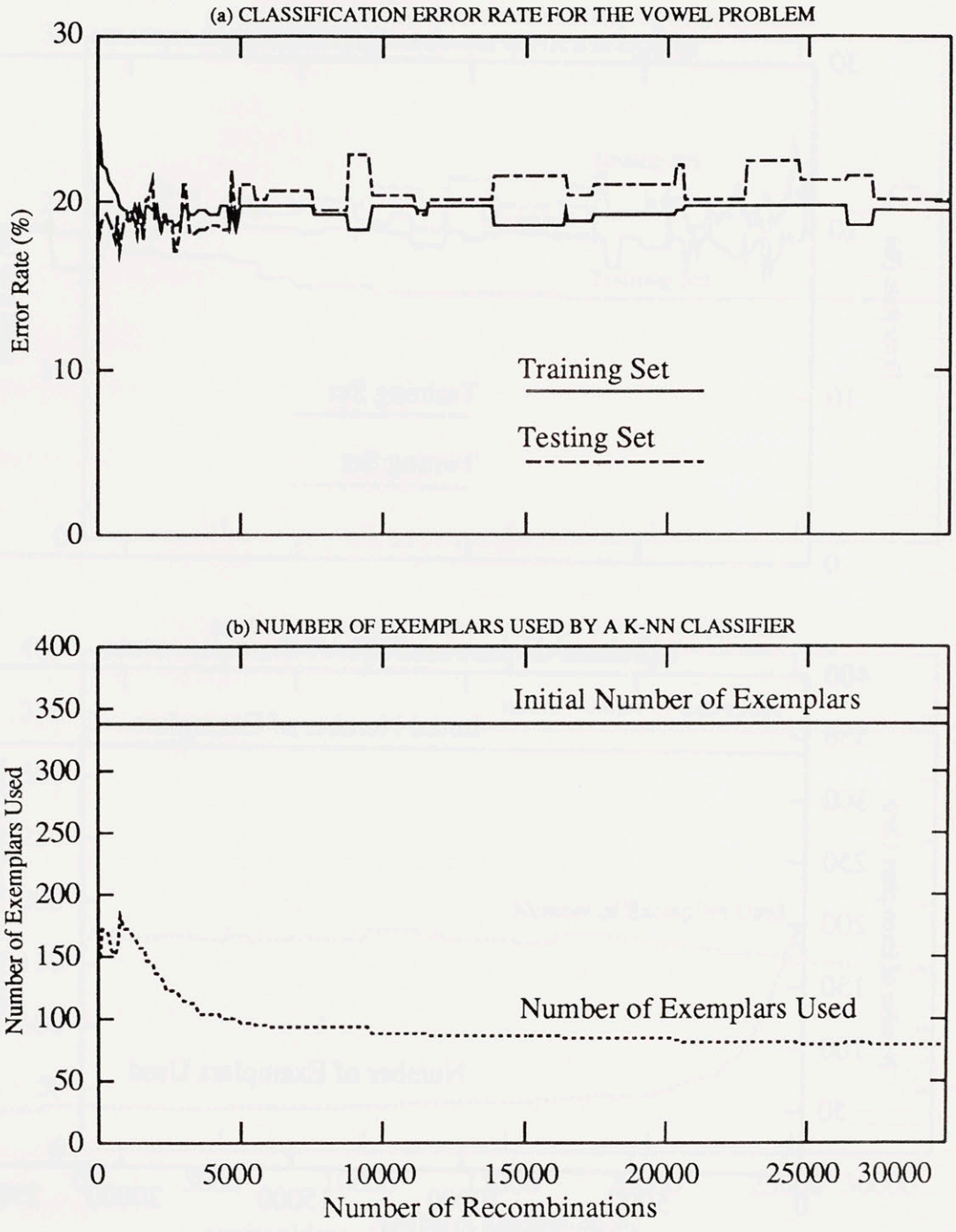

Figure 7.4: Progress of genetic reduction of exemplars for the vowel problem with $k = 8$ and "bonus-above-the-threshold" policy, (a) classification error rate, and (b) number of exemplars used.

poorly on the testing patterns (25%). By carefully choosing exemplars, even using a low value of k can provide low classification error rate. Selecting a good set of exemplars can thus allow the use of a low k, with the benefit of shorter classification time.

Table 7.1 summarizes the results of using genetic algorithms to select exemplars. Using k of one resulted in lowest testing set accuracy. Using larger k resulted in better and approximately equal testing set accuracies. However, the training set performance differed according to the bonus policy used. If "only-the-best" policy was used, then training set classification error rate was about 15%. With "bonus-above-the-threshold" policy and a threshold of 80%, the training set classification error rate was just below 20%. By using the "bonus-above-the-threshold" policy, the number of exemplars was reduced by roughly a factor of 2 more than using the "only-the-best" policy. This result suggests that the "bonus-above-the-threshold" is the proper policy to use for reducing the number of exemplars.

Using genetic algorithms to select k resulted in classifiers that required fewer exemplars while providing the same classification accuracy. This result suggests that genetic algorithms can be used to select k without problems.

Ng recently studied the effectiveness of the condensed k nearest neighbor classifier, described in Section 7.1, on this problem as well[13]. With $k = 8$, he obtained a stored exemplar set with 152 exemplars. The classification error rates on the training set and the testing set were 26.3% and 21.0% respectively. The number of exemplars is much larger than 43 obtained by using the genetic algorithms approach. Furthermore, since these 43 exemplars offered better classification performance while using only a k of 1, a k nearest neighbor classifier using these exemplars would require less time due to using both fewer exemplars and lower k value. The genetic approach however took

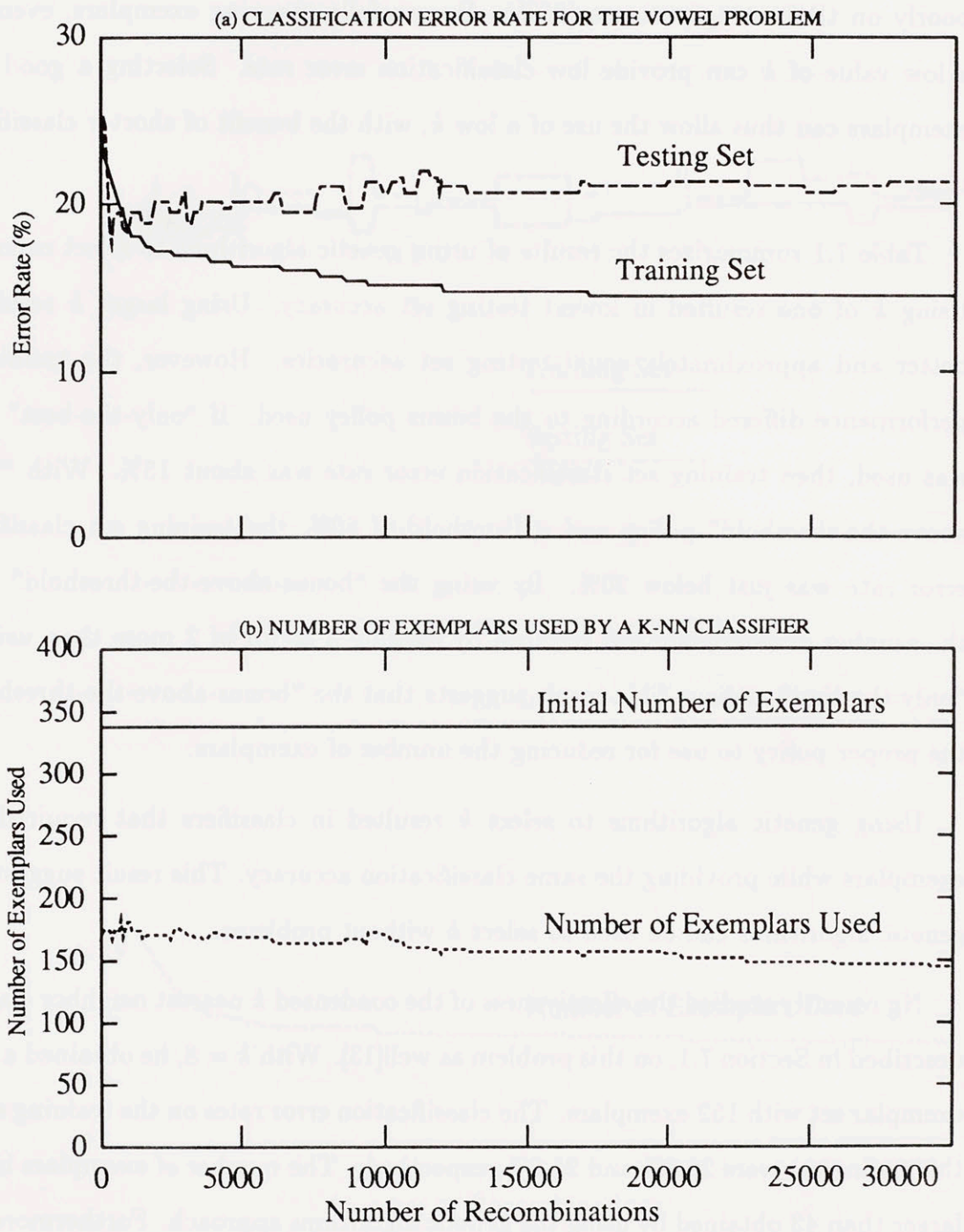

Figure 7.5: Progress of genetic reduction of exemplars for the vowel problem with k selected by genetic algorithms to be 7 and "only-the-best" bonus policy, (a) classification error rate, and (b) the number of exemplars used.

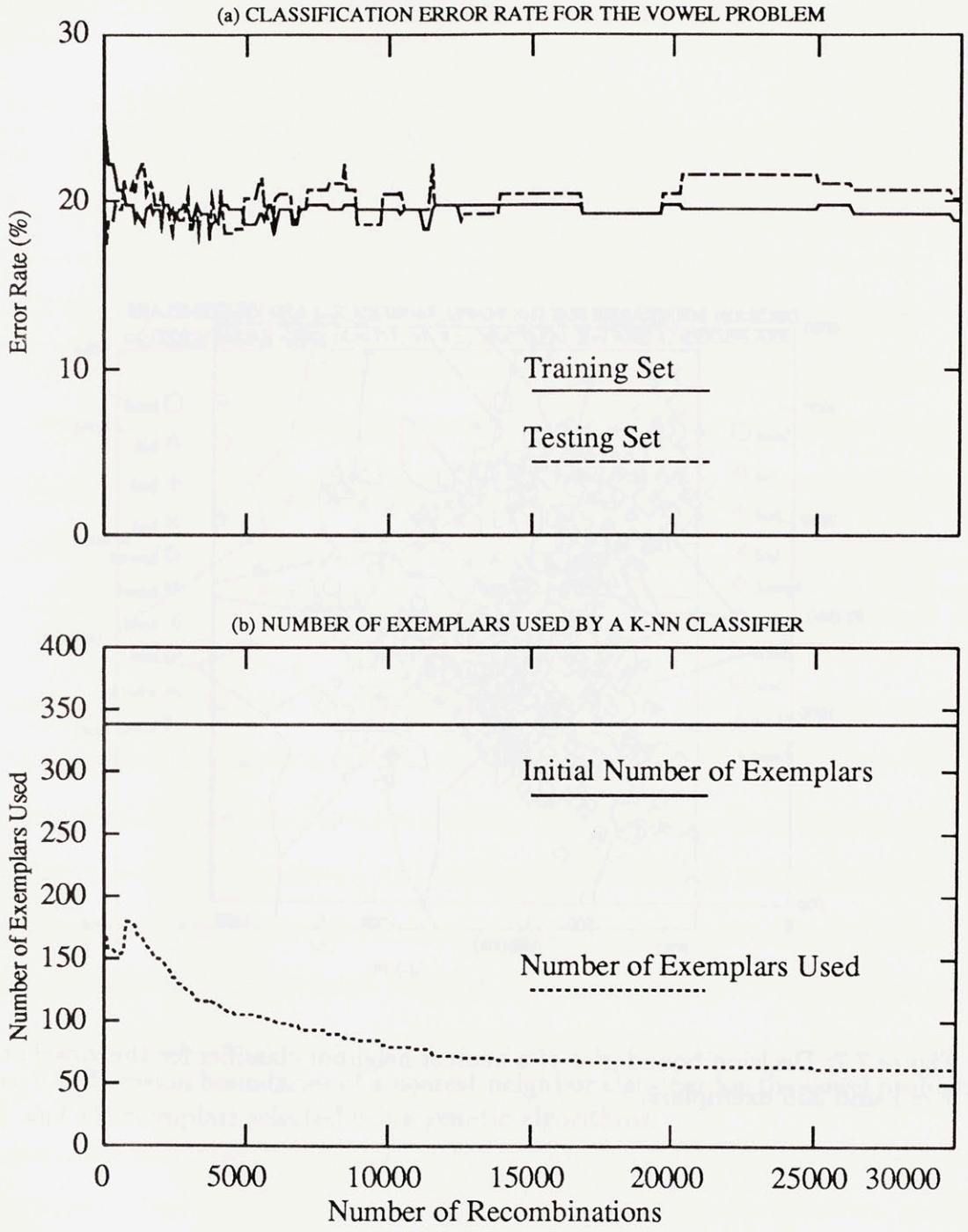

Figure 7.6: Progress of genetic reduction of exemplars for the vowel problem with k selected by genetic algorithms to be 6 and "bonus-above-the-threshold" policy, (a) classification error rate, and (b) the number of exemplars used.

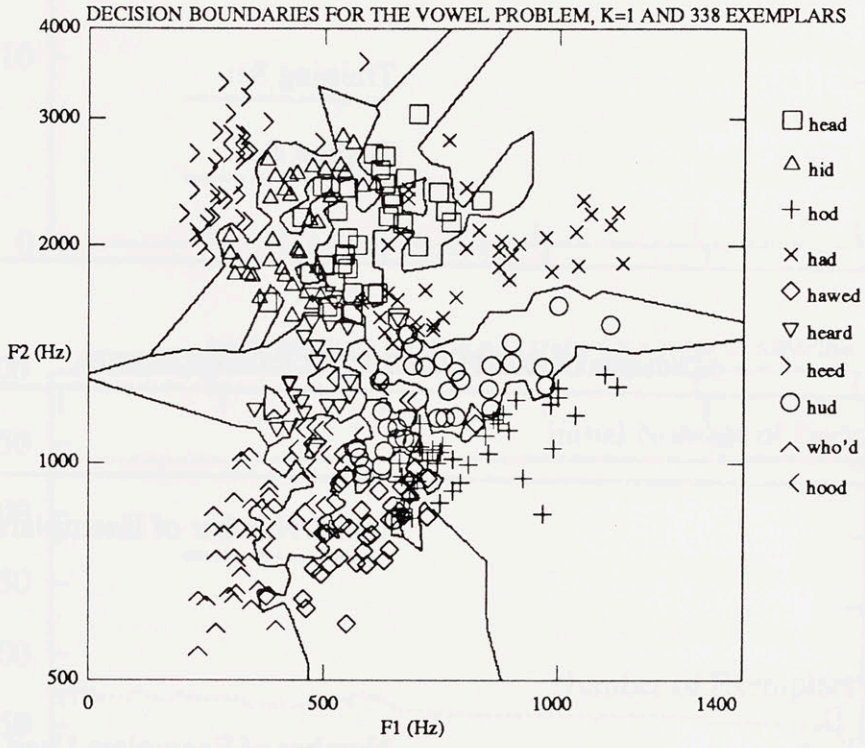

Figure 7.7: Decision boundaries of a nearest neighbor classifier for the vowel problem, $k = 1$ and 338 exemplars.

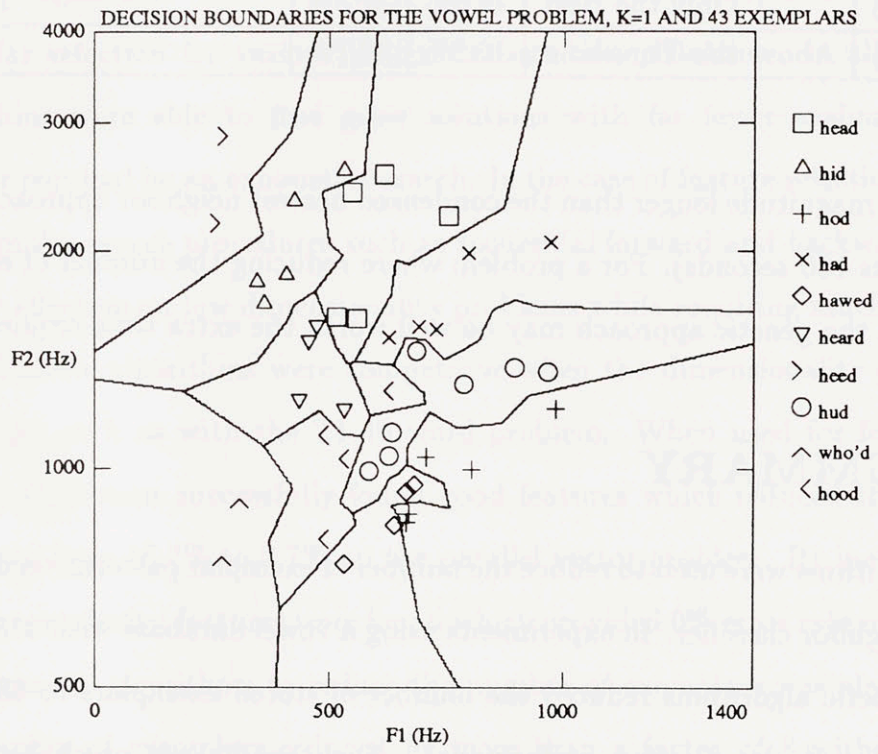

Figure 7.8: Decision boundaries of a nearest neighbor classifier for the vowel problem, $k = 1$ and 43 exemplars selected using genetic algorithms.

Table 7.1: Summary of Using Genetic Algorithms to Select Exemplars.

k	Bonus Policy	Error Rate		Number of Exemplars
		Train	Test	
1	Only the Best	12.4%	24.3%	101
1	Above the Threshold	19.2%	20.4%	43
8	Only the Best	14.5%	20.7%	147
8	Above the Threshold	19.5%	20.1%	79
7 (GA)	Only the Best	14.5%	21.3%	146
6 (GA)	Above the Threshold	18.9%	20.1%	63

two orders of magnitude longer than the condensed nearest neighbor approach (19,600 seconds versus 180 seconds). For a problem where reducing the number of exemplars is important, the genetic approach may be well worth the extra time required.

7.4 SUMMARY

Genetic algorithms were used to reduce the number of exemplar patterns required by a k nearest neighbor classifier. In experiments using a vowel database with 338 training patterns, genetic algorithms reduced the number of stored exemplars to 43 without significant loss of classification performance. Such results are much better than those obtained using the simpler condensed k nearest neighbor algorithms. Based on the success achieved with this problem, it is expected that genetic algorithms can efficiently reduce the number of training patterns for other types of problems.

Chapter 8

CONCLUSIONS

Genetic algorithms were applied to both feature selection, feature creation, and exemplar selection for various pattern recognition problems. In all cases, genetic algorithms were able to find good solutions with far fewer evaluations than the number required by an exhaustive search. In the case of feature selection, it was found that simpler search procedures such as sequential forward and backward search were equally effective on low dimensionality problems while requiring much less computation. Genetic algorithms were competitive when the dimensionality of the problem was large, such as with the TI 46 word problem. When used for feature creation, genetic algorithms successfully found good features which reduced the classification error rate from 18.7% to 2.7% on the parallel vector problem. By increasing created feature complexity, features were found which provided 0% error rate on this problem. Using genetic algorithms to reduce the number of exemplars was also fruitful, with the number of exemplars reduced by more than a factor of 8 without substantial increasing classification error rate for the vowel problem.

Although genetic algorithms proved useful for feature selection, feature creation, and exemplar reduction, this approach required long computation times. A computation time of days may not be acceptable for some applications. However, genetic algorithms can be easily adapted to parallel machines to reduce run times. Even single processor machines are becoming increasingly more powerful, making genetic algorithms more practical for feature selection and exemplar reduction. Also, ge-

netic algorithms proved to be a good search technique which is widely applicable in pattern classification. Compared to tailoring a special search technique for each type of searching problem, genetic algorithms offer the benefit of simplicity and good performance on all problems.

More research is needed in quantifying the relationship between the parameters of genetic algorithms and the problem size and problem type. Better measurements of population convergence and optimality are also needed. Lastly, more experiments need to be performed on other real problems to fully characterize the effectiveness of genetic algorithms and understand their performance compared to other search techniques.

Bibliography

[1] R. Bellman. *Adaptative Control Processes: A Guided Tour.* Princeton University Press, Princeton, NJ, 1961.

[2] M. Brady. Optimization strategies gleaned from biological evolution. *Nature*, vol. 317, 1985.

[3] C. Y. Chang. Dynamic programming as applied to feature selection in pattern recognition systems. *IEEE Trans. Systems, Man and Cybern.*, 3:166–171, 1973.

[4] T. M. Cover. The two best measurements are not the best two. *IEEE Trans. on System, Man, and Cybernetics*, SMC-4:116–117, 1974.

[5] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1973.

[6] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, New York, NY, 1972.

[7] A. S. Gevins and N. H. Morgan. Ignorance-based systems. In *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 39A.5.1 – 39A.5.4, New York, NY, April 1984.

[8] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, New York, 1989.

[9] William Y. Huang and Richard P. Lippmann. HMM speech recognition systems with neural net discrimination. In *proceedingss Neural Information Processing*

Systems - Natural and Synthetic Conference, Denver, CO, November 1989. IEEE. In Press.

[10] Yuchun Lee. Classifiers: Adaptive modules in pattern recognition systems. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1989.

[11] R. P. Lippmann. Pattern classification using neural networks. *IEEE Communications Magazine*, 27(11):47–54, November 1989.

[12] P. M. Narneda and K. Fukunaga. A branch and bound algorithm for feature subset selection. In *Proceedings Cybernetics and Society International Conference, Wash. D.C.*, Washington, DC, 1976.

[13] Kenneth Ng. A comparative study of the practical characteristics of neural network and conventional pattern classifiers. Master's thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1990.

[14] T. Parsons. *Voice and Speech Processing.* McGraw-Hill, New York, 1986.

[15] Gorden E. Peterson and Harold L. Barney. Control methods used in a study of vowels. *The Journal of the Acoustical Society of America*, 24(2):175–84, March 1952.

[16] S. Renals and R. Rohwer. Phoneme classification experiments using radial basis functions. In *Proceedings International Joint Conference on Neural Networks*, pages I.461–I.467, Washington DC, June 1989. IEEE.

[17] W. Siedlecki and J. Sklansky. On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2:197–220, 1988.

[18] W. Siedlecki and J. Sklansky. Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In *Proceedings Third International Conference on Genetic Algorithms*, Washington, DC, June 1989.

[19] W. W. Siedlecki. *Feature Selection for Large Scale Problems.* PhD thesis, University of California at Irvine, 1989.

[20] Manoel F. Tenorio and Wei-Tshi Lee. Self organizing neural networks for the identification problem. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 57–64. Morgan Kauffman, San Mateo, CA, 1989.

[21] Enrique Vidal, Hecor Rulot, Francisco Casacuberta, and Jose-Migues Benedi. On the use of a metric-space search algorithm (AESA) for fast dtw-based recognition of isolated words. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, ASSP 36-5:651–660, 1988.

[22] Darrel Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings Third International Conference on Genetic Algorithms*, Washington, DC, June 1989.

[23] Darrel Whitley. Optimizing neural networks using faster, more accurate genetic search. In *Proceedings Third International Conference on Genetic Algorithms*, Washington, DC, June 1989.