GENETIC ALGORITHM FOR FEATURE SELECTION AND WEIGHTING FOR OFF-LINE CHARACTER RECOGNITON

by

FATEN T. HUSSEIN
B.Sc. Cairo University, Egypt, 1995

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINNERING

We accept this thesis as conforming to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA, 2002
April 2002
© Faten Hussein, 2002

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Electrical & Computer Eng.

The University of British Columbia
Vancouver, Canada

Date 8 April, 2002

Abstract

Computer-based pattern recognition is a process that involves several sub-processes, including pre-processing, feature extraction, classification, and post-processing. This thesis is involved with feature selection and feature weighting processes. Feature extraction is the measurement of certain attributes of the target pattern. Classification utilizes the values of these attributes to assign a class to the input pattern. In our view, the selection and weighting of the right set of features is the hardest part of building a pattern recognition system. The ultimate aim of our research work is the automation of the process of feature selection and weighting, within the context of character/symbol recognition systems. Our chosen optimization method for feature selection and weighting is the genetic algorithm approach.

Feature weighting is the general case of feature selection, and hence it is expected to perform better than or at least the same as feature selection. The initial purpose of this study was to test the validity of this hypothesis within the context of character recognition systems and using genetic algorithms. However, our study shows that this is not true. We carried two sets of experimental studies. The first set compares the performance of Genetic Algorithm (GA)-based feature selection to GA-based feature weighting, under various circumstances. The second set of studies evaluates the performance of the better method (which turned out to be feature selection) in terms of optimal performance and time. The results of these studies also show that (a) in the presence of redundant or irrelevant features, feature set selection prior to classification is important for k-nearest neighbor classifiers; and (b) that GA is an effective method for feature selection and the performance obtained using genetic algorithms was comparable to that of exhaustive search. However, the scalability of GA to highly

dimensional problems, although far superior to that of exhaustive search, is still an open problem.

Table of Contents

List of Tables

List of Figures

Acronyms and Abbreviations

GA	**Genetic Algorithm**
FS	**Feature selection**
FW	**Feature Weighting**
GFS	**Genetic Feature Selection**
GFW	**Genetic Feature Weighting**
OCR	**Optical Character Recognition**
K-NN	**K-Nearest Neighbor**
1-NN	**1-Nearest Neighbor**
BAB	**Branch and Bound**
SFS	**Sequential Forward Selection**
SBS	**Sequential Backward Selection**
SFFS	**Sequential Forward Floating Selection**
SBFS	**Sequential Backward Floating Selection**
SA	**Simulated Annealing**
TS	**Tabu Search**

Acknowledgments

First, I would like to thank God, for his endless blessings and mercy, and without whom I can do nothing.

I would like to express my appreciation to my supervisor Dr. Nawwaf Kharma for his continuous guidance, advice and availability throughout this research work. He has provided me with valuable suggestions and technical assistance, which made this thesis possible.

I would like to express my gratitude to my supervisor Dr. Rabab Ward for her support and encouragement. I can't thank her enough for her invaluable feedback.

To my husband, Ayman, I thank you for your continuous love, encouragement, help and understanding. Without your support I wouldn't have been able to complete this work. I would like to thank my son Omar, for all the joy and happiness he has brought to my life.

My heart-felt gratitude goes to my parents for their love, support and never-ending compassion throughout my life, which I will never be able to repay.

Chapter 1 Introduction

1.1 Motivation

The main purpose of pattern recognition systems is to classify objects into one of a given number of class labels. Features are scalar properties that represent objects. Multiple features are combined together to form a feature vector. Any pattern recognition system includes two basic parts: feature extraction and classification (Wang & Fan, 1996). Feature extraction is the process of defining the most relevant features, which will minimize the within-class pattern variability and maximize the between-class pattern variability (Devijver & Kittler, 1982). As the number of features extracted increases, the cost and the running time of the recognition system increase as well. On the other hand, using fewer features could lead to failure of classification. These contradicting requirements have emphasized the need for well-balanced feature selection methods.

The goal of feature selection is to reduce the number of features extracted, by eliminating irrelevant and redundant features, while simultaneously maintaining or enhancing classification accuracy. Feature selection is sometimes performed in an ad-hoc way and accomplished by hand. A human designer implicitly selects the features that appear to him or her to be of most potential use. This process of manual feature selection usually depends on the experience of the designer in the domain knowledge and on trial-and-error. Another problem with the human expert method of feature selection is that humans cannot usually predict the existence of non-linear interactions between features.

The process of choosing the best features to represent the data is a difficult and time-consuming task. As a matter of fact, for handwritten character recognition problem we have

many variants of letter shape, size, and generally, style. Also, different writers have different writing styles. For the letters of an alphabet, there are nearly an unlimited number of variations. So many features must be used in a typical character recognition system to accommodate these variations. However, for a problem with a large number of features, it is not feasible to perform an exhaustive search to find all relevant features. Therefore, a computer algorithm must be developed to determine which features most accurately represent the pattern. This process of automatic feature selection will ensure that the recognition system is optimized.

The problem of feature selection, especially automated feature selection, has received a great deal of attention. Several search algorithms are used to solve the feature selection problem (Dash & Liu, 1997; Jain & Zongker, 1997). Among these, genetic algorithm has been revealed as a powerful search tool to select an optimal subset of features. Genetic Algorithm (GA) has been used for feature selection and weighing in many pattern recognition applications (e.g., texture classification and medical diagnostics). GA has proven to be an effective computational method, especially in situations where the search space is mathematically uncharacterized, not fully understood, or/and highly dimensional. Moreover, GA is domain independent (i.e. do not require derivative information or other auxiliary knowledge about the problem), and has been shown to be an excellent approach for solving combinatorial optimization problems. However, their use in feature selection (let alone weighting) in character recognition applications has been infrequent. This has inspired the work of this thesis, which is to apply GA for feature selection and weighting for character recognition applications. This work has resulted so far in (Hussein, Kharma & Ward, 2001; Kharma, Hussein & Ward, 2002a; Kharma, Hussein & Ward, 2002b) publications.

The main purpose for our work is to apply genetic algorithms for the problem of feature weighting for character recognition application. The work is motivated by the fact that there is no published work in the literature that has applied genetic feature weighting (GFW) in the context of character recognition problems. So it will be applied for the first time. After the employment of GFW to character recognition application, we are interested in comparing the performance of both genetic feature selection (GFS) and GFW also for character recognition applications. We are encouraged to perform such a comparison because it is often mentioned in the literature that feature weighting has the potential of working better than (or at least as well as) feature selection, when applied to the same situation (Komosinski & Krawiec, 2000; Punch, Goodman, Pei, Chia-Shun, Hovland & Enbody, 1993). However, this proposition was never fully and comprehensively assessed before. Only a single comparison exited in the literature (kohavi et. al. 1997), which compares the classification accuracy of feature selection (FS) and feature weighting (FW). However, the search method used in this comparison is not genetic algorithms, and the comparison was not employed for character recognition applications. Therefore, we intend to test the validity of this proposition. In addition, we need to carry out an inclusive comparison between GFS and GFW for character recognition applications. This comparison between GFS and GFW will tackle several issues. These issues include the number of eliminated features by both methods, their performance in situation where irrelevant/redundant features exist and the classification accuracies of both methods in regular situations. Moreover, we aim to test the performance of the better method (which turns out to be GFS) for both optimality and time complexity.

1.2 Thesis Contributions

Basically, the main contributions of the thesis are:

- Developed a GA system to be used to configure the real-valued weights of a k-nearest neighbor (k-NN) classifier component of a character recognition system. This genetic feature weighting system GFW will be applied, for the first time, to off-line recognition of isolated handwritten digits.
- A comparison of GFW with, the previously applied methods by other researchers for character recognition systems, GFS (genetic feature selection) under various circumstances.
- An evaluation for the performance of the better method in terms of optimal performance and time.

In addition to the abovementioned direct contributions, we believe that this research work is important for the field of character recognition, because it provides the following:

- Investigates the pragmatic aspects (in terms of the number of eliminated features and the performance under situations where irrelevant/redundant features exist) for the automatic feature selection and weighting using genetic algorithm for character recognition systems.
- Provides testable hypothesis (i.e. feature weighting has the potential of working better than feature selection) and formal explanations for the behavior of the GA-based feature selection and weighting in character recognition applications.
- Represents an empirically proven method (GFS) for the feasibility of a search for an optimal set of features (of a moderate size) for enhancing the recognition rates in

character recognition systems while suggesting a possible use of distributed computational implementations for highly dimensional problems.

- Offers a way for building a semi automatic optimization method of a general hand-written symbol recognition system, in which the human intervention is needed to provide new libraries for feature extraction and classification functions.

1.3 Thesis Structure

The thesis consists of seven chapters. The first chapter is an introduction. Chapter two presents an overview of pattern recognition and character recognition methods and introduces the k-nearest neighbor classifier. In chapter 3, we present the feature selection and weighting problem and describe the various algorithms used in feature selection. Chapter 4 introduces genetic algorithm and its various parameters. It also compares genetic feature selection with other feature selection methods and surveys genetic feature selection in character recognition applications. Chapter 5 presents the details of the developed system, the platform and the methodologies used. In addition, it provides a comparative study between genetic feature selection and weighting, describes the experimental work and explains the results obtained from these experiments. Chapter 6 presents an evaluation for genetic feature selection in terms of optimality and time and shows an experimental verification for some of the results obtained form the comparative study and evaluation. Finally Chapter 7 offers the conclusions and suggestions for future work.

Chapter 2 Pattern Recognition

The goal of pattern recognition is to classify objects into a number of classes. These objects, depending on the application, can be images or signal waveforms or anything that needs to be classified. Pattern recognition applications range from automated speech recognition, optical character recognition to fingerprint identification and so on.

2.1 Character Recognition

Character Recognition or Optical Character Recognition (OCR) is the process of converting scanned images of machine printed or handwritten text (numerals, letters, and symbols), to computer-processed format (such as ASCII). The popularity of OCR has been increasing each year with the advent of fast microprocessors providing the vehicle for vastly improved recognition techniques. There is a wide variety of OCR systems in use today, from automatic postal address readers through massive document handling computers used by offices, to the desktop systems that employ scanners for reading text into word processing and spreadsheet applications.

In general, there are two main categories for the character recognition problem, on-line and off-line. The purpose of on-line handwritten recognition is to recognize the symbols while they are being written. On the other hand, in the off-line case, the recognition process is performed after the symbols have already been written. OCR belongs to the off-line recognition category. In our research work we are interested in the case of off-line recognition of isolated handwritten digits.

2.1.1 Components of Character Recognition System

An OCR system typically involve the following processing steps (Kharma & Ward, 1999):

1- Pre-Processing.

2- Feature Extraction.

3- Pattern Classification.

4- Post-Processing.

The pre-processing step aims to improve the image data by suppressing unwanted distortions or enhancing some image features important for further processing. So the output of the pre-processing step is a cleaned up version of the original image, which can be used into the next step. Examples of pre-processing functions are: Noise removal, skeletonization, thinning, normalization and segmentation.

Feature extraction is an important step in achieving good performance of OCR systems. It is the process of defining the most relevant features, which will minimize the within-class pattern variability and maximize the between-class pattern variability (Devijver & Kittler, 1982). Several feature extraction methods exist; for an extensive survey, see (Trier, Jain & Taxt, 1996). The goal is to find those features that are of possible relevance for classification.

Assume now that a list of measured features is provided. The portion of the process that must map these input features onto classes is called the classifier. This step can be accomplished by means of a number of algorithms, including clustering techniques, rule-based systems, neural net works and decision trees (Kharma & Ward, 1999).

Traditional OCR performs post-processing to assist in the resolution of errors produced in the character recognition processes. The goal is to increase the level of confidence in the

classification results. One of the post-processing methods used is a word dictionary to verify word results. Alternatively, recognition results may be verified interactively with the user.

2.2 Pattern Recognition Approaches

There are three approaches for pattern recognition: the statistical, the structural and the neural network approaches (Pandya & Macy, 1996). In the statistical approach, the input pattern is represented by a number of attributes or features (e.g. a set of measurements performed on the raw data). If a suitable set of features is chosen to properly represent the patterns, feature vectors having the same class will be close to each other while feature vectors belonging to different classes will be positioned in different regions of the feature space. In this way, the recognition task is reduced to partitioning the feature space into regions of different classes. In our work, we have used the statistical approach for the task of recognizing handwritten digits. The patterns, which are images of handwritten digits, are represented by d dimensional feature vector.

On the other hand, in the structural approach, a pattern is assumed to be decomposed into simpler sub-patterns, which in turn can also be decomposed into simpler sub-patterns (called primitives) in a recursive way (Jain, Duin & Mao, 2000). In syntactic pattern recognition, which is a sub-set of structural pattern recognition, patterns of a class are viewed as sentences in a language defined by means of grammar, and primitives are considered the alphabet of the language. Each class has its own defined set of rules or what is called grammar. The grammar specifies the way in which sub-patterns can be combined to form a valid pattern for a specific class. The pattern recognition problem in this case is to determine whether a given pattern belongs to the language generated by that grammar (Devijver & Kittler, 1982).

Neural networks (NN), in general, are large number of highly interconnected processing elements (nodes) that usually operate in parallel. The collective behavior of a NN, demonstrates the ability to learn, recall and generalize from training patterns. The analogy between neural networks and human brain made neural networks good candidates for pattern classification problems. In fact, neural networks are a non-parametric type of classifiers with predictive capabilities. They make no assumptions about the underlying distributions. Another main advantage of neural networks is their capability to learn complex nonlinear relationships between inputs and outputs (Jain et al. 2000). Moreover, they have the ability to generalize and even to recognize partially degraded patterns (Pandya & Macy, 1996).

2.3 Supervised verses Unsupervised Learning

Pattern recognition can be classified into two broad categories: supervised and unsupervised (Pandya & Macy, 1996). A supervised learning process is one in which the user provides some external information about the problem. That is, a set of examples, or training set of classified elements. In the unsupervised case, no prior information is provided and the system is required to discover the fundamental structure of the data on its own. Correct classes are not available, and grouping or clustering is used in order to infer correct classification from the data elements.

2.4 Parametric verses Non-parametric

In statistical pattern recognition, there are two general ways to design a classifier, parametric and non-parametric. The key distinguishing feature is the form of the information "learned" during training and passed on to the classifier. Parametric classifiers assume that the patterns in the training set fit a known statistical distribution. These classifiers are parametric, in that

they are specified in terms of parameters (mean and covariance) of class distributions. Nonparametric classifiers are useful in cases where the underlying distribution cannot be easily parameterized. In such cases, we can either estimate the density function or directly construct the decision boundary from the training data (e.g. k-nearest neighbor) (Jain et al. 2000).

2.5 The Nearest Neighbor Rule

Nearest neighbor classification is a nonparametric method that does not assume that the patterns to be classified have known density functions. The K Nearest Neighbor Rule (k-NNR) is a very intuitive method that classifies unlabeled samples based on their similarity to classified samples in the training set (Dasarathy, 1991). The algorithms for the nearest neighbor rule is described as follows:

- Given a feature vector x, whose class is to be determined, and n training samples, identify the k training samples that form the nearest neighbors to x, regardless of the type of the class (where k is chosen to be an odd number).
- Out of these k samples, determine the number of vectors k_i, which belongs to class w_i (where $i=1,2,\ldots, m$ and $\sum_i k_i = k$ and m is the number of classes).
- Assign the unknown vector x to the class w_i with the maximum number of k_i samples.

In other words, the k nearest neighbor method assigns to an unclassified sample x the class most heavily represented among its k nearest neighbors. When k=1, in this case it is known as nearest neighbor rule, the feature vector x is assigned to the class of its nearest

neighbor. The k- NNR is very simple; it only requires an integer k, a set of labeled examples (training data) and a metric to measure the "closeness".

K-NNR is considered an instance based learning algorithm. Instance based learning algorithms are a class of supervised machine learning algorithms. These algorithms do not construct abstract concepts, but rather base their classification of new instances on their similarity to specific training instances (Aha, 1992). Old training instances are stored in memory, and classification is postponed until new instances are received by the classifier. When a new instance is received, older instances are retrieved from memory and used to classify the new instance. Other names for instance based algorithms are: Memory- based, Exemplar- based or Case- based.

Instance based learning algorithms have the advantages of being able to (a) learn complex target concepts (e.g. functions); and (b) estimate target concepts distinctly for each new instance. In addition, their training is very fast and simple; it only requires storing all the training instances in memory. In contrast, the cost of classifying new instances can be high because every new instance is compared to every training instance. Hence, efficient indexing of training instances is important. Another disadvantage of these learning algorithms is that their classification accuracy degrades significantly in the presence of noise (in training instances).

In the k-nearest neighbor method, the similarity between two cases can be measured in various ways. The most common similarity measure is based on Euclidean distance. This is described as follows:

$$D(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad \text{(Eq.1)}$$

Where D is distance, x and y are two instances, x_i and y_i are the i-th attribute for the x and y instances, and n is the total number of features. To compensate for the difference in units between features, normalization should be performed. This often scales all features to a range between 0 and 1, inclusive.

2.6 Attribute Weighted K-Nearest Neighbor

One major drawback of the Euclidean distance function is its sensitivity to the presence of noise, and particularly, redundant or irrelevant features. This is because it treats all features of an instance (relevant or not) as equally important to its successful classification. A possible remedy is to assign weights to features. The weights can then be used to reflect the relative relevance of their respective features to correct classification. Highly relevant features would be assigned high weights relative to the weights of redundant or irrelevant features. Taking that into account, the Euclidean distance measure can be now refined to:

$$D(x,y) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2} \quad \text{(Eq.2)}$$

Where w_i is the weight of the i-th feature.

Assume (see Figure 1) there are several instances belonging to two classes, class 1 and class 2, each having two features (attributes) represented in 2-dimensional space. It is required to classify the unknown instance to either one of these two classes. In the left side of Figure 1, the unknown instance is classified according to the majority class of its k nearest neighbors (k=3, in this case). Thus the unknown instance is incorrectly classified to be belonging to class 1. However, the right side of Figure 1 shows the effect of using the attribute weighted k-nearest neighbor. This weighting is achieved by multiplying the Y-axis by a small weight, which corresponds to decreasing the effect of attribute Y and increasing

the influence of attribute X by multiplying it with a large weight. As a result of the attribute weighting, the unknown instance is correctly classified as belonging to class 2. Note that the dimension in the feature space, where a high weight is assigned, is extended. On the other hand, the dimension in the feature space, where a low weigh is assigned, is compressed.

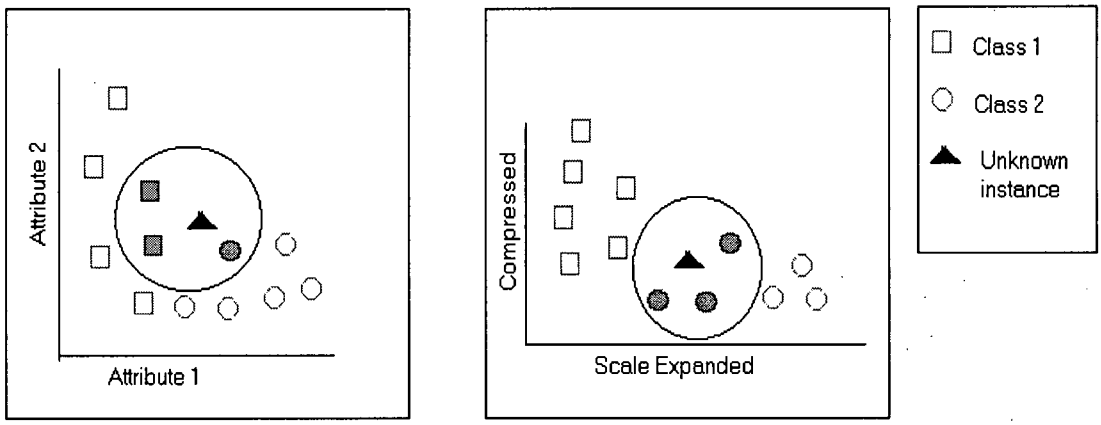


Figure 1: Example of the effect of feature weighting.

2.7 Data Normalization

In addition to feature weighting, it is also important to apply data normalization before classification. Different features have different measurement units, which means that their values lie within different ranges. So, using the Euclidean distance function with features having different ranges of values will result in a significant problem; features having large values will have larger effect on the classification than those features that have small values. However, this does not necessarily reflect their relative importance for classification (Theodoridis & Koutroumbas, 1998). Therefore, data normalization must be performed first to overcome the differences in units between feature values. A common method used for normalization is to restrict all feature values in a certain range e.g. [0,1] or [0,10] or any other

range. This way of scaling features guarantees that all features will be normalized to the same range of values.

2.8 Estimating Classification Error Rate

The assessment of a classifier, such as the k-NN, is based on its ability to successfully classify and predict the unseen data. The most commonly method used to measure the performance of a classifier is its error rate (Weiss & Kulikowski, 1991). Error rate is the number of incorrectly classified data samples divided by the total number of all samples. In general, the estimation of the error rate of a recognition system is performed by dividing all the samples available into two sets, a training set and a testing set. The classifier is built and designed using the training set, and then the error rate of the classifier is calculated using the testing set. However, both the training and testing sets should be independent and have a large number of samples so as to return a true measure of the classifier error rate (Jain et al. 2000). There are many methods for estimating classification error rate. These methods mainly differ in the way the available samples are split into training and testing sets. These methods are described as follows.

2.8.1 Hold Out Method

In this method, all the available samples are separated into two sets, called the training set and the testing set. The common splits used for the data samples are 2/3 of the data assigned to training and 1/3 to testing (Weiss & Kulikowski, 1991). Alternatively, half of the data can be used for training and the second half for testing. The classifier is trained using the the training set only. Then the trained classifier is asked to predict the output values of the testing set. The accumulated errors using the testing set is used as the classifier error rate. The

problem with this mehtod is that its evaluation can have a high variance. Since the holdout method is a single train-test method, its evaluation of error rate depends heavily on which data points end up in the training set and which end up in the test set. Thus the evaluation may be significantly different depending on how the division is made (Jain et al. 2000).

2.8.2 Cross-Validation Methods

The limitations of the holdout can be overcome with a family of cross-validation methods at the expense of higher computational cost. These methods are: the k-fold, the random sub-sampling and the leave-one-out methods (Weiss & Kulikowski, 1991; Jain et al. 2000).

2.8.2.1 K-Fold Cross-Validation Method

In this method, the data set is divided into *k* parts (folds), and the holdout method is repeated *k* times. Each time, one of the *k* subsets is used as the testing set and the other *k-1* subsets are combined to form a training set. The error rate is the average of the error rates obtained from all *k* trials. Every data point gets to be in a test set exactly once, and gets to be in a training set *k-1* times. The variance of the resulting estimate is reduced as *k* is increased. The disadvantage of this method is that the training algorithm has to be rerun *k* times, which means it takes *k* times as much computation to make an error estimation.

2.8.2.2 Random Sub-Sampling Method

Random sub-sampling can be viewed as a variant of k-fold cross-validation method, where data is randomly divided into a test and training set *k* different times. In this method, multiple random train-test experiments are performed *k* times. The train-test sets are chosen randomly each time. Then the classifier is built using the training set, and tested using the testing set. The error rate is the average of the error rates obtained from *k* runs. Random sub-sampling

has better error rates than the single train-test holdout method (Weiss & Kulikowski, 1991). The advantage of this method over the k-fold method is that you can independently choose how large each test set is and how many trials you average over. On the other hand, in the k-fold method all the examples in the dataset are eventually used for both training and testing.

2.8.2.3 Leave-One-Out Cross-Validation Method

It is a k-fold cross validation taken to its extreme, with k equal to N, the number of data samples. For a dataset with N examples, perform N runs. For each run, use $N-1$ examples for training and the remaining example for testing. As before the error rate is the average of the error rates obtained from N runs. Leave-one-out method has unbiased estimation for the error rate but it has large computational requirements. For very sparse datasets, we may have to use leave- one- out in order to train on as many examples as possible. Conversely, for large data sets, leave-one-out is computationally expensive, so random sub-sampling or k-fold methods are preferred.

2.8.3 Bootstrap Method

The bootstrap is a re-sampling technique with replacement. Given a dataset with N examples, N examples are randomly selected (with replacement) and this set is used for training. The remaining examples that were not selected for training are used for testing. This process is repeated for a specified number of folds (K). The error rate is the average of the error rates obtained from k folds. Obviously, the number of test examples is likely to change from fold to fold. Usually bootstrap method is used for small sample datasets (Jain et al. 2000).

Chapter 3 Feature Selection and Weighting

3.1 The Curse of Dimensionality

Intuitively, one would expect that the more information that is available, the better we can make decisions. That is, the more features available to the classifier, the better the classification results. However, in practice this is not always true. In fact, adding more features is not always helpful. For a given size of training samples, as the feature dimension increases, the number of data points becomes more sparse relative to the problem dimension. In addition, new features may not add useful information and some features may be noise. This phenomenon, which is often observed in pattern recognition, is called the peaking phenomena (Jain et al. 2000). Peaking phenomena happens when adding new features to a feature set leads to a decrease in the classification accuracy of a classifier trained on a finite set of training samples.

In general, the classifier performance depends on the relationship between the sample sizes and the number of features (Jain et al. 2000). For a given problem with d dimensional features, there exists some minimum number of training samples that are required by the classifier to achieve good classification rate. However, the required number of training samples grows exponentially with the dimensionality of the feature space (Pandya & Macy, 1996). This is known as the "curse of dimensionality". In practice, the curse of dimensionality means that, for a given sample size, there is a maximum number of features above which the performance of the classifier will degrade rather than improve. It is very difficult to draw the exact relation between the probability of misclassification, the sample

size and the number of features. However, a general accepted rule is to employ a number of training samples, which is at least ten times as the number of features used (Jain et al. 2000).

3.2 Dimensionality Reduction

As mentioned in the previous section, a major problem associated with the pattern recognition problem is the so-called curse of dimensionality. Usually, there is a very large number of features that a domain expert can provide when designing any pattern recognition problem in general, or character recognition system in particular. This number can easily range from a few dozens to hundreds. Thus, features must be evaluated and the most effective ones chosen. This process is referred to as the dimensionality reduction. Several reasons have motivated the use of dimensionality reduction techniques. The reduction of the number of features will certainly help in reducing/eliminating the curse of dimensionality problem. Moreover, reducing the dimensionality of the problem will, in turn, reduce the time complexity and the memory requirements of the recognition system. In addition, reducing the dimensionality will increase classifier efficiency by eliminating redundant and irrelevant features.

3.3 Feature Selection verses Feature Extraction

Two approaches are available to perform dimensionality reduction: Feature selection and feature extraction. Feature selection can be defined as follows. Given a number of features, the feature selection process aims to select the most important features of them so as to reduce their number and at the same time retain as much as possible of their discriminatory power. Feature selection can be modeled as follows: Given a feature set $X=\{x_j \mid j=1,2,..D\}$

of D features, a feature selection method should find a subset $Y=\{y_i \mid i=1,2,..d\}$ of d features, where $d<D$, such that the combination of d features optimizes a criterion function J, usually but not necessarily, the error rate of the classifiers (Devijver & Kittler, 1982).

On the other hand, feature extraction refers to the process of creating a subset of new features based on combinations or transformations of the existing features. The problem of feature extraction can be stated as follows: Given a feature set $X=\{x_j \mid j=1,2,..n\}$ of n features, a feature extraction method should find a mapping (either linear or non-linear) $Y=f(X)$, where $Y=\{y_i \mid i=1,2,..m\}$ and $m<n$, such that the transformed feature vector Y preserves (most of) the information or structure in X. An optimal mapping will be one whose set of features results in no increase in the classification error. The selection of the feature extraction mapping $Y=f(X)$ is guided by an objective function that we seek to maximize (or minimize). Examples of feature extraction methods are: principle component analysis, linear discriminant analysis and feature clustering.

The choice between feature selection and feature extraction depends on the type of application and data available (Jain et al. 2000). Feature subset selection is necessary in a number of situations. Feature selection decreases the cost of feature measurement by removing some features from the original set of features. The original features are important to keep so as to extract meaningful rules from classifier. However in feature extraction, where features are transformed or projected, the physical meaning of the features after such transformation are lost. In addition, features may not be numeric (a typical situation in the machine learning domain). In this case feature selection is the only applicable way for dimensionality reduction.

3.4 Feature Selection verses Feature Weighting

As we pointed out before, the k- NNR is very sensitive to noisy features. A solution to this problem is to modify the Euclidean metric by a set of weights that represent relevance of each feature. In feature weighting, the weights can hold any value from a continuous range of values (e.g. [0,1]). The purpose of feature weighting is to find a vector of real-valued weights that would reduce the damaging effect of irrelevant and redundant features while fine-tuning the weights of useful features to achieve higher classification accuracy.

Several feature weighting methods for case based learning algorithms exist. For example, the weighting could be global, meaning that there is a single weight vector for the classification task, or it could be local, in which weights vary over local regions of the instance space (Howe & Cardie, 1997). Moreover, the weight vector could have continuous real values or binary (zero or one only) values. In addition, the method for assigning the weights could be guided by the classifier performance or not (will be explained in section 3.5.1). For an extensive review, Wettschereck, Aha & Mohri (1997) provide a five-dimensional framework that categorizes different feature weighting methods.

On the other hand, feature selection aims at reducing the number of features used in classification, while maintaining or improving the classification accuracy (Dash & Liu, 1997). This entails that weights can either equal '0' for 'not selected', or '1' for 'selected'. Though both feature selection and weighting seek to enhance classification accuracy, only feature selection has the (real) potential of reducing problem dimensionality (by assigning '0' weights to features). This is contrary to feature weighting, where irrelevant/redundant features are almost always assigned small (but sometimes non-zero) weights. Feature

selection can also enhance classification accuracy as a result of completely eliminating highly irrelevant and redundant features.

Nevertheless, feature selection may be regarded as a special case of feature weighting. The trick is to find a way to use feature weighting to both (a) assign weights to relevant features, in a way that reflects their relative relevance, and (b) completely eliminate highly irrelevant and redundant features from the original set of candidate features.

3.5 Feature Selection Algorithms

Ideally, feature selection methods search through the subsets of n features, and try to find the best one among the possible 2^n candidate subsets according to some evaluation function. For example, for a problem using only 40 features, the number of possible subsets of the full feature set is $2^{40} \cong 10^{12}$. In general, feature selection attempts to select the minimally sized subset of features where the classification accuracy does not significantly decrease. Therefore, feature selection procedure is exhaustive as it tries to find only the best subset. It may be too costly and practically prohibitive, even for a medium-sized feature set size n. An exact searching algorithm through all subsets has an exponential complexity. Therefore an efficient search algorithm is required to explore the space of all possible feature subsets.

Several search techniques for feature selection have been proposed in the literature. Dash & Liu (1997) have categorized different feature selection methods according to the search technique used and the evaluation function. For extensive reviews of feature selection see Refs. (Blum & Langley, 1997; Jain & Zongker, 1997; Dash & Liu, 1997). In general, a feature selection method requires two things:

1- An objective function to evaluate candidate feature subsets.

2- A search strategy to select these candidate subsets.

3.5.1 Feature Selection Objective Function

The objective function evaluates candidate subsets and returns a measure of their "goodness", a feedback that is used by the search strategy to select new candidates. Objective functions are divided into two groups, which in turns divide the feature selection into two approaches: Filter and Wrapper. In the filter approach, the objective function evaluates feature subsets by their information content, typically by distance measures or dependence measure. For the distance measures, filter methods use distance to determine class separability, such as Euclidean distance to measure distance between classes. The dependence measure is based on the rationale that good feature subsets contain features that are highly correlated with the class, yet uncorrelated with each other. So it measures the correlation coefficient between features and class label and also between the features themselves. For both the distance and the dependence measures of the filter approach, feature selection does not depend on the classification algorithm.

On the other hand, in the wrapper approach, the classification algorithm is used as a part of the evaluation function of the feature subset. In other words, the search algorithm employs the classifier's predictive accuracy to evaluate the subset of features.

It is clear that the filter method requires less computational time when compared to the wrapper one; however, the filter approach ignores the effect of the selected features on the performance of the classification algorithm as opposed to the wrapper method, which could lead to better performance. Generally, wrappers achieve better recognition rates than filters since they are tuned to the specific interactions between the classifier and the dataset (Kohavi & John, 1997). Figure 2 depicts both the filter and wrapper approaches.

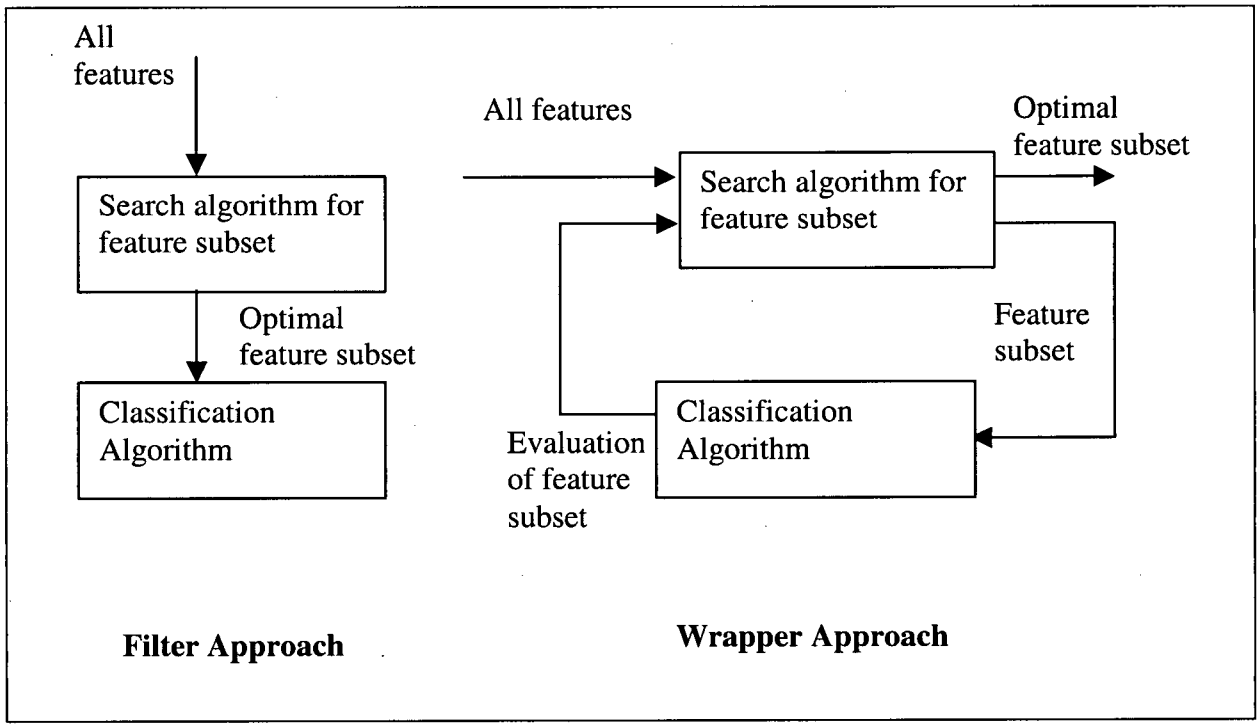


Figure 2: The two feature selection approaches, filter and wrapper.

3.5.2 Feature Selection Search Strategy

Before evaluating the goodness of the candidate feature subsets, a search strategy is required to select these candidate subsets. The search strategy is required to direct the feature selection process as it explores the space of all possible combination of features. Since in practice the exhaustive search over all possible subsets of a feature set is not computationally feasible, several search strategies have been introduced in the literature to guide the feature selection. Examples of these search methods are: branch and bound, sequential forward selection, sequential backward elimination and genetic algorithms.

A number of authors have proposed different groupings for feature selection search methods (Dash & Liu, 1997; Yang & Honavar, 1998; Jain & Zongker, 1997). Broadly, these search strategies can be grouped into three categories: exponential (optimal), deterministic and randomized (stochastic) searches (Yang & Honavar, 1998; Jain & Zongker, 1997).

3.5.2.1 Feature Selection Using Exponential Search Methods

These algorithms evaluate a number of subsets that grows exponentially with the dimensionality of the search space. Although the order of the search space is 2^n, a fewer subsets are evaluated (Dash & Liu, 1997). The most representative algorithms under this class are: exhaustive search, branch and bound.

In general, for a set of n measured features, the only certain way to find the optimum subset of features, by whatever criterion, is to exhaustively test all 2^n subsets, a procedure which is computationally infeasible for all except when n is small, since the size of the search space grows exponentially with the number of features included (Jain & Zongker, 1997). As an alternative to exhaustive search, a widely investigated technique is branch-and-bound (BAB) search (Narendra & Fukunaga, 1997). The branch and bound algorithm has been shown to produce an optimal feature subset under specific circumstances, and its designers demonstrate that it is much faster than exhaustive search. The technique involves setting up a sorted search tree starting from the full feature set, with each node uniquely identified by the feature discarded at that node. Since the tree is sorted in order of the selection criterion, any node, which has a value of the selection criterion less than the current "best" value can be discarded, along with its attached sub-tree; since this section of the tree is guaranteed not to increase in value. In this way whole areas of the search space can safely be ignored,

increasing the computational efficiency of the feature selection process, while still guaranteeing to find the optimum feature subset.

BAB algorithm will only find the optimal feature subset under the monotonicity assumption. The monotonicity assumption states that the addition of features can only increase the value of the objective function. Nevertheless its computational cost is prohibitive for large feature spaces: in the worst case, it does an exhaustive search and its time complexity is exponential on the dimension of the feature space. Another problem of this approach is that, if the criterion distance is not monotonic, the BAB algorithm will face the nesting effect (explained in the next section). Furthermore, Yang and Honavar (1998) observe that while branch and bound approaches often work well with conventional statistical classifiers, their performance may be poor with non-linear classifiers such as neural networks.

3.5.2.2 Feature Selection Using Deterministic Search Methods

Deterministic methods produce a feature subset, which is always the same every time for a given data set (Jain & Zongker, 1997). Examples of these algorithms are: sequential forward selection, sequential backward selection and sequential floating selection.

Sequential forward selection (SFS) and sequential backward selection (SBS) algorithms are based on the same idea. Starting from the empty set, SFS first picks a feature that results in the highest objective function (i.e. classification accuracy). Then sequentially, the next best feature is added provided that the first one has already been chosen. This process is repeated until either the required number of features is chosen, or there is no benefit from adding further features. Alternatively, SBS starts from the complete set and eliminates the worst feature. Then sequentially, the next worst feature is removed. However,

SBS approach involves a larger number of feature set evaluations, which may make it unsuitable for very large data sets. A major disadvantage for both SFS and SBS is the lack of backtracking or what is called the nesting effect. A feature that is added by SFS at the beginning can't be removed; also a discarded feature by SBS can't be added again.

A modification to the SFS and SBS is the floating search methods. There are two main categories of floating search methods: forward (SFFS) and backward (SFBS). Basically, in the case of forward search (SFFS), the algorithm starts with a null feature set and for each step, the best feature that satisfies some criterion function is included with the current feature set, i.e., one step of the sequential forward selection (SFS) is performed. The algorithm also verifies the possibility of improvement of the criterion if some feature is excluded. In this case, the worst feature (concerning the criterion) is eliminated from the set, that is, it is performed one step of sequential backward selection (SBS). Therefore, the SFFS proceeds dynamically increasing and decreasing the number of features until the desired number of features d is reached.

The backward search (SFBS) works analogously, but starting with the full feature set (of size m) and performing the search until the desired dimension d is reached, using SBS and SFS steps. All of these approaches are heuristic, and there is no guarantee that they will find the globally best subsets. In addition, these algorithms have a tendency to become trapped in local minima (Zhang & Sun, 2002).

3.5.2.3 Feature Selection Using Randomized Search Methods

Unlike deterministic algorithms, stochastic algorithms include an element of chance. This means that two runs of the same algorithm with different random number seeds will produce different results. In fact, these algorithms incorporate randomness into their search procedure

to escape local minima. Examples are: simulated annealing (SA), tabu search (TS) and genetic algorithm (GA). In this section we will cover SA and TS and in the next chapter GA will be discussed widely.

Simulated annealing is an optimization technique based on an analogy with the physical annealing of solids (Sadiq & Youssef, 1999). Annealing refers to the process by which a solid material is first melted and then allowed to cool by slowly reducing the temperature. At high temperatures, all the particles of the solid are randomly organized as a liquid. The temperature of the system is then gradually lowered, and the particles arrange themselves in the lowest energy state as an orderly lattice. The probability that a particle is at any energy level can be calculated. As the temperature of the material decreases, the probability tends toward the particle configuration that has the lowest energy. The system is perturbed to yield a new configuration of the particles. Using this criterion, the material will eventually reach its equilibrium configuration.

For feature selection, the set of potential solutions assumes the role of the states of a solid, and the cost function (or fitness function) replaces energy. The goal of the procedure is then to achieve a state of minimum cost (corresponding to maximum fitness) by moving between solutions with a probability, which is dependant upon the temperature of the system. At a high temperature a higher cost solution is more likely to be accepted, while as the temperature decreases the probability of accepting a higher cost move also decreases. The temperature is lowered during the running of the algorithm according to a predetermined schedule (Van Laarhoven & Aarts, 1987).

Simulated annealing has found wide application in fields of science and engineering. SA's major advantage over other methods is an ability to avoid becoming trapped at local

minima. Thus, the ability to find the global optimum is not related to the initial conditions (i.e., the starting point). The primary disadvantages to SA are the subjective nature of choosing the SA configuration parameters (e.g., T and step size) and that it typically requires more response or objective function evaluations than other optimization approaches. Thus it tends to be very slow; and a realistic temperature schedule must be established, usually by trial-and-error.

Tabu search is an iterative procedure designed for the solution of optimization problems. TS was invented by Glover (Glover, 1986) and has been used to solve a wide range of hard optimization problems. Tabu search, like simulated annealing, is a neighborhood (local) search. Local search employs the idea that a given solution S may be improved by making small changes. Those solutions obtained by modifying solution S are called neighbors of S. The local search algorithm starts with some initial solution and moves from neighbor to neighbor as long as possible while decreasing the objective function value. The main problem with this strategy is to escape from local minima where the search cannot find any further neighborhood solution that decreases the objective function value. However, tabu search can escape from local minima.

Tabu search considers the set of all possible neighborhood states and takes the best one. Yet, it will take the best move in the neighborhood, even though it might be worse than the current move. The main principle behind tabu search is that it has some a short-term memory component (called tabu list), which keeps track of the states that has already been visited and it does not allow revisiting those states. Preventing previously visited states helps in two ways. It avoids the search getting into a loop by continually searching the same area without actually making any progress. In turn, this helps the search explore regions that it might not

otherwise explore, thus avoiding local minima. The moves that are not allowed to be revisited are held in the tabu list and these moves are called "tabu" (thus the name). Typically, old moves are removed from the tabu-list after some number of iterations.

During the search, the tabu status of a solution is overridden if certain criterion called the aspiration criteria is met. For example, if the cost of the solution found is better than the best known so far, then the aspiration criterion is satisfied and the tabu constraint is removed by allowing the move to this solution (Sadiq & Youssef, 1999). Simulated annealing and genetic algorithms are both memory-less, however, tabu search has a memory to record and guide the search, which prevents cycling through solutions.

Chapter 4 Genetic Algorithms for Feature Selection and Weighting

4.1 Genetic Algorithms Review

Genetic algorithms are search algorithms based on analogy with biology in which a group of solutions evolves via natural selection. Since their development by John Holland (Holland, 1975), genetic algorithms have been widely and successfully employed in many fields such as optimization problems, machine learning and pattern recognition. Genetic algorithms operate on a population of individuals that represents candidate solutions for a given problem. Each individual or chromosome compete with one another to reproduce based on Darwinian's principle of 'survival of the fittest' in each generation of evolution. An individual is evaluated by a fitness function that expresses how much this chromosome is appropriate as a solution. The best parent individuals, which survived the struggle in a GA population, crossover to produce better offspring. To prevent GA from being caught at a good but not optimal solution, mutation is performed. After a number of generations in evolution, the chromosomes that survived in the population are the optimal solutions. In general, GA consists of the following steps (which are depicted in Figure 3):

1. Start with a randomly generated population of n chromosomes (candidate solutions).
2. Evaluate each chromosome in the population by calculating the fitness function.
3. Create new chromosomes by mating current chromosomes; apply mutation and reproduction as the parent chromosomes mate.

4. Delete chromosomes from the current population to make room for the new chromosomes.
5. Evaluate the new chromosomes and inset them into the population.
6. If the stop condition is satisfied, then stop and return the best chromosome; otherwise, go to step 3.

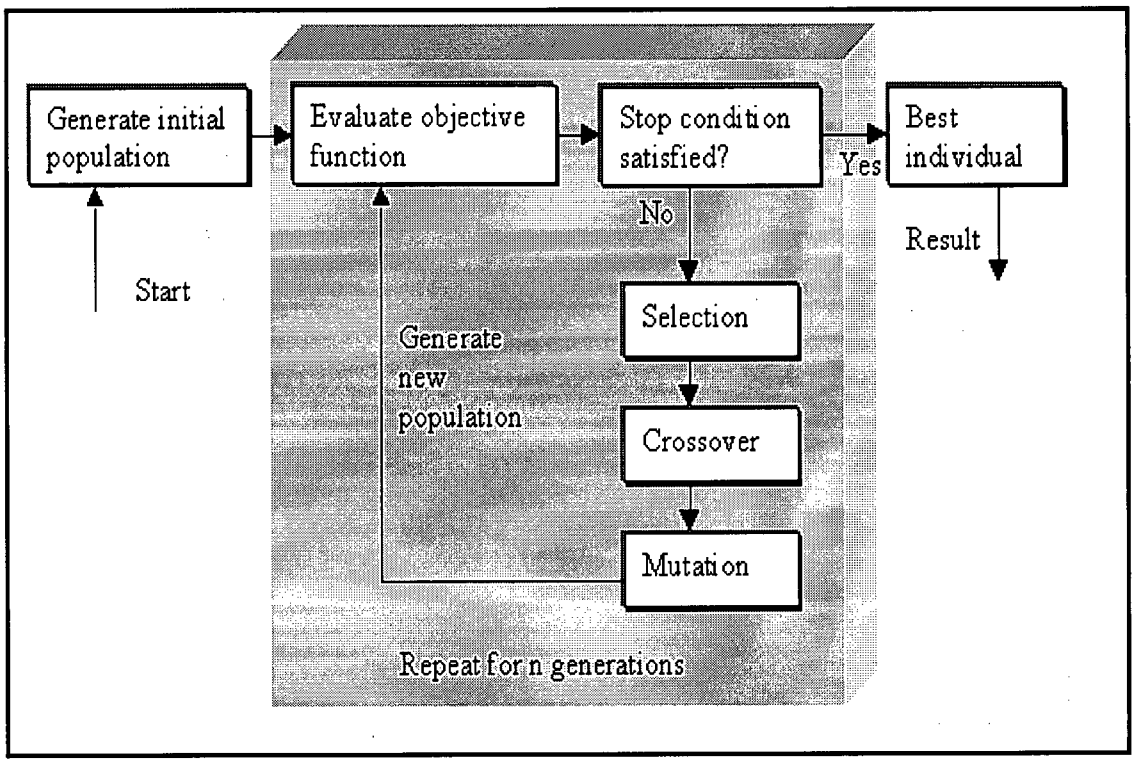


Figure 3: Basic steps of genetic algorithms.

4.1.1 The Basics of Genetic Algorithm

There are some basic operations and parameters found in every genetic algorithm. In this section we will discuss them in details.

4.1.1.1 Initialization

In general, it is required to initialize the population with diverse individuals. This will provide a good start for the GA and offer diversity in the population. There are many ways to arrange this initial diversity. Usually, the initial population is randomly generated; where candidate solutions (feature subsets) are created randomly from the search space with a uniform distribution. Another method for initialization is to use the solutions found by other search techniques. While this does not encourage diversity, it can guarantee that the genetic algorithm will do at least as well as the initial seed algorithm. However, this initialization method can lead to premature convergence to sub-optimal solutions.

4.1.1.2 Representation

In general, any representation for the individual genomes in the genetic algorithm can be used. The majority of the developmental work of GA theory and the most widely used representation is performed using a binary-coded GA. Historically, Holland (1975) worked with strings of binary bits. In a binary coding each chromosome is vector containing zeroes and ones with each bit representing a gene. The two main common representations are the binary and real number coding. They differ mainly in how the recombination and mutation operators are performed. However, other representations can be used such as, arrays, trees, lists, or any other object. But genetic operators must be defined (initialization, mutation,

crossover, comparison) for any representation used. Normally, the proper choice of genetic representation is problem-dependent and based upon the type of application.

4.1.1.3 Selection (Reproduction)

Reproduction or selection determines, which individuals are chosen to be copied to the mating pool from which the next generation is created. The selection is often performed to yield a mating pool with the same size as the original population. Then the mating pool serves as templates (parents) on which genetic operators are applied (crossover and mutation), interchanging and modifying sets of genes, to produce a new generation (offspring). The chance that an individual will be copied to the mating pool is based on the individual's fitness function. Generally, to emulate natural selection, individuals with a higher fitness should be selected with higher probability.

There are many different types of selection methods (Davis, 1991). Examples are proportional (roulette wheel) selection, tournament selection, and rank selection. The simplest selection scheme and the most commonly used is roulette-wheel selection (Goldberg, 1989). In this method, each chromosome is assigned a slice of a circle "roulette wheel", where the size of the slice is proportional to the chromosome's fitness. Each time an offspring is needed the wheel is spun, and the individual under the wheel's marker is selected to be in the pool of parents for the next generation. The process is repeated until the desired number of individuals is obtained (called mating population). This method will guarantee that good individulas will probabely be selected several times, while poor individulas may not be selected at all.

In tournament selection, n individuals are selected at random and the fittest is selected. The most common type of tournament selection is binary tournament selection,

where just two individuals are selected at random. The individual with the higher fitness will win. In the rank selection, ndividuals are sorted (ranked) on the ground of their fitness, where an idvidiual is assigned a rank r such that the least fit member has rank 0 and the fittest member rank 1. The selection probability is linearly assigned to the individuals, which depends on their rank and not on the actual fitness function value.

It is worth mentioning that none of the aforementioned selection methods are right or wrong. In fact, some will perform better than others depending on the problem domain being explored. In general, one will always select the fittest and discards the worst so that good solutions survive, and weak solutions die. However, imposing too much selection pressure, the solution will converge to less than optimal. On the other hand, applying too little selection pressure, a solution will possibly never reached. So, it is a balancing act to find the right selection technique for the problem at hand.

4.1.1.4 Crossover (Recombination)

Once all individuals have been selected for reproduction, an exact copy of them is made and put into the mating pool, a tentative new population, for more genetic operations. Notice that selection does not create any new chromosomes it just selects the best existing chromosomes and places them into the mating pool, which is used as the basis for creating the next generation.

In the mating pool, GA performs crossover first then mutation. Crossover is a version of artificial mating. In crossover, the GA selects two individuals at random from the mating pool. The selected individuals may be different or identical it does not matter. The GA then calculates whether crossover should take place using a parameter called the crossover

probability. This is simply a probability value p and is calculated by flipping a weighted coin. The value of p is defined by the user and usually has large value.

If the GA decides not to perform crossover, the two selected chromosomes are simply copied to the new population. On the other hand, if crossover happens, a crossover point is selected at random. Then the parents are crossed and separated at this point producing two children. These new child strings are then placed in the new population. Crossover is performed and continued until the new population is created. There are different techniques for crossover. The simplest technique is called one-point crossover (Davis, 1991) that randomly picks a crossover point and exchanges the segments to the right of this point between the two chromosomes.

On the other hand, in two-point crossover, two crossover points are selected at random. The genes between the two crossover points are exchanged. Another crossover method is the uniform crossover. Uniform crossover can be thought of as a generalization of one and two-point crossover. In this case, each bit in the offspring has equal chance of being chosen from either parent. A crossover mask, the same length as the individual structure is created at random and the parity of the bits in the mask indicates which parent will supply the offspring with which bits. Generally, for each bit in the first offspring, a 0 in the corresponding position of the crossover mask will mean that the bit is taken from first parent, while a 1 will mean that the bit is taken from the second parent. The exact opposite will apply for the second offspring. Yet, the best type of crossover operator to use will depend on the problem being solved.

4.1.1.5 Mutation

After performing the crossover operation to produce the new offspring, individuals in the population may undergo mutation. In binary coding, mutation is a very simple operation. A position in the string is chosen at random, and the value at that position is inverted from 0 to 1, or 1 to 0. Generally, a mutation is a small random change to a value that occurs with low probability called mutation rate. Mutation rate determines the probability that a mutation will occur. Large mutation rates increase the probability that good schemata will be destroyed, but increase population diversity. The best mutation rate is application dependent but for most applications is between 0.001 and 0.1.

Mainly, crossover represents a way of moving through the space of possible solutions based on the information gained from the existing solutions (exploitation). Mutation, on the other hand, represents innovation. Mutation is extremely important; by making small moves mutation allows a population to explore the search space (exploration) (Beasley, Bull & Martin, 1993). Additionally, it increases the diversity of the population by preventing the population from becoming saturated with similar chromosomes. Even if most of the search is being performed by crossover, mutation can be vital to provide the diversity which crossover needs.

4.1.1.6 Fitness Function

The fitness of a chromosome is based on the user defined evaluation function. It is a positive number that measures the goodness of an individual and discriminates between better and worse solutions. The simplest way is to use the evaluation function directly as the fitness function. However, since the fitness function must be non-negative, it is sometimes necessary to map the evaluation function into a valid fitness function. To maintain the topology of the

cost landscape, this is often a simple linear map. Other methods include exponential mapping but in all cases the important point is to reflect the value of the chromosome fitness in a correct way so that the problem is not distorted.

For some evaluation functions, the difference between the best and the worst chromosomes may be small. This results in a slow search, where the probabilities of reproduction for all chromosomes are almost the same. To approach this problem, fitness scaling has been introduced (Goldberg, 1989). Moreover, fitness scaling is also applied to overcome the problem of premature convergence. This problem arises from the fact that at the beginning there may be some extraordinary chromosomes in a population. In the first couple of generations these chromosomes might take over a huge part of the population before the crossover operator is able to construct a more diverse set of good chromosomes. Using fitness scaling, the few extraordinary chromosomes are scaled down while the lowly members of the population get scaled up. Fitness scaling imposes a maximum number of copies on an individual chromosome, thus preventing the few super chromosomes from taking over the population in the beginning.

4.1.1.7 Generation Replacement

There are different ways how to handle populations from one generation to the next generation. Given the constraint that the number of individuals should remain constant some individuals have to be discarded. Transition between generations can be done by total replacement, elitist replacement or steady state replacement. For total replacement or sometimes called non-overlapping populations only the newly created offspring enters the next generation and the parents of the previous generation are completely discarded. So, each generation the algorithm creates an entirely new population of individuals (Goldberg, 1989).

However, this has the disadvantage that a fit parent can be lost if it only once produces bad offspring.

To overcome the problem of replacing good parents elitist replacement is introduced. Elitism (De Jong, 1975) forces the GA to keep some of the best individuals at each generation. By allowing elitism, at least a copy of the best individual of the population is always passed to the new generation. This will gurantee that the best individuals ever discovered are not destroyed.

In the steady state or called overlapping populations, only a single population of individuals is maintained at any given time. Two individuals are selected from the population based on their fitness and then modified by mutation and crossover. The newly created individuals are then returned to the single population by means of the replacement operator, which selects chromosomes to be removed. In this variation, you can specify how much of the population should be replaced in each generation (generation gap) and the replacement criterion to be used (i.e. replace worse individual or replace random individual in the population) (De Jong, 1975).

4.1.1.8 Schema Theorem

Even though genetic algorithms depend on their work on randomized operators, they use random choice to guide a highly exploitative search (Goldberg, 1989). The concept behind why GA works well lies in the schema theorem (Holland, 1975). A schema (H) is defined as a template for describing a subset of chromosomes with similar sections. For example, consider a schema such as, #0000. This schema matches two chromosomes, 10000 and 00000. The template is a powerful way of describing similarities among patterns in the chromosomes.

According to Holland, the order of a schema (o(H)) is equal to the number of fixed positions (i.e., non-meta-characters) and the defining length of a schema (L(H)) is the distance between the first non * symbol position to the last non * position. Thus, the schema #00#0 is an order 3 schema (o(H) = 3) and has a length of (5-2) (L(H) = 3). Holland (1975) derived an expression that predicts the number of copies a particular schema, H, would have in the next generation after undergoing selection, crossover and mutation. In short, the schema theorem states that short, low order above average schemata (i.e. building blocks) receives exponentially increasing trials in subsequent generations.

So, during the GA run, the average fitness of all schemata existing in the population is evaluated, and accordingly they receive increasing or decreasing trials in the next generations. Thus, high performance schemata, whose average fitness is high, are copied and recombined to form instances of equal or higher performance schemata.

4.2 Genetic Feature Selection and Weighting

Feature selection is a process that often involves the optimization of partially understood and mathematically- uncharacterized systems. Sometimes, but not always, this process also involves a large number of parameters. Genetic Algorithms (Michalewicz & Fogel, 1998) are optimization techniques that are well suited for these situations. Another advantage of GA algorithm is that it has an implicit parallelism in which a set of solutions is evaluated at the same time, rather than individually (Goldberg, 1989) and thus, a GA gradually guides the search towards regions of optimality. In addition, GA has already proved to be efficient solutions for highly dimensional feature selection problems (Moser & Murty, 2000; Kudo & Sklansky, 2000).

GA-based feature selection (or GFS) has been employed in many applications, such as pattern/texture classification (Vafaie & De Jong, 1993), character recognition (Kim & Kim, 2000), medical diagnostics (Handels, Ross, Kreusch, Wolff & Poppl, 1999) and data mining (Martin & Vila, 1998). In addition, genetic feature weighting (GFW) was also employed and presented in (Kelly & Davis, 1991; Punch et al. 1993; Komosinski & Krawiec, 2000; Demiroz & Guvenir, 1996). Despite this diversity there are some common aspects, which must be decided upon when using GA for feature selection/weighting problem. These aspects are discussed below.

4.2.1 Chromosome Representation

In GA-based feature selection, each 'individual', which represents a feature subset, competes with other individuals for survival into the next generation of individuals. An individual is often represented as a binary string of finite length *n,* where *n* is the number of features. A value of '1' in the string indicates that the corresponding feature is included in this subset, while a value of '0' means it is not included. In the case of GA-based feature weighting, continuous real valued strings instead of binary strings represent the individuals of a population. The value of a gene in the chromosome indicates the weight of the corresponding feature. This representation allows the relative importance of each feature to be assessed according to its weighting. (Punch et al. 1993; Komosinski & Krawiec, 2000) used both binary and real valued weighting.

4.2.2 Type of Classifier

The majority of surveyed genetic feature selection GFS papers employed either neural networks or nearest neighbor classifiers. Nearest neighbor classifiers are very widely used

because they are robust, non-parametric, fast and easy to understand, and hence have been used in conjunction with OCR applications in several experiments (Moser & Murty, 2000). GFS systems that employed nearest neighbor classifiers are presented in (Kelly & Davis, 1991; Punch et al. 1993; Komosinski & Krawiec, 2000; Kudo & Sklansky, 2000).

On the other hand, several authors have used neural network classifier for GFS. Examples of such works are described in (Jack & Nandi, 2000; Sahiner, Chan, Wei, Petrick, Helvie, Adler & Goodsitt, 1996; Yang & Honavar, 1998). The main disadvantage of using neural networks in the GFS problem is that training a neural network for each evaluation is computationally expensive. On the other hand, nearest neighbor has the disadvantage of high storage requirements since all the training instances are stored in memory.

4.2.3 Fitness Function

Some of the fitness functions used in GFS are solely based on classification error rate. This type of feature evaluation results in an unconstrained optimization problem (Siedlecki & Sklansky, 1989). In contrast, Yang & Honavar (1998) use a multi-criterion fitness function, which incorporates both accuracy and cost of classification. Siedlecki & Sklanski (1989) use a fitness function based on a penalty formula, and the number of features selected. The penalty function apportions punishment values to feature sets that produce error rates, which are greater than a pre-defined threshold. It is worth noting that fitness functions that use the number of features and the classifier's error rate lead to constrained optimization problems (Siedlecki & Sklansky, 1989).

4.2.4 Computational Speed-Up Techniques

Numerous attempts have been undertaken to speed-up GA-based feature selection systems. Brill, Brown & Martin (1992) apply a nearest neighbor classifier to evaluate feature sets by running multiple populations independently on multiple processors. This process results in a diminished set of feature sets. These are then conclusively evaluated using a full-fledged counter-propagation neural network. Brill et al. (1992) demonstrate that the idea of separating the evaluation classifier from the eventual target classifier leads to significant reductions in computational time. Punch et al. (1993) employ a Micro Grain distributed GA. In this method, groups of feature sets are passed to individual processors for fitness evaluation. Given that fitness evaluation occupies most of the computational time of a wrapper-configuration GFS, it is not unusual that this technique leads to linear speed-ups.

Recently, Moser (1999) proposed Distributed Vertical GA's to very large-scale feature selection applications with more than 500 features. His technique allocates subsets of the test patterns to different evaluators. An adaptive load-balancing algorithm is used to manage the capabilities of a network of heterogeneous computers.

4.2.5 GA Parameters

Several GA parameters need to be determined, such as population size, number of generations, selection methods, and crossover and mutation probabilities and so on. In fact, it is a problem common to all GA applications: how do we decide these parameters in order to make GA truly autonomous optimization and design tools (that require minimal human intervention)? In fact, there's no clear guide for the choice of the GA parameters for a specific type of problems like GFS. Many authors have used several variations for these

parameters in literature. However, the choice of these parameters is usually done by trial and error or based on experience.

4.3 Comparisons

How does GA-based feature selection perform in comparison with other feature selection algorithms? Several studies exist that compare GA with other feature selection search algorithms. For example, Vafaie & DeJong (1993) compared sequential backward selection SBS with GA for feature selection in texture classification problem. They observe that SBS is "brittle"; that is, the algorithm becomes trapped in local minima due to higher-order interactions between features. On the other hand, they show that GA outperforms SBS in robustness because GA may escape from such minima, since they are highly stochastic.

Moreover, Handels et al. (1999) compare a number of feature selection methods: Genetic Algorithms other heuristic search, and 'greedy' methods (forward and backward searches). Their domain of application is diagnosis of skin tumors. The best classification accuracy was obtained by the GA. Furthermore, Estevez & Fernandez (1999) perform a comparison between Genetic Algorithms, statistical methods and the leave-one-feature-out approach, all in the context of classifying wood board defects. The GA gives the best performance among the three techniques, but at the expense of high computational complexity.

Additionally, Siedlecki & Sklansky (1989) suggested that the GA approach is particularly useful when the dimensionality of the entire feature set is greater than 20. Recently Kudo & Sklansky (2000) provided an extensive comparison of different feature selection algorithms (including GA) for large feature spaces. The authors conclude that Genetic Algorithm is more suited than other feature selection algorithms for problems with

more than 50 features. However, for small (0-19 features) and medium (20-49) cardinality, GA though useful, is too slow.

As for the performance of GA in comparison with SA for the feature selection problem, there appears to be no record of any study. In fact, there have been sparse reports of the use of SA for feature selection. For example, Ferri & Riccioni (1992) describe an application of simulated annealing to the selection of observatories for predicting overnight temperature. Their method outperforms forward selection, using a temperature method tailored to the problem. But there exists no comparison between SA and GA for feature selection problem, which could be an area for future research.

Zhang & Sun (2002) have recently compared tabu search with GA and other feature selection methods such as sequential methods and branch and bound method. In a set of experiments, which were implemented using a dataset of 30 features, they have shown that the tabu search algorithm obtained the best solution among all algorithms tested (including the GA) and with a lower computational cost than the GA. In a second set of experiments, which used 20 features and 2 class sets, they used the Mahalanobis distance to measure the goodness of a feature subset (the probability of error is inversely proportional to the Mahalanobis distance). Their results demonstrate that 18 times out of the 20 runs they performed, GA has achieved the optimal solution. In the worst case, it still obtained a near optimal solution. As for the tabu search, it achieved global optimal feature subset in all the 20 runs.

Several observations can be made regarding the results reported in (Zhang & Sun 2002). They have replaced the error rate classifier's function with a model of the classifier's error rate, which is a function of the feature selection vector. So, the true performances of the

feature selection algorithms were never really tested on a true classifier, thus, reducing the reliability of the reported results. In addition, the datasets they used were artificially generated, not a real datasets. Finally, the differences in performance mentioned between the GA and the tabu search were small that it does not, in itself, provide an absolute confirmation that tabu search TS is superior to GA. In fact, to test the validity and credibility of these results, both GA and TS need to be compared and tested on real datasets and using the true error rates generated from a classifier.

4.4 Genetic Feature Selection for Character Recognition Systems (Literature Review)

Though (Siedlecki & Sklansky, 1988) is probably the first paper to suggest the use of genetic algorithms for feature selection, several other researchers have, in fact, used them for feature selection. However, there are somewhat sparse examples in the literature of character recognition applications that employ GA to satisfy its feature selection needs. This fact becomes particularly pronounced, when one looks at the steadily increasing number of GA-based feature selection (or GFS) applications in pattern classification domains. Below is a list of these studies, classified according to the type of input (printed or handwritten). We also include GFS applications to signature verification due to similarity of the problem characteristics.

4.4.1 Recognition of Printed Characters

Smith, Fogarty & Johnson (1994) applied GFS to printed letter recognition. They used 24 features (16 features, plus 8 redundant features) to describe the 26 letters of alphabet. A nearest neighbor classifier using Euclidian distance was used for classification. To speed the

GA run, a sub-sampling method is used where 10% of the training data is randomly sampled and selected at the beginning of each run; only this subset is used for the GA evaluation. The system reduces the feature set to 10 features. It does so while maintaining a mean error rate lower than that generated when using all 24 features.

4.4.2 Recognition of Handwritten Characters

A handwritten digit recognizer, which is used to assess a GFS algorithm, is presented in (Kim & Kim, 2000). During the training phase, the recognizer performs clustering to obtain a KxP dimension codebook (K is the number of clusters and P is the number of features) that represents the centroid of the K clusters. During the testing phase, a matching process performs a distance calculation between the centroids and the testing data. The objective is to use GFS to speed up the matching process as well as to reduce the size of the codebook. Testing was carried out using two datasets: one with 74 features and another with 416 features. For the 74-feature test, experimental results show that the recognition rate trivially decreased when the number of features was lowered. However, for the case of the 416-features test, the GA-selected set of features leads to a higher recognition rate than the original 416 feature-set does. This result emphasizes the usefulness of GFS in large search spaces.

In addition, Kim & Kim (2000) propose a variable weight method to assign weights to features in the matching process. During the GA feature selection, a weight matrix for the features is build, which represents how often the feature was selected throughout the GFS. After the GFS is complete, the matrix is used in the recognition module. Features having high weights denote more frequently selected features, which implies their importance over low weight features. Results using this variable weight method show a slight improvement in

the performance over the un-weighted method. One important remark is that this method of weighting the features is completely different than the one we are using. Their method depends on counting the frequencies of selecting the features during the GFS, while in our approach the weights are assigned using the GA itself. A major drawback for weight matrix method suggested in (Kim & Kim, 2000) is that it does not achieve any dimensionality reduction and the number of features remains the same. Also, the enhancement in accuracy achieved is very small, where the non-weighted method has an accuracy of 96.3% while the suggested weighting method has 96.4%.

Furthermore, Gaborski & Anderson (1993) use a GA for feature selection for a hand-written digit recognition system. They used several variations for population organization, parent selection and child creation. The result is that the GA was capable of pruning the feature set from 1500 to 300, while maintaining the same level of accuracy achieved by the original set.

Moreover, Shi, Shu & Liu (1998) suggest a GFS for handwritten Chinese character recognition. They craft a fitness function that is based on the transformed divergence among classes, which is derived from Mahalanobis distance. The goal is to select a subset of m features from the original set of n features (where $m<n$) for which the error is minimum. Starting with 64 features, the algorithm is able to reach to 26 features with less error rate than the original feature set.

Finally, Moser & Murty (2000) investigate the use of Distributed Genetic Algorithms in very large-scale feature selection (where the number of features is larger than 500). Starting with 768 initial features, a 1-nearset-neighbor classifier is used to successfully recognize handwritten digits using 30 SUN workstations. The fitness function used is a

polynomial punishment function, which utilizes both classification accuracy as well as the number of selected features. The punishment factor is used to guide the search towards regions of lower complexity. The experiments were aimed to demonstrate the scalability of GFS to very large domains. The researchers were able to reduce the number of features by approximately 50% while having comparable classification accuracies to those of the full feature set.

4.4.3 Signature Verification

Fung, Liu & Lau (1996) use GA to reduce the number of features required to achieve a minimum acceptable hit-rate, in a signature verification system. The goal was to search for a minimum number of features, which would not degrade the classifier's performance beyond a certain minimum limit. They use the same fitness function, which was proposed in (Siedlecki & Sklansky, 1989), which is based on a penalty formula and the number of features selected. The penalty function apportions punishment values to feature sets that produce error rates, which are greater than a pre-defined threshold. Using a 91-feature set to describe 320 handwritten signatures from 32 different persons, the system was able to achieve an accuracy of 93% with only 7 selected features as opposed to a 88.4% accuracy using the whole 91 feature set.

Chapter 5 A Comparative Study between Genetic Feature Selection and Weighting

In chapters 5 and 6, we carry out two sets of studies; the results of these studies are recently reported in (Kharma, Hussein & Ward, 2002a; Kharma, Hussein & Ward 2002b). The first set (described in this chapter) compares the performance of GA-based feature selection (GFS) to GA-based feature weighting (GFW), within the context of character recognition systems and under various conditions. The second set of studies will be described in chapter 6.

5.1 Introduction

In chapter 4 we have illustrated the application of genetic feature selection/weighting in many application and specifically to the domain of character recognition systems. What is clear from that review is that:

1. Many features can be used for the character recognition systems and there is no easy way to distinguish useful features or sets of features from less informative features.
2. Much research has been done into feature selection, and a large number of algorithms have been developed and applied to various problem domains.
3. Genetic Algorithms have proven to be an effective tool for reducing the feature-dimensionality of character (or signature) recognition problems, while maintaining a high level of accuracy (of recognition).

4. GFS has only been used off-line due to the time it takes to run a GA in the wrapper approach.
5. There is no published work that focuses on the more complicated problem of automatic feature weighting using GA for character recognition applications.
6. There is no comparison done to compare the performance of genetic feature selection (GFS) and genetic feature weighting (GFW) for the character recognition problem.

5.2 Purpose

The main purpose for our work is to apply genetic algorithms for the problem of feature weighting for character recognition application. This work is motivated by the fact that there is no published work in the literature that has applied genetic feature weighting (GFW) in the context of character recognition problems. So it will be applied for the first time. After the employment of GFW to character recognition application, we are interested in comparing the performance of both GFS and GFW also in the context of character recognition applications. We are encouraged to perform such a comparison because it is often mentioned in the literature that feature weighting always has the potential of working better than (or at least as well as) feature selection, when applied to the same situation (Komosinski & Krawiec, 2000; Punch et al. 1993). However, this proposition was never fully and comprehensively assessed before. Only a single comparison exited in the literature (Kohavi, Langley & Yun, 1997), which compares the classification accuracy of feature selection (FS) and feature weighting (FW). However, the search method used in this comparison is not genetic algorithms, and the comparison was not employed for character recognition applications. Naturally, we expect that for regular datasets (not necessarily having redundant or irrelevant features) genetic feature weighting would outperform genetic feature selection in classification rates. This is

because features, in general, have a varying degree of relevance (strongly relevant, weakly relevant or irrelevant as described in section 5.6.3), and FW assigns weights to features that reflect their relative relevance to correct classification. Highly relevant features would be assigned high weights relative to the weights of redundant or irrelevant features. On the other hand, FS treats the features as either relevant or irrelevant, and does not accommodate for a varying degrees of relevance. Therefore, we intend to test the validity of this proposition. In addition, since only a single comparison exists in the literature (kohavi et al. 1997) that compares FS and FW in terms of classification accuracy only, we need to carry out an inclusive comparison between GFS and GFW for character recognition applications. The comparison will tackle the following issues:

1. The effect of varying the number of values that weights can take on the number of selected features (comparison 1).
2. Comparing the performance of both GFS and GFW in the presence of irrelevant features (comparison 2).
3. Comparing the performance of both GFS and GFW in the presence of redundant features (comparison 3).
4. Comparing the performance of both GFS and GFW for regular databases (not necessarily having redundant or irrelevant features) (comparison 4).

Before we describe the comparative study and evaluation as well as their results, we will first describe the developed system, the experimental platform used to carry out these studies and the methodologies employed.

5.3 The Developed System

We built a pattern recognition experimental bench with the following modules: (a) a pre-processing module; (b) a feature extraction module (FE); (c) a feature selection and weighting module (FSW) which embeds the GA optimization component; and (d) an evaluation module, which embeds the classifier component. The system is shown below in Figure 4. The purpose of the pre-processing module is to prepare an input pattern for effective and efficient extraction of relevant features. After applying the pre-processing functions on input images to obtain enhanced and clean images, the feature extraction module follows. The feature extraction module applies certain functions that measure a set of relevant features for classification (we will describe the pre-processing and feature extraction functions used in details in chapter 6 in section 6.6.1).

The next module, which goes after feature extraction, is the feature selection/weighting module (FSW). The FSW module assigns binary or real valued weights to the extracted set of features, before presenting them to the classification module. The FSW module is configured by the GA optimizer (or simply GA). The purpose of the GA is to find a set of features and associated weights that will optimize the (overall) performance of the pattern recognition system, as measured by a given fitness function. The final module is the evaluation module. The evaluation module is essentially the fitness function. Given that we are using the wrapper approach, the fitness function used implements the k-NN classifier. Since we are comparing both GFS and GFW in terms of performance and number of features selected, we decided to use a fitness function that is solely based on the classification error rate. We did not want to impose any selective pressure to find weight sets that have smaller number of features. After the termination of the GA, we have the final and best feature subset, which

has the lowest error rate. During the GA optimization process (training/testing phase) we used training and testing samples. After the GA termination, the validation set is used in validation phase to evaluate the quality of the produced solution by running the k-NN classifier using the final set of weights obtained form the GA (this will be explained more in details in section 5.5).

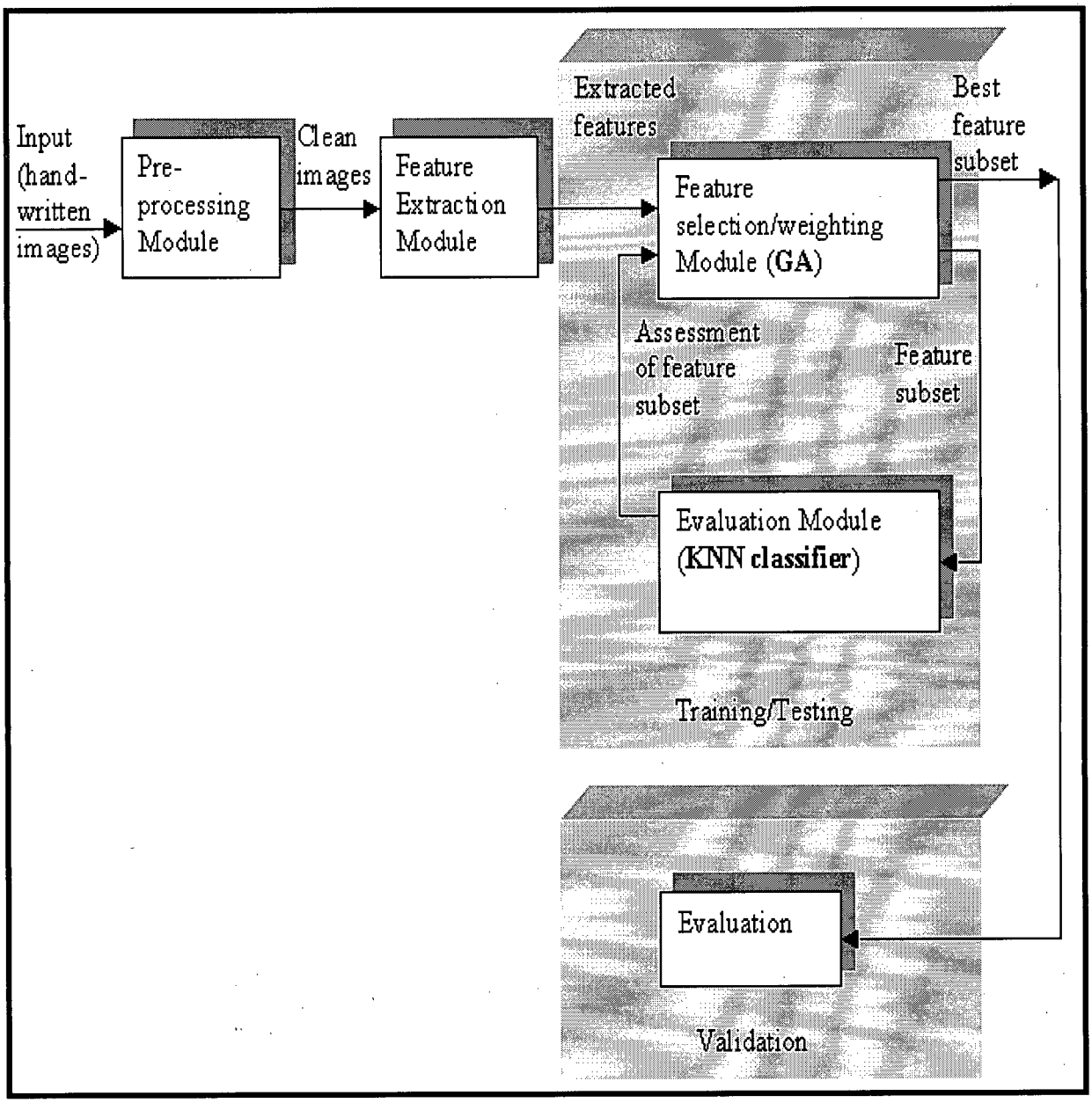


Figure 4: The developed system.

5.4 Experimental Platform

The experimental work described in the following sections is implemented using publicly accessible character databases and GA. This makes it possible for researchers to replicate our work and attempt to improve upon our results. The experiments were run on a Pentium III (500MHz) PC with 128MB of RAM and running Windows 98.

The Machine Learning Data Repository at the University of California at Irvine (Murphy & Aha, 1994) is our source of databases. All the experiments done through this research work are performed on real-world databases. Artificially generated databases do not have the level of noise nor the variation, which exists in real databases. We used three different handwritten digits databases:

1. "Optical Recognition of Handwritten Digits" database (or DB1), which consists of 64 features. These features were extracted from 32x32 bitmaps of handwritten digits written by 43 people. The 32x32 bitmaps are divided into non-overlapping blocks of 4x4 and the number of on pixels is counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range [0,16].
2. "Pen-Based Recognition of Handwritten Digits" (or DB2), which consists of 16 features. These features are obtained from handwritten digits samples written by 44 writers on a tablet. The 16 features were extracted from the coordinates information after re-sampling the written digits to obtain a constant number of regularly spaced points.
3. "Multiple Features Database" (or DB3), which consists of features of handwritten numerals. These digits are represented in terms of six feature sets, which are: Fourier coefficients of the character shapes, profile correlations, Karhunen-Love coefficients,

zoning features, Zernike moments and morphological features. Of those feature sets, we have only used the last two feature sets (Zernike moments and morphological features), which contain 47 and 6 features, respectively.

The Genetic Algorithm used for optimization is the Simple Genetic Algorithm (or SGA) described by Goldberg (Goldberg, 1989). The actual software implementation used comes from the "Galib" GA library provided by the Massachusetts Institute of Technology (Matthew, 1999). In this SGA we used non-overlapping populations, roulette-wheel selection with a degree of elitism, as well as two-point crossover. As for the choice of GA parameters, it was merely based on trial and error and other suggestions presented in the literature. We tried several variations of these parameters and for the problem at hand they had somewhat comparable results. So, the GA parameters we used are crossover probability *Pc* of 0.9. As for mutation, we used two styles: flip mutation for GFS, and Gaussian mutation for GFW. Gaussian mutation uses a bell-curve around the mutated value to determine the random new value. Under this bell-shaped area, values that are closer to the current value are more likely to be selected than values that are farther away. The mutation probability *Pm* was 0.02. The number of generations *Ng* was 50, and the population size *Pop* was also 50. The fitness function was the classification accuracy of our own 1-nearest-neighbour (1-NN) classifier.

5.5 Methodology

The comparison between GFS and GFW is done using the following methods:

- Feature selection and weighting in the wrapper approach and using the k-nearest neighbor as the classifier and genetic algorithms as the search method.
- The value of k for the k-nearest neighbor was fixed to one, because our objective is to compare genetic feature selection and weighing not to study the effect of k.

- Prior to feature selection or weighting, all feature values are normalized to the range of [0,1] to overcome the differences in units between feature values (explained before in section 2.7).

GA is appropriate for feature selection and weighting. GA is suitable for large-scale, non-linear problems that involve systems, which are vaguely defined. Further, character recognition systems (a) often use a large number of features, (b) exhibit a high degree of inter-feature dependency, and (c) are hard, if not impossible, to define analytically.

In our study, we are obliged to use a wrapper approach to find the best weight set, despite their relative computational inefficiency, because the classifier is needed to determine the relevancy of features. Also, since the feature selection is done off-line, the execution time of the algorithm is not as critical as the optimality of the feature set generated. In addition, there have been attempts to speed up the GFS run time (see section 4.2.4) by introducing methods of global parallel GA and course-grain parallel GA. However, applying these methods is outside the scope of our work.

In any case, a classifier is necessary, and the k nearest neighbor (K-NN) classifier is our first choice because of its excellent asymptotic accuracy, simplicity, speed of training, and its wide use by researchers in the area. In fact, Weideman, Manry & Yau (1989) have compared the performance of both nearest neighbor classifier and neural networks using back-propagation for numeric handwritten character recognition. They state that in terms of recognition performance, both nearest neighbor and neural networks have almost similar results. So, the choice between them was based on a tradeoff between either memory requirements or computational time. We decided to use nearest neighbor for several reasons. First nearest neighbor is easy to implement, requires no training as opposed to the neural

networks and is computationally faster. In addition, GFS in the wrapper approach is computationally expensive. So, using neural networks in conjunction with GFS would make it more computationally demanding. Finally, memory is cheap and the nearest neighbor additional storage requirements can be easily compensated for.

It is important to mention that throughout our work we have tested the generated feature subset by a validation phase. The validation phase is essential to judge the generality of the resultant feature subsets. Despite the fact that validation is vital to evaluate the quality of the obtained results, the majority of the results reported by the researchers in the field of feature selection ignore the use of validation set (Moser, 1999). So, in the course of the experiments, we divided the samples into three sets: training, testing and validation. The training and testing samples were used during the optimization process (training/testing phase). The validation set, on the other hand, is fully kept aside to be used in the validation phase (see Figure 4 in section 5.3).

In addition to the aforementioned methods, we also used more than one database (see section 5.4) and different error estimation methods. The reason is that certain databases and error estimation methods were more suitable than others for carrying out particular experiments. For example, in comparison 1 (will be described in section 5.6.1), the dataset used in this experiment is DB1, which contains a relatively large number of features (64). When running preliminary tests on this dataset a lot of features were eliminated, while the classification accuracy remained almost the same. Using a database, which has a lot of features to be removed, will be useful in studying the effect of varying the number of weights on the number of selected features. In addition, for both comparisons 2 and 3 (will be described in sections 5.6.3, 5.6.5) we used DB3 with the dataset that contains 6 features. We

needed a dataset, which contains small number of features for these two comparisons because we are continuously adding irrelevant and redundant features to the original feature set until we reached 60 features. So, using a dataset, which initially contains large number of features will tremendously increase the run time and the dimensionality of the problem. For comparison 4 (will be described in section 5.6.7), we chose both DB2 and DB3 because we wanted to compare the performance of both GFW and GFS on datasets that contains little redundant/irrelevant features, which is the case for these two datasets. This is unlike the 64 optical digits dataset, which has large number of features and contains many redundant/irrelevant features.

As for the error estimation methods, it is known that the leave-one-out cross validation method has larger computational requirements than the random sub-sampling method (see sections 2.8.2.2 and 2.8.2.3). Therefore, for large data sets, leave-one-out is computationally expensive and random sub-sampling or k-fold methods are preferred. Hence, the main reason to chose the random sub-sampling error estimation method for comparisons 2 and 3 rather than the leave-one-out (which is used in comparisons 1 and 4) is that both comparisons 2 and 3 experiments involve running GA on larger number of features (up to 60 features) and using relatively large number of training samples (1000), which would require extensive computational time. Contrary to comparisons 2 and 3, comparison 1 uses a small number of training samples (200), though it has large number of features (64), which makes it computationally undemanding. Moreover, experiments on comparison 4 involve small number of features, which ease the use of the leave-one-out method.

5.6 Comparative Study

Following are four empirical studies that compare the performance of GA-based feature selection (GFS) to GA-based feature weighting (GFW) using the described system in section 5.3, with respect to (a) the number of eliminated features, and (b) classification accuracy. In all the experiments, 1-NN stands for the 1-nearest neighbor classifier (i.e. no GA-based optimization), FS stands for GFS, and *X*FW stands for GFW with an *X*-number of weight values. In the case of GFS, the GA use binary weights in the FSW module (shown previously in Figure 4 section 5.3), while for the case of GFW the weights are real valued.

5.6.1 The Effect of Varying the Number Weight Values on the Number of Selected Features (Comparison 1).

It is known that feature selection, generally, reduces the cost of classification by decreasing the number of features used (Dash & Liu, 1997). Genetic feature weighting (GFW) methods should, theoretically, have the same potential (because genetic feature selection GFS is a special case of GFW). However, it has been argued that in reality, GFS eliminates many more features than GFW. How many more, though, is unknown. Since the essential difference between GFS and GFW is the number of values that weights can take, we decided to study the relationship between the number of values that a weight can take and the number of eliminated features. We also tested a method for increasing the ability of GFW to eliminate features.

The database used in this experiment is DB1, which contains a relatively large number of features (64). The error estimation method used for the training phase is the leave-one-out cross validation technique (explained in section 2.8.2.3). It is applied to the training

data itself. The number of training samples is 200. As explained before in section 5.3, the resultant weights are assessed using a validation data set of size 200. This validation set is new, in that it is not used during training.

5.6.2 Results of Comparison 1

The results of the experiments are presented in Table 1 and Table 2. We will first describe the columns in both tables then we will describe the purpose of each table. The following description of columns applies to the contents of both tables. The first column presents the method of feature selection or weighting. It is (a) 1-NN, which means that a 1-NN classifier is applied directly to the full set of features, with no prior selection or weighting; (b) FS, which means that feature selection is applied first before any classification is carried out; or (c) FW (with different weighting schemes), which means that feature weighting is applied prior to 1-NN classification. The second column, increment value, shows the difference between any two successive weight levels. The third column presents the total number of weight values that a weight can take. A weight can take on any one of a discrete number of values in the range [min, max] and using the specified increment value. It is calculated as follows: Number of weight values = [(max-min)/increment] +1. Note that the difference between any two consecutive values is the 'increment value'. The fourth column is simply the inverse of the number of weight values, which is termed: 'probability of zero'. It represents the probability of a weight taking the value zero (or POZ), assuming a uniformly random distribution of weight values, and calculated as: POZ = 1/number of weight values. The fifth and sixth columns show classification accuracies for both training and validation phases, respectively. The final column presents the number of eliminated features, which we call the number of zero features (i.e. the number of features that are assigned a weight value

equal to zero). This is easily computed by subtracting the number of selected features from the total number of features (64). It is worth noting that values shown in columns 5-7 are the average values of five runs having identical parameters but different seed for the GA.

For Table 1 we have compared FS with FW where the weight values are ranging between 0-10 inclusive and the number of weight values that a weight can have varies for in each row (shown in Table 1). Our main purpose in this table is to study the effect of changing the weight values on the number of features eliminated for both FS and FW. For Table 2, we also compared FS with FW, but FW in this case have different weight settings (weights are discrete, or weights are in range [0,10] and sometimes weights less than a certain threshold value is set to zero, see Table 2). In Table 2, we aim to verify the results obtained in Table 1 while using different weight settings than those used in Table 1. In addition, we also intend to test a method for increasing the ability of GFW to eliminate features without changing the number of weight values. This is achieved by setting the weight values below a certain threshold to zero.

Method of Selection/ Weighting	Increment Value	Number of Weight Values	Probability of Zero	Accuracy of Training	Accuracy of Validation	Number of Zero Features
1-NN	-	-	-	97.5	84.5	-
FS	1	2	½	99.4	84.8	27
FW	0.5 (1/2)	20+1	1/21	99	84	3
FW	0.25 (1/4)	40+1	1/41	98.9	83.8	1
FW	0.125 (1/8)	80+1	1/81	99	83.5	0
FW	0.0625 (1/16)	160+1	1/161	98.9	83.9	0
FW	0.03125 (1/32)	320+1	1/321	98.6	83.2	0
FW	0.015625 (1/64)	640+1	1/641	99	83.2	0
FW	0.0078125 (1/128)	1280+1	1/1281	98.9	83.7	0
FW	0.0039 (1/256)	2560+1	1/2561	98.2	83	0

Table 1: Accuracy of recognition and number of zero features for various selection and weighting schemes.

Method of Selection/ Weighting	Increment Value	No. of Weight Values	Prob. of Zero	Acc. of Train.	Accuracy of Validation	No. of Zero Features
FS	1	2	½ (0.5)	99.4	84.8	27
FW: three discrete values: 0, 0.5, and 1.	0.5	3	1/3 (0.33)	97.8	83.3	18
FW: values belong to [0, 10], with weights < 1 forced to 0.	0.125	81	8/81 (0.098)	98.1	84.2	7
FW: values belong to [0,10].	1	11	1/11 (0.09)	98.5	82.8	5
FW: six discrete values: 0, 0.2, 0.4, 0.6, 0.8, and 1.	0.2	6	1/6 (0.166)	98.3	83.3	8
FW: six discrete values: 0, 0.2, 0.4, 0.6, 0.8, and 1, with weights < 0.8 forced to 0.	0.2	6	4/6 (0.66)	96.6	81.6	38
FW: six discrete values: 0, 1, 2, 3, 4, and 5.	1	6	1/6	97.9	83.4	9
FW: six discrete values: 0, 1, 2, 3, 4, and 5, with weights < 4 forced to 0.	1	6	4/6 (0.66)	97.4	81.2	34

Table 2: Accuracy of recognition and number of zero features for various selection and weighting schemes, some with low weights forced to zero.

The following observations can be made, based on the results in Table 1.

- Although feature selection succeeded in eliminating roughly 42% of the original set of features (27 out of 64), classification accuracy did not suffer as a result. On the

contrary, training accuracy increased to 99.4% from the 97.5% realized by the 1-NN alone. Also, validation accuracy increased to 84.8% from the 84.5% value achieved by the 1-NN classifier (alone).

- FS far outperforms FW in terms of the number of zero features (i.e. eliminated features). Also, training accuracies achieved by both FS and FW were better than those achieved by the 1-NN classifier (alone). Using the validation set, the accuracy levels achieved by FW range between slightly worse to worse than the accuracy levels achieved by the 1-NN classifier. In contrast, the accuracy levels achieved by FS are slightly better than those of the 1-NN classifier alone (and hence better than those of FW as well).
- Increasing the number of values a weight can take beyond a certain threshold (81, in this case) reduces the number of zero features to nil. This suggests that the greater the number of weight values the less likely it is that any of the features will have zero weight, and visa versa. Whether this relationship is (roughly) proportional or not, is studied below.

A careful observation must be pointed out here: GFS/GFW does not eliminate features at the expense of classification accuracy. This is due to the following reasons:

1. The fitness function used is dependent on classification accuracy only. There is no selective pressure to find weight sets that have smaller number of features.
2. Because we are running the GFS/GFW in the wrapper configuration, GA has a continuous feedback loop from the classifier, which continuously guides it towards higher accuracies.

The following observations can be made, based on the results in Table 2.

- The number of zero features is greater in cases where the number of possible values a weight can take is countably finite, than weights take values from an infinitely dense range of real numbers.
- When we use FW with discrete values (0,1, 2, 3, 4, 5), and forced all weights less than four to zero, FW actually outperforms FS in the number of zero features, however, the classification rates of FW in this case for both training and validation decrease. It is worth noting that the Probability of Zero for this FW configuration is 0.66, compared to 0.5 for FS. The same observation can be made for FW using values (0,0.2,0.4,0.6,0.8,1) with weights less than 0.8 forced to zero. In fact, these two FW settings were tried to emphasize the fact that number of zero features is influenced by the POZ. Hence changing the POZ, even without changing the number of values that a weight can take will certainly affect the number of zero features.
- Using FW with weights in the range of [0,10] and with weights less than 1 forced to zero increases the ability of GFW to eliminate features without changing the number of weight values. Note that there is difference between this method, which forces weight values less than a certain threshold to zero, and with using less number of weight values from the first place. In general, the weights that have values near to zero indicate that they are irrelevant and not contributing to the classification. So, applying this method will allow weights, which have values that are quite low or near zero (but not zero) to be eliminated, while keeping at the same time, the number of weight values that a feature can take unchanged (not decreased). This in turn, will allow features to have more weight values to accommodate their varying degrees of relevance, which is contrary to decreasing the number of weight values from the first

place. However, the classification rates for both training and validation in this case are still lower than those obtained by FS.

- Regardless of the method of selection or weighting, the number of zero features appears to be influenced by only the number of values that weights can take. For example, using six discrete values, but in two different configurations, (0, 0.2, 0.4, 0.6, 0.8, and 1) and (0,1,2,3,4, and 5), produces almost similar numbers of zero features: 8 and 9, respectively.

All the points above suggest that, generally, the greater the total number of weight values, the less likely it will be that any of the features will have zero weights. Whether this apparent relationship is strictly proportional or not is investigated further below. Using data from Table 1 and Table 2 the relationships between the number of zero features and both the number of weight values and the probability of zero weight has been drawn. These relationships are depicted in Figure 5 (a and b).

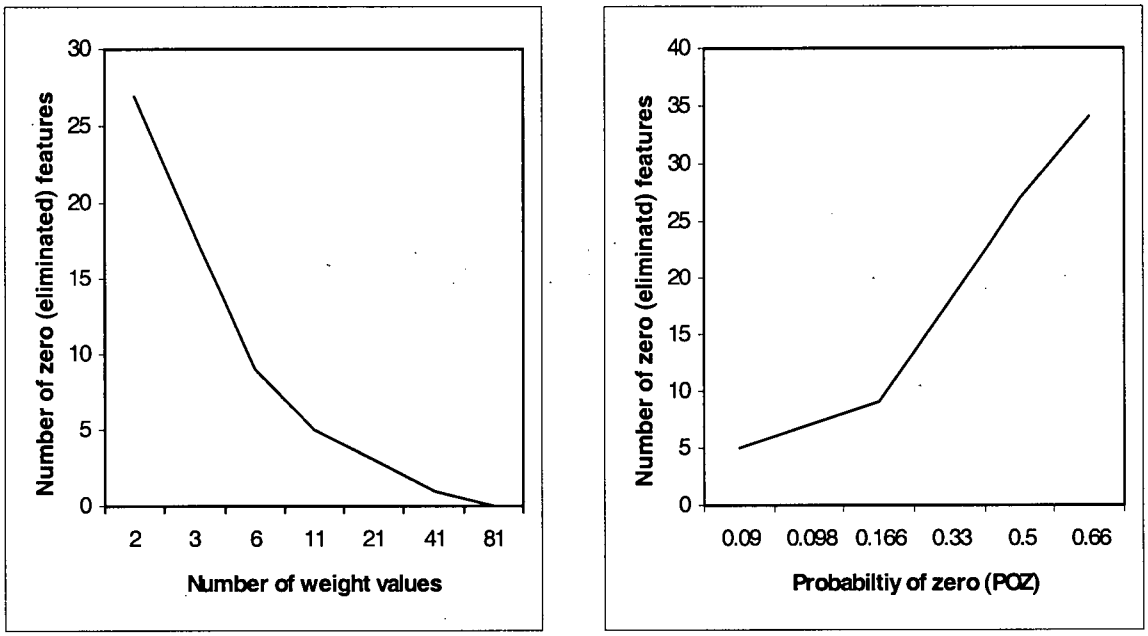


Figure 5: (a) The relationship between number of weight values and actual number of zero (eliminated) features. (b) The number of zero (eliminated) features as a function of the probability of zero.

Figure 5a represents the empirical relationship between the number of weight values and the actual number of zero (or eliminated) features. The relationship is close to an inversely proportional one. It further appears, from Figure 5b that the relationship between the *'probability'* of zero features and the *actual* number of zero features (i.e. eliminated features) is roughly linear, though not strictly proportional. These two figures represents credible evidence that (a) the number of eliminated features is a function of, mainly, the number of values a weight can assume; and hence (b) that the main reason behind the superiority of feature selection over feature weighting (in eliminating features) is the smaller number of weight values FS uses.

In conclusion, it is possible to state that feature selection is clearly superior to feature weighting in terms of feature reduction and without compromising the classification rates.

The main reason for this superiority appears to be the smaller number of weight values that feature selection uses (2 weight values), compared to feature weighting (potentially infinite weight values). However, it is possible to make feature weighting as effective as feature selection in eliminating features (but still feature selection has better classification rates than feature weighting) via, for example, the forcing of all weights less than a given threshold to nil.

5.6.3 Performance of both Genetic Feature Selection and Weighting in the Presence of Irrelevant Features (Comparison 2)

For most classification problems, relevant features are not known in advance. Therefore, many more features than necessary could be added to the initial set of candidate features. Many of these features can turn out to be either irrelevant or redundant. Kohavi & John (1996) define two types of relevant features. They state that features are either *strongly* relevant or *weakly* relevant, otherwise they are *irrelevant*. A strongly relevant feature is one that cannot be removed without degrading the prediction accuracy of the classifier (in every case). A weakly relevant feature is a feature that *sometimes* enhances accuracy, while an irrelevant feature is neither strongly nor weekly relevant. Irrelevant features lower the classification accuracy while increasing the dimensionality of the problem. As a result, removing irrelevant features by either feature selection or weighting is required.

Wettschereck et al. (1997) claim that domains that contain either a) equally relevant features or b) completely irrelevant features, which are most suited to feature selection, feature weighting might outperform feature selection. We intend to investigate this claim by comparing the performance of GFS to that of GFW in the presence of irrelevant and (later in section 5.6.5) redundant features. It is important to indicate that Wilson & Martinez (1996)

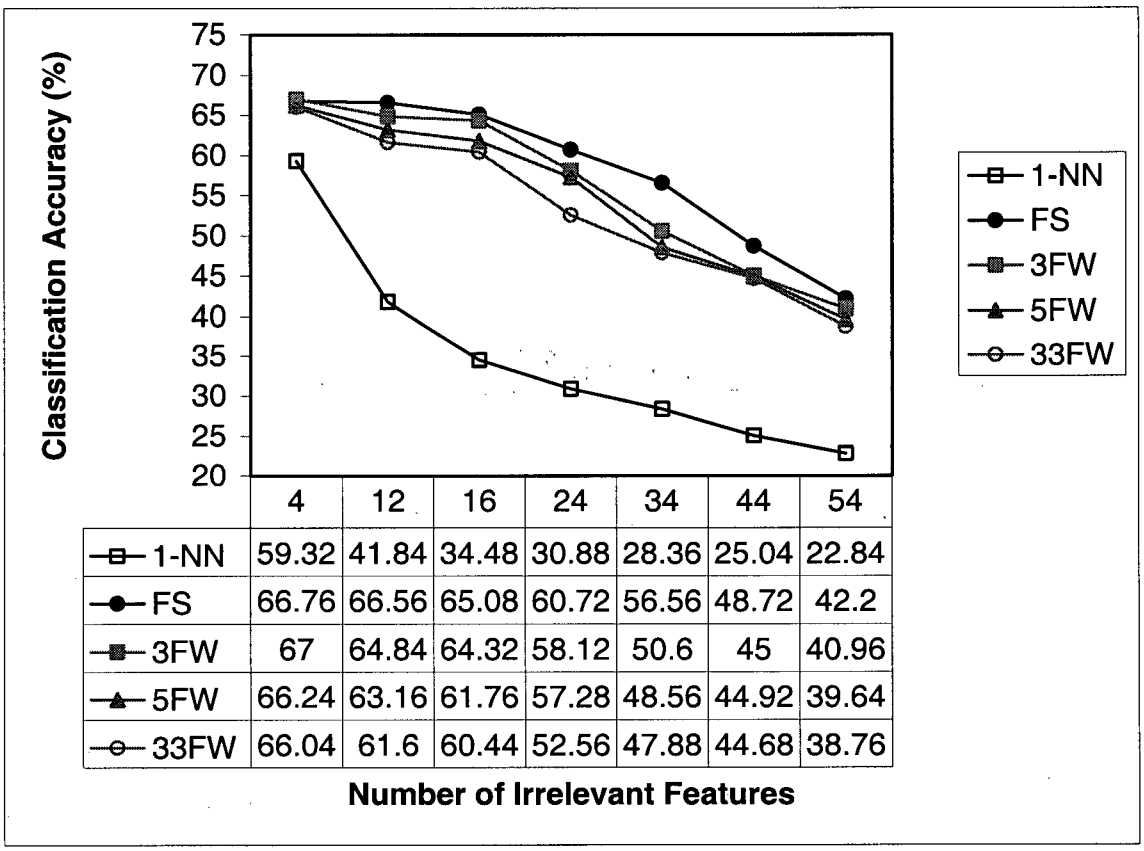
compare the performance of genetic feature weighting GFW with the non-weighted 1-NN for domains with irrelevant and redundant features. They use GA to find the best possible set of weights, which gives the highest possible classification rate. In the presence of irrelevant and redundant features, they show that GFW provides significantly higher results than the non-weighted algorithm. However, they do not compare GFW to GFS for classification tasks with irrelevant or redundant features. Therefore, we are presented with a good chance to see how far genetic feature selection tolerates irrelevant (and redundant features in section 5.6.5), as opposed to genetic feature weighting.

In this experiment we use the dataset that contains 6 features within DB3. We gradually add irrelevant features, which are formed by assigning them uniformly distributed random values. We observe the classification accuracy for GA-based feature selection, GA-based feature weighting, as well as a 1-NN classifier (unaïded by any kind of FS or FW). During GA evaluation, we use the random sub-sampling method of error estimation (see section 2.8.2.2). The samples are split into three sets, a training set, a testing set, and a validation set. The training samples are used to build the 1-nearest neighbor classifier, while the testing samples are used during GA optimization. After GA optimization finishes, a separate validation set is used to assess the weights (produced by the GA optimization). The number of training samples is 1000, the number of testing samples is 500, and the number of validation samples is 500. To avoid any bias due to random selection of the validation set, the train, test and validation samples are completely different. Also, each validation sample is different from the test sample and was never used during the GA run. In addition, this random partitioning is stratified, meaning that all the classes (character classes) are equally represented (i.e. the number of training/testing/validation samples for each class are equal).

The random sub-sampling process is repeated 5 times, and the accuracy results reported represent average values of 5 runs.

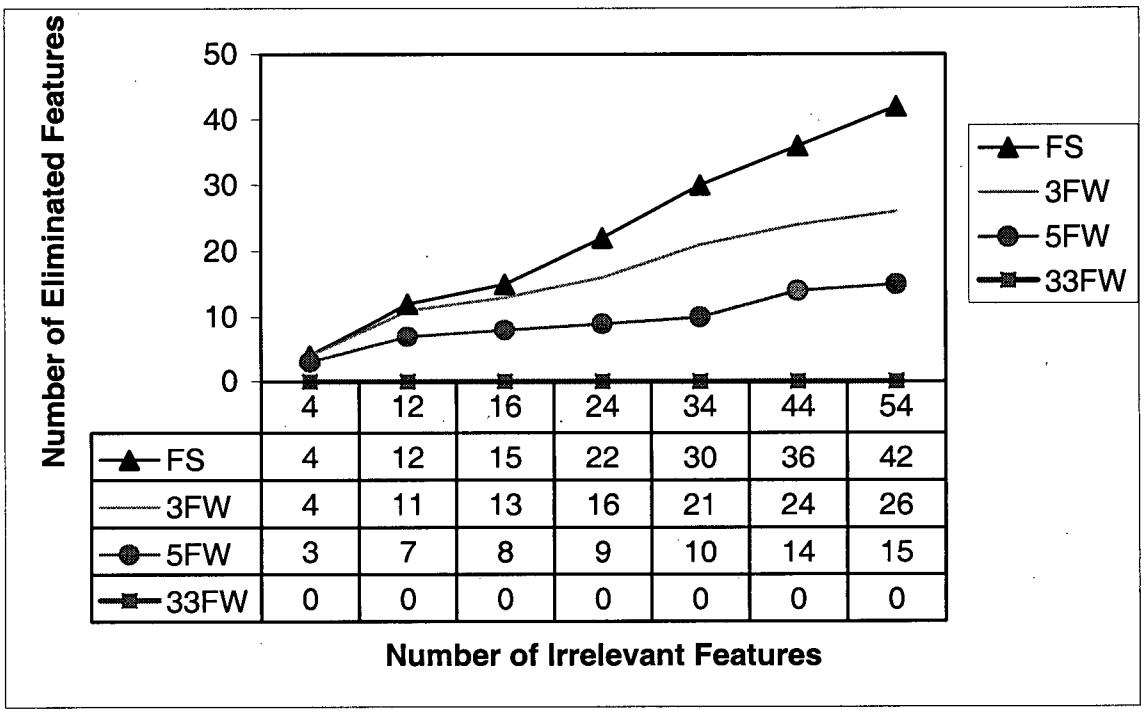
5.6.4 Results of Comparison 2

The results of experimentation are shown in Figure 6 and Figure 7 below. Figure 6 represents classification accuracy of validation (of the various selection and weighting schemes) as a function of the number of irrelevant features included in the initial set of features. Figure 7 represents the number of eliminated features as a function of irrelevant features. In the figures, FW3 stands for feature weighting using 3 discrete equidistant weight levels (0,0.5, 1), FW5 stands for FW with 5 discrete equidistant weight levels, while FW33 stands for FW with 33 discrete equidistant weight levels.



	4	12	16	24	34	44	54
1-NN	59.32	41.84	34.48	30.88	28.36	25.04	22.84
FS	66.76	66.56	65.08	60.72	56.56	48.72	42.2
3FW	67	64.84	64.32	58.12	50.6	45	40.96
5FW	66.24	63.16	61.76	57.28	48.56	44.92	39.64
33FW	66.04	61.6	60.44	52.56	47.88	44.68	38.76

Figure 6: Classification accuracy as a function of the number of irrelevant features.



	4	12	16	24	34	44	54
FS	4	12	15	22	30	36	42
3FW	4	11	13	16	21	24	26
5FW	3	7	8	9	10	14	15
33FW	0	0	0	0	0	0	0

Figure 7: Number of eliminated features as a function of the number of irrelevant features.

From Figure 6 and Figure 7 we can conclude the following:

- As the number of irrelevant features increases, the classification accuracy of the 1-NN classifier *rapidly* degrades, while the accuracies attained by FS and FW *slowly* degrade. Therefore, nearest neighbor algorithms need feature selection/weighting in order to eliminate/de-emphasize irrelevant features, and hence improve accuracy.
- As the number of irrelevant features increases, FS *outperforms* every feature weighting configuration (3FW, 5FW and 33FW), with respect to both classification accuracy and elimination of features. However, when the number of irrelevant features is only 4, 3FW returns slightly better accuracies than FS. This changes as soon as the number of irrelevant features picks up.

- When the number of irrelevant features is 34, the difference in accuracy between FS and the best performing FW (3FW) is *considerable* at 6%.
- As the number of weight levels increases, the classification accuracy of FW, in the presence of irrelevant features, *decreases*.
- As the number of weight levels increase, the number of eliminated features *decreases*.
- When the number of irrelevant features reaches 54, the classification accuracy of FS drops to a value *comparable* to that of other FW configurations. However, the number of eliminated features by FS continues to *outperform* any other FW configuration.
- The gap in the number of features eliminated by FS compared to FW and 1-NN *increases* as the number of irrelevant features increases.

Why does genetic feature weighting perform worse than genetic feature selection in the presence of irrelevant features? Giving the same number of generations for both GFS and GFW, it is hard to explore a space of real valued weights in R^d when d is large. R is the number of real-valued weights, and d is the number of features. However, for the case of feature selection, the search space is 2^d in size. For example, with 10 irrelevant features, the search space for FS is 2^{10} (=1024) compared to 11^{10} (=25937424601) for FW using 11 weight values. Moreover, a single increase in the number of weight levels from 2 to 3 increases the size of the search space form 2^{10} (=1024) to 3^{10} (=59049). As witnessed earlier, as the number of weight levels increases, the number of eliminated features decreases. Therefore, GFS will always eliminate more features than GFW, given the same computational resources and the same number of generations. In addition, in applications

with large sets of features (which implies a high degree of irrelevancy/redundancy among features), it has been shown that feature selection had better results than IB4, an on-line feature weighting algorithms (Aha & Bankert, 1994).

We conclude that in the presence of *irrelevant* features, feature selection and *not* feature weighting is the technique *most* suited to feature reduction. Furthermore, it is *necessary* to use some method of feature selection before a 1-NN or similar nearest-neighbor classifier is applied, because the performance of such classifiers degrades *rapidly* in the presence of irrelevant features. Since it has been shown that GA-based feature selection is effective in eliminating irrelevant features (here and in the literature), it seems sensible to try a GA (at least) as a method of feature selection before classification is carried out using nearest-neighbor classifiers.

5.6.5 Performance of both Genetic Feature Selection and Weighting in the Presence of Redundant Features (Comparison 3)

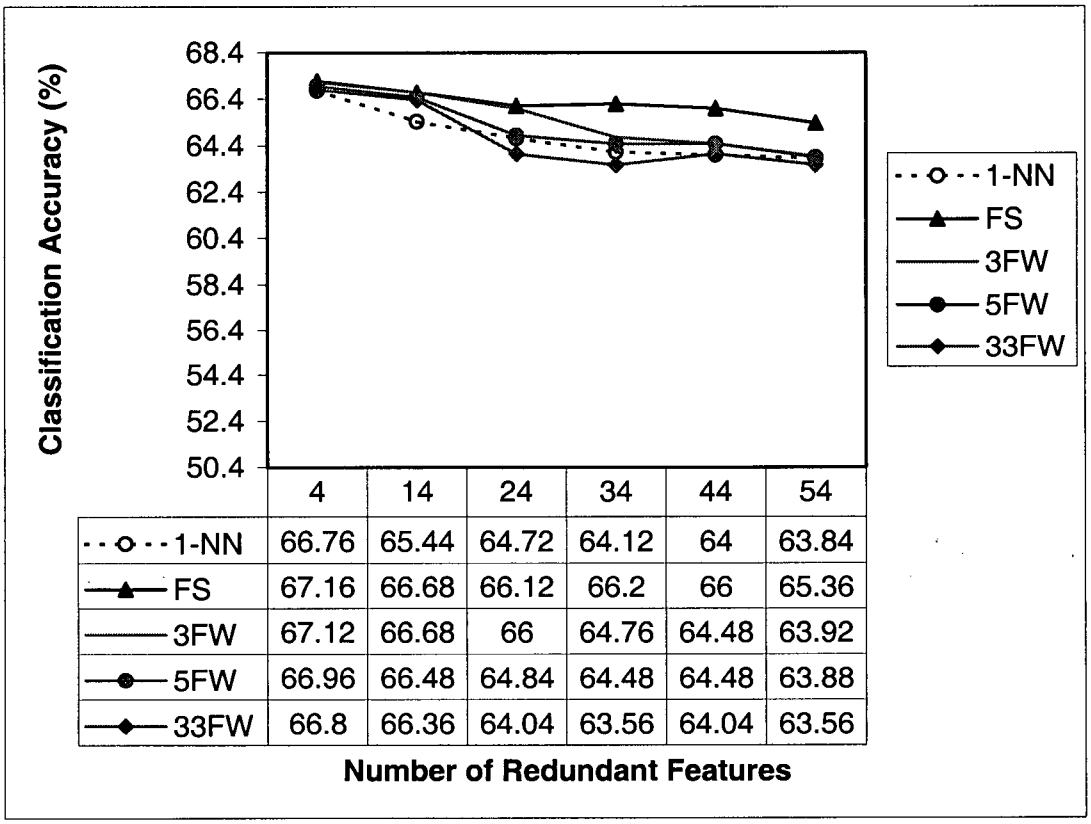
In classification tasks, redundant features add nothing new to the target concept (Dash & Liu, 1997). A redundant feature is a feature, which its value can be extracted from other features values, for example if its value is the average or square or even multiple of other feature values (Wilson & Martinez, 1996). Like irrelevant features, redundant features have the same drawbacks of accuracy reduction and dimensionality growth. Although redundant feature add nothing new, their presence in classification increase the dimensionality of the problem and hence assist in reducing the accuracy (see section 3.1). Therefore, removing redundant features by either feature selection or weighting is required. As mentioned before in section 5.6.3, no comparison exists between GFW and GFS for classification tasks with redundant

features. Therefore, we intend to see how far genetic feature selection tolerates redundant features, as opposed to genetic feature weighting.

In this experiment we used the dataset with 6 features in the database DB3. We randomly selected one feature from the dataset, and repeatedly added this features several times. We observed the accuracy of classification for GA-based FS, GA-based FW, as well as the unaided 1-NN classifier. The error estimation method is the same as that used in section 5.4.3 (above).

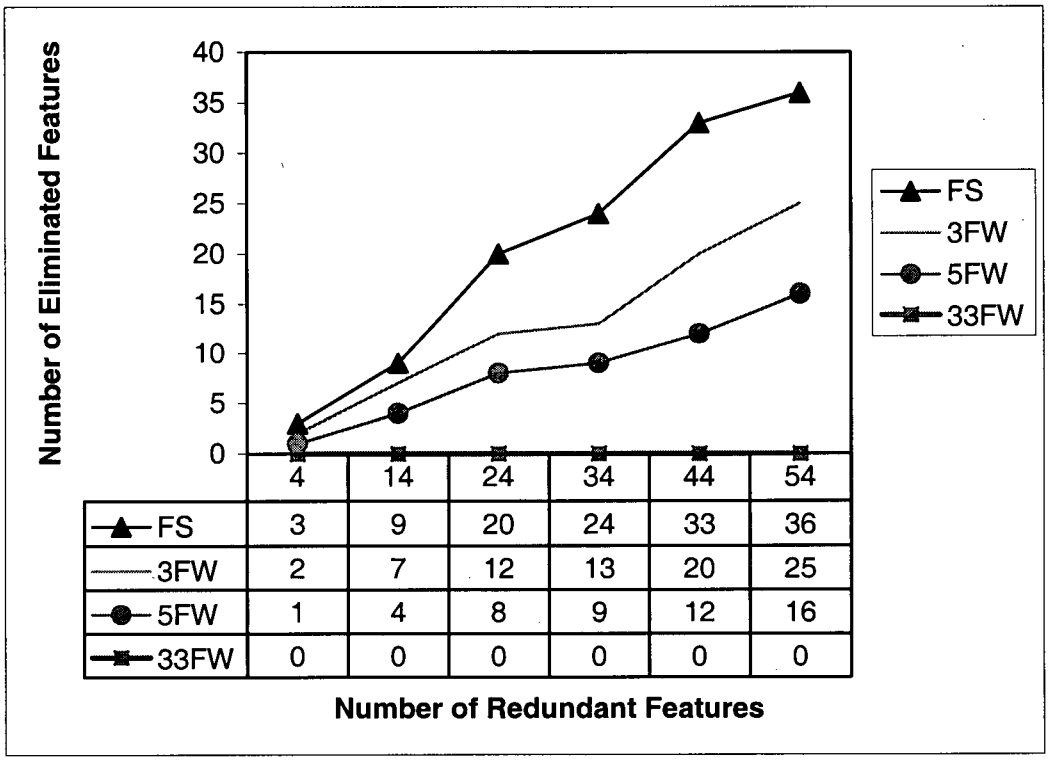
5.6.6 Results of Comparison 3

The results of experimentation are shown in Figure 8 and Figure 9. Figure 8 shows the empirical relationship between the number of redundant features, present in the initial set of features, and the validation classification accuracy of the 1-NN classifier, acting alone, and with the help of GA-based feature selection/weighting. Figure 9 presents the relationship between the number of redundant features, and the number of features eliminated by FS and FW. A 1-NN, on its own does not eliminates any features, of course.



	4	14	24	34	44	54
1-NN	66.76	65.44	64.72	64.12	64	63.84
FS	67.16	66.68	66.12	66.2	66	65.36
3FW	67.12	66.68	66	64.76	64.48	63.92
5FW	66.96	66.48	64.84	64.48	64.48	63.88
33FW	66.8	66.36	64.04	63.56	64.04	63.56

Figure 8: Classification accuracy as a function of the number of redundant features.



	4	14	24	34	44	54
FS	3	9	20	24	33	36
3FW	2	7	12	13	20	25
5FW	1	4	8	9	12	16
33FW	0	0	0	0	0	0

Figure 9: Number of eliminated features as a function of the number of redundant features.

From Figure 8 and Figure 9, the following observations can be drawn:

- Contrary to the case of irrelevant features, the classification accuracy of the 1-NN classifier degrades *very gradually*, as a function of the number of redundant features.
- In experiments with 4, 14 and 24 redundant features, FS classification accuracies are almost *identical* to those achieved by 3FW. However, for the same number of redundant features, the other feature weighting configurations returned classification accuracies that are *worse*, but only *slightly*.

- FS *outperforms* all other FW configurations in experiments where the number of redundant features is 34 or greater.
- As the number of weight levels *increases*, the classification accuracy of the feature weighting configurations slightly *decreases*.
- Although the difference in classification accuracy between FS and FW is not very large, the *difference* in terms of the number of eliminated features (shown in Figure 9) is *significant*.
- The *difference* in performance between FS and FW *increases* as the number of *redundant* features climbs. Finally, when the number of redundant features reaches 54, FS succeeds in eliminating 34 redundant features, 11 more than the best performing FW configuration (3FW).

Why does not the performance of the 1-NN classifier quickly degrade as the number of redundant features increases? The reason is that adding copies of an existing feature repeatedly to a set of features (to act as redundant features) is like giving that feature an added weight equal to the number of copies. For example, if a certain feature is added 20 times, this has the same effect as using this feature once, but multiplied by a weight of 20. On the other hand, completely irrelevant features add randomness and noise to the feature set, which cause higher classification error rates. Though the presence of redundant features in the training data might not significantly decrease the accuracy of the classification algorithm, it will certainly worsen the problem of dimensionality. Moreover, giving a certain feature higher weight simply because it was repeated is considered to be a random procedure (Wilson & Martinez, 1996).

In conclusion, the classification accuracy of a 1-NN classifier does not suffer greatly as a result of having a significant number of redundant features in the (original) feature set. However, due to the other problems associated with redundant features increased dimensionality (and hence computational cost), and arbitrariness of procedure, as well as slightly worse classification accuracies, it is recommended that a GA-based feature selection method be used to eliminate as many of the redundant features as possible. It is clear that the best-suited feature selection/weighting methods for such a task are FS and 3FW.

5.6.7 Performance of both Genetic Feature Selection and Weighting with Regular Databases (Comparison 4)

Wettschereck et al. (1997) state that feature weighting (GA-based or not) is more suitable than feature selection for domains where features have varying degrees of relevance. Naturally, we expect that for regular datasets (not necessarily having redundant or irrelevant features) genetic feature weighting would outperform genetic feature selection in classification rates. This is because FW assigns weights to features that reflect their relative relevance to correct classification. Highly relevant features would be assigned high weights relative to the weights of redundant or irrelevant features. So, FW takes into account the different degrees of feature relevance (strongly relevant, weakly relevant or irrelevant as described in section 5.6.3). In contrast, FS treats the features as either relevant or irrelevant, and does not accommodate for a varying degrees of relevance. However, contrary to what we assumed, we obtained unexpected but justified results (explained in section 5.6.8).

On the other hand, Kohavi et al. (1997) show that increasing the number of weights above two rarely reduces classification error, for many real world datasets. They show that using only two weights (which is equivalent to feature selection) gives better results than

increasing the set of weights. It is worth noting that the results reported in (Kohavi et al. 1997) are for small datasets (with less than 300 training samples), and use a best-first search algorithm. In contrast, we intend to compare feature selection to feature weighting using regular databases (not necessarily having redundant or irrelevant features), with 1000+ training samples, and using GA as a search method.

Presented here is a study of the classification accuracies achieved by the 1-NN classifier and the various GA-based feature selection (GFS) and weighting (GFW) configurations. The focus in this study is on assessing the generalization accuracy for both GFS and GFW on regular databases (not necessarily having redundant or irrelevant features). This requires (a) that real-world databases (not containing human-generated samples) are used, and (b) that any results achieved using training sets are checked against separate results obtained with new (unseen) validation sample sets.

5.6.8 Results of Comparison 4

The resultant accuracies for training and validation sample sets are displayed in Figure 10 and Figure 11. The database used for Figure 10 and Figure 11 was the 6 feature dataset in DB3. For Figure 12 and Figure 13, we used DB2. The error estimation method is the leave-one-out cross validation (described in section 2.8.2.3). It was applied to 1000 training samples. The best weights obtained from the previous training stage are tested using a separate set of 500 validation samples. To avoid bias, we randomly selected different training/validation sets for each experimental run. Each experiment is repeated 5 times, and the results reported are average values for those. The symbols on the X-axes indicate the following:

- 1-NN: 1-nearest neighbor classifier (no GA).

- FS: FS using 2 weights (0,1).
- FW3: FW using 3 weights (0,0.5,1).
- FW5: FW using 5 weights (0,0.25,0.5,0.75,1).
- FW17: FW using 17 weights in [0,1], and an increment value of 0.0625 (1/16).
- FW33: FW using 33 weights in [0,1], and an increment value of 0.03125 (1/32).

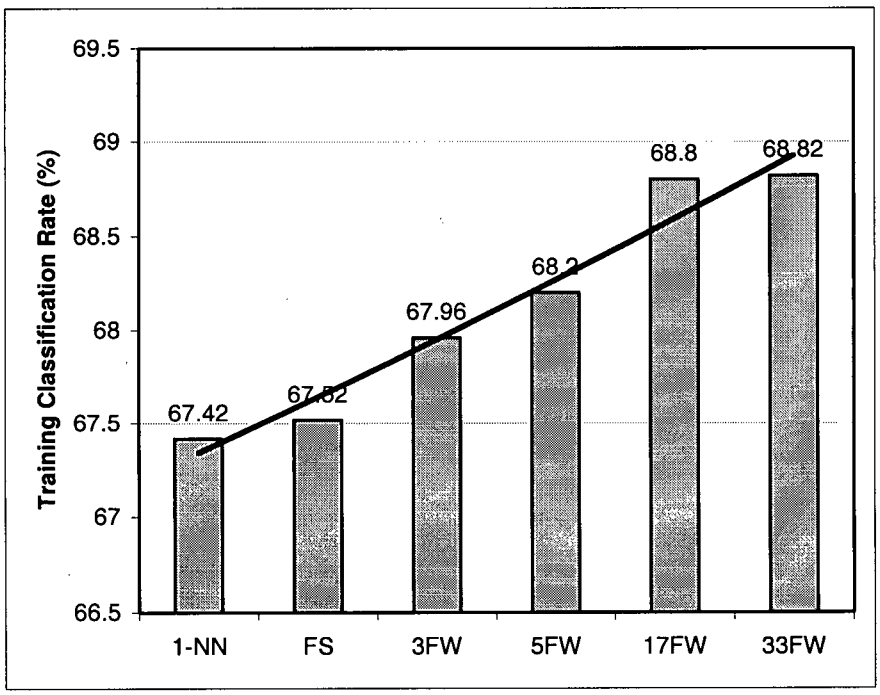


Figure 10: Training classification accuracies for the various feature selection and weighting methods (using DB3).

Results show that classification accuracy is worst for the (unaided) 1-NN, but better for FS and FW. The trend line highlights the fact that the greater the number of weight levels the higher the training accuracy rate achieved. Finally, it appears that classification accuracy could not be improved much by increasing the number of weights beyond 17; the difference in classification accuracy between 17FW and 33FW is 0.02, which is insignificant.

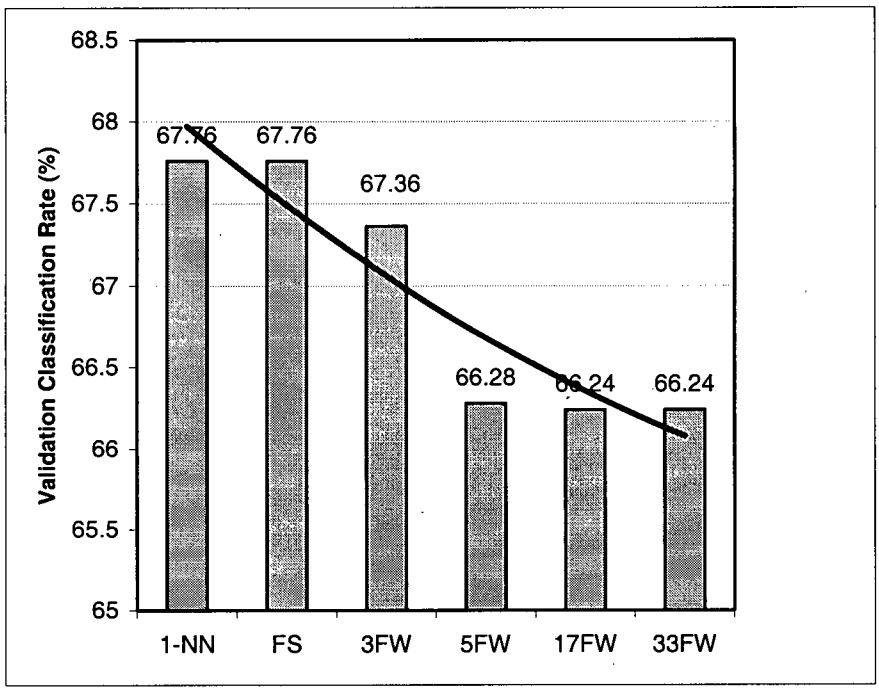


Figure 11: Validation classification accuracies for the various feature selection and weighting methods (using DB3).

Results show that classification accuracy is best for the (unaided) 1-NN and FS, but worse for FW. The trend line highlights the fact that the greater the number of weight levels the lower the validation accuracy rate achieved. Finally, it appears that classification accuracy cannot be further degraded by increasing the number of weights beyond 17; there is no difference in classification accuracy between 17FW and 33FW. These results are the opposite of those found in Figure 10! This means that FW is returning good accuracy results for training data, but then returning bad results for validation data. Also, it is the performance of a classifier on validation data that truly reflects its (real-world) predictive capacity.

To verify these interesting results, we ran the same set of experiments on another database (DB2). This is to ensure that the above results are independent of the particular

sample database used. The results are shown in Figure 12 and Figure 13. It emerges that the results of these experiments indeed confirm the results of Figure 10 and Figure 11.

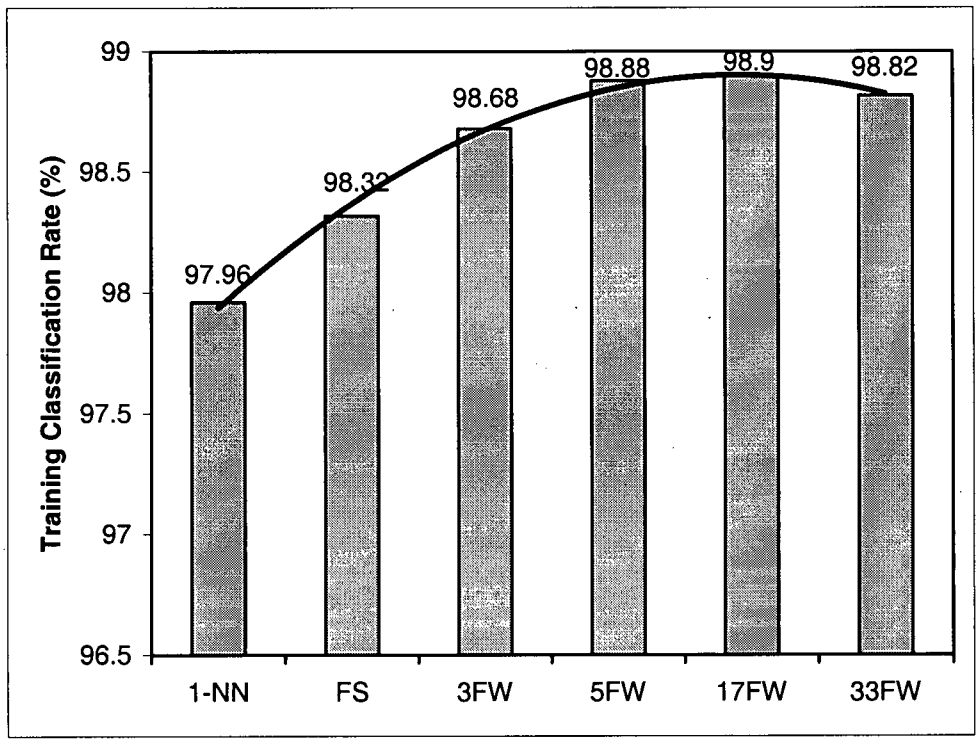


Figure 12: Training classification accuracies for the various feature selection and weighting methods (using DB2).

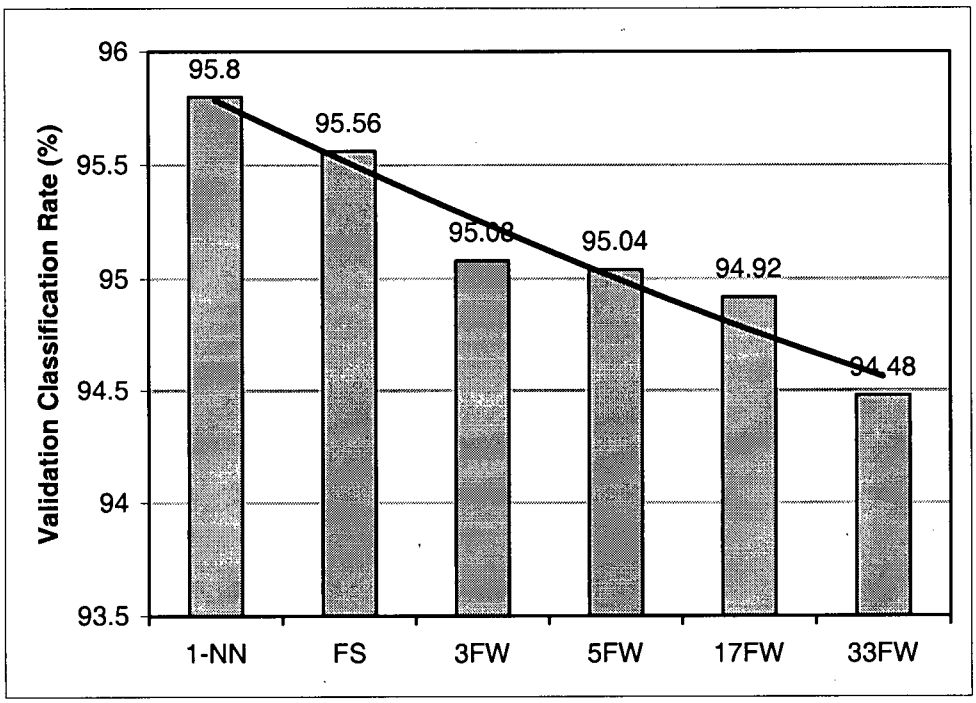


Figure 13: Validation classification accuracies for the various feature selection and weighting methods (using DB2).

Based on the results of Figure 10, Figure 11, Figure 12and Figure 13 we make the following observations.

- For the validation sets, the classification accuracy of FS was slightly better than all the FW settings for both databases. However, it was the same as 1-NN for DB3 and slightly worse for DB2.
- For the training sets all FW settings have better classification accuracy than FS.
- Increasing the number of weight levels led to a slight decrease in the classification accuracy of the validation sets.
- Increasing the number of weights has led to slight increase in the classification accuracy of the training sets.

We believe the interesting disparity between training and validation results is due to over-fitting (which is caused because of the bias-variance tradeoff). Over-fitting means that the learning algorithm adapts so well to a training set but it predictions on new samples are poor (the performance on the validation set is bad) (Duda, Hart & Stork, 2000). A classifier can be seen as a learning machine or learning algorithm aiming to estimate correct classes for the given data samples. A classifier with excellent generalization ability is the one that can make good predictions for the samples that are not in the training set. However, a classifier, which allows perfect classification for the training samples while having poor predictions for the new (unseen) samples, is said to over-fit the training samples.

Over-fitting is related to the bias *vs.* variance tradeoff (Geman, Bienenstock & Doursat, 1992). In general, classification error can be decomposed into two components, *bias* and *variance*. Bias measure how much the error estimation deviates from the true value, whereas the variance measures the variability in classification for different training samples. For a data set having a finite number of samples, there's a trade-off between these two. Increasing the bias decreases the variance and vice versa (Theodoridis & Koutroumbas, 1998). Generally, as the number of parameters of a learning algorithm increases, the classifier will have more flexibility to adapt to the details of the specific training set, hence the bias will reduce but the variance will increase. Conversely, if the number of parameters is few, the classifier will not fit the training data well (high bias) but this fit will not change much for different new samples (low variance) (Duda et al. 2000).

As a result, GFW allows for a finer-grain representation of the search space, but at the expense of an increased classification error rate. So, allowing many weights (for the case of

GFW) will reduce bias but will also increase variance, and hence increase the probability of over-fitting. In contrast, using a small number of weights (which is the case for GFS) will increase the bias due to the lack of representation of the space. However, this will also reduce variance, which in turn reduces the possibility of over-fitting of data (Kohavi et al. 1997). The only way to get a zero bias and zero variance at the same time is to increase the number of training samples very large (possibly infinity) and to have a prior knowledge about the problem (i.e. the shape of the decision boundary). Unfortunately, the number of samples in practice is finite and the prior knowledge about the true model of the problem is not known (Duda et al. 2000). Hence, the best to do is to find the best compromise for the number of parameters, which optimizes the bias-variance trade off. However, obtaining low variance is generally more important, to have accurate classifications, than having low bias (Duda et al. 2000). This is because low variance means better generalization ability and less chances of over-fitting.

In (Kohavi et al. 1997), it is illustrated that increasing the number of weights above two hardly ever decreases classification errors. In their study they show that with 10 weights levels, FW fails to outperform FS on 11 real-world databases. They conclude that, "On many natural data sets, restricting the set of weights to only two alternatives-which is equivalent to feature subset selection-gives the best results". Though we used different search methods and larger datasets than those used by Kohavi et al. (1997), we get similar results and conclusions.

In addition, we should take into account that the databases we are using to report our results are handwritten character recognition databases. This means that there are many variants of digit shape, size, and generally, style. Also, different writers have different

writing styles. For the 10 digits, there are nearly an unlimited number of variations. Thus, feature weighting over-fits the training data by finding suitable weights, but these weights obtained do not generally represent the underlying variations in the handwritten training samples due to the large variance among these samples.

On the other hand, other papers have shown that feature weighting using GA yield a slightly better classification accuracy than feature selection for real-world databases. For example, Punch et al. (1993) perform tests using binary and real-valued weights (with weights in [0,10]). Their database consisted of images of soil samples. Their results show that the error rates obtained using real-valued weights are better than those with binary weights. Also, Komosinski & Krawiec (2000) provide further evidence that feature weighting is somewhat better than selection when applied to a brain-tumor diagnoses system. Their results confirm that feature weighting (using weights in [0,9]) leads to somewhat better classification accuracy rates than just feature selection.

However, several points must be raised here. First, results reported in (Komosinski & Krawiec, 2000) are those of GA evaluation. No validation tests were done to test the resultant weights on separate data sets (i.e. their reported results are based on training only). In fact, Kohavi & Sommerfield (1995) have noticed this fact and states that separate holdout sets, which were never used during the feature selection/weighting, should be used in the final test of performance. Second, results reported in (Punch et al. 1993) were performed on validation samples, which were randomly drawn from the *training* samples. This also has the same biased effect on the results. Third, the improvement in classification accuracies mentioned in both papers for FW over FS is so little that it does not, in itself, provide final evidence that FW gives better results than FS on real datasets. Komosinski & Krawiec (2000)

report classification accuracies of 83.43 ± 1.93 and 77.83 ± 2.03 (for two datasets) for FW, *vs.* accuracies of 80 ± 1.22 and 75.7 ± 1.64 for FS. While Punch et al. reports a classification error rates for FW of 0.83% and 2% for training and validation respectively, those of FS were 1.66 and 3.2% for training and validation.

In conclusion, despite the fact that feature weighting has the best training classification accuracies, feature selection is better in generalization, and hence more suited to real-world applications (in which most data is new). This is because FW overfits the training data, losing generalizability in the process. Therefore, it is advisable to use 2 (FS) or 3 (3FW) weight levels at most, as Kohavi et al. (1997) recommend.

Chapter 6 Genetic Feature Selection Evaluation

In chapters 5 we compared the performance of GA-based feature selection to GA-based feature weighting, within the context of character recognition systems and under various conditions. In this chapter, we intend to evaluate the performance of the better method (which turns out to be genetic feature selection GFS) in term s of optimality and time.

6.1 Introduction

In chapter 5 we have compared genetic feature selection and weighting and showed that GFS outperforms GFW in many aspects such as dimensionality reduction, presence of irrelevant and redundant features and classification accuracy. Therefore, it is important to assess the performance of GFS for both optimality and time. For example, how does GFS perform in comparison with the exhaustive search? To put it in other way, are the solutions obtained from the GFS are optimal (or near optimal) solutions? Also, if GFS is indeed capable of reaching optimal (or near optimal) solutions, what is the number of generations required to reach such optimal? Does increasing the number of features necessitate an increase in the number of generations to obtain optimal solutions?

In fact, we intend to tackle all these questions in this chapter. In the following sections are two empirical studies that study the effectiveness of GA-based feature selection (GFS) with respect to (a) aiming at finding an optimal set of features, and (b) doing so within an acceptable time frame, for off-line applications (e.g., pre-release optimization of character recognition software).

6.2 Convergence of Genetic Feature Selection to an Optimal or Near-Optimal Set of Features (Evaluation 1)

It is clear from the previously described literature that several studies exist comparing GA with other feature selection search algorithms (see section 4.3), so it not our intention to repeat such work. However, there is a need to determine whether or not GFS does indeed return optimal or near optimal feature sub-sets. This necessitates an exhaustive search of the features space in order to find one or more *bona fide* optimum feature sub-set. These can then be used to assess (with certainty) the optimality of the best GFS-generated feature sub-sets.

For the experiments in sections 5.5.2 and 5.5.4, we used a single training/testing set of size 1000 and 500, respectively. There was no need to have a separate validation set in this case, because it is our goal to see whether or not the GFS will reach the optimal values reached by the exhaustive search, not to test the generalization ability of GFS, which was already proven in previous sections. We run both the exhaustive search and GFS using different number of features each time. The numbers of features used in the searches are 8, 10, 12, 14, 16, and 18. We used DB3 with the dataset that contains 6 features, and to obtain the required number of features (8,10,12,14,16,18) we randomly selected 2,4,6,8, and 10 features from the 47 dataset and added them to the 6 feature dataset.

In general, running the exhaustive search would mean to explore the space of 2^d where d is the number of features. However, when d gets large, the exhaustive search becomes computationally prohibitive. For example, having 16 features, this means that we have to try 2^{16}=65536 different feature combinations, while having 18 features would require 2^{18}=262144. When we run the exhaustive search for FS using 16 features it took almost 6 hours, while for 18 features took one and half days. We have only run the

exhaustive search for a maximum of 18 features. However, based on these actual run times, the estimated time to run the exhaustive search for 20 features will be seven and half days, whereas for 47 features the time needed will be 2791 years.

6.3 Results of Evaluation 1

The results presented in Table 3 compare the best and average accuracy rates achieved via GA-based feature selection (GFS) with the optimal accuracy rate found via an exhaustive search of the entire feature space. We run both the exhaustive search and GFS using different number of features each time (8,10,12,14,16,18), which are represented in Table 3 as different rows. For the GA, we repeated the run for 5 times with different seeds and recorded the average and best accuracies achieved during these 5 runs (shown in the third and fourth columns in Table 3). For the exhaustive search, we run it once and reported the best accuracy obtained (shown in the second column in Table 3).

Comparing the numbers in the best GA column with the numbers in the best exhaustive column exhibits the success of the GA as a feature set optimizer. In every case but one (when the number of features is 16), the best accuracy achieved by the GA was identical to that found by the exhaustive search of the feature space. Furthermore, the fact that the worst average GA value is less than 1% percentage point away from the optimal value (i.e. in the row that has 16 features the difference between average GA and optimal value is 0.92) means that the GA was *consistently* able to return optimal or near-optimal values of accuracy. Hence, GA-based feature selection consistently converges to an optimal or near-optimal set of features.

Number of Features	Best Exhaustive (classification rate)	Best GA (classification rate)	Average GA (for 5 runs)
8	74	74	74
10	75.2	75.2	75.2
12	77.2	77.2	77.04
14	79	79	78.56
16	79.2	79	78.28
18	79.4	79.4	78.92

Table 3: Best classification accuracy rates achieved by GA and exhaustive search.

6.4 Convergence of Genetic Feature Selection to an Optimal or Near-Optimal Set of Features within an Acceptable Number of Generations (Evaluation 2)

In general, the run time for the GA is proportional to the number of features, number of generations and size of the population (Kudo & Sklansky, 2000). Moreover, the time complexity of the nearest neighbor classifier is proportional to (t^2 x d), where t is the number of training samples and d is the number of features used. So a GA that is applied in a wrapper configuration (see Figure 2 in section 3.5.1) to a nearest neighbor classifier spends most of its time running the nearest neighbor classifier (Brill et. al. 1992). Below, we list those factors that affect the time complexity of our GFS or GFW:

- Number of generations
- Population size
- Number of features

- Number of training samples

In this evaluation, we will investigate the relationship between the number of features (in the original set) and the number of generations required to reach an optimal or near-optimal sub-set. We will do that while keeping the two other factors (i.e. population size and number of training samples) constant.

6.5 Results of Evaluation 2

The results of experimentation are shown in Table 4. The second column of Table 4 shows the optimal and near optimal (the second best value after the optimal) classification accuracy rates attained using an exhaustive search, whereas the third column displays the duration of that search. The fourth column contains the best results achieved by a GA-based search, while the last column contains the time it took the GA to attain those values.

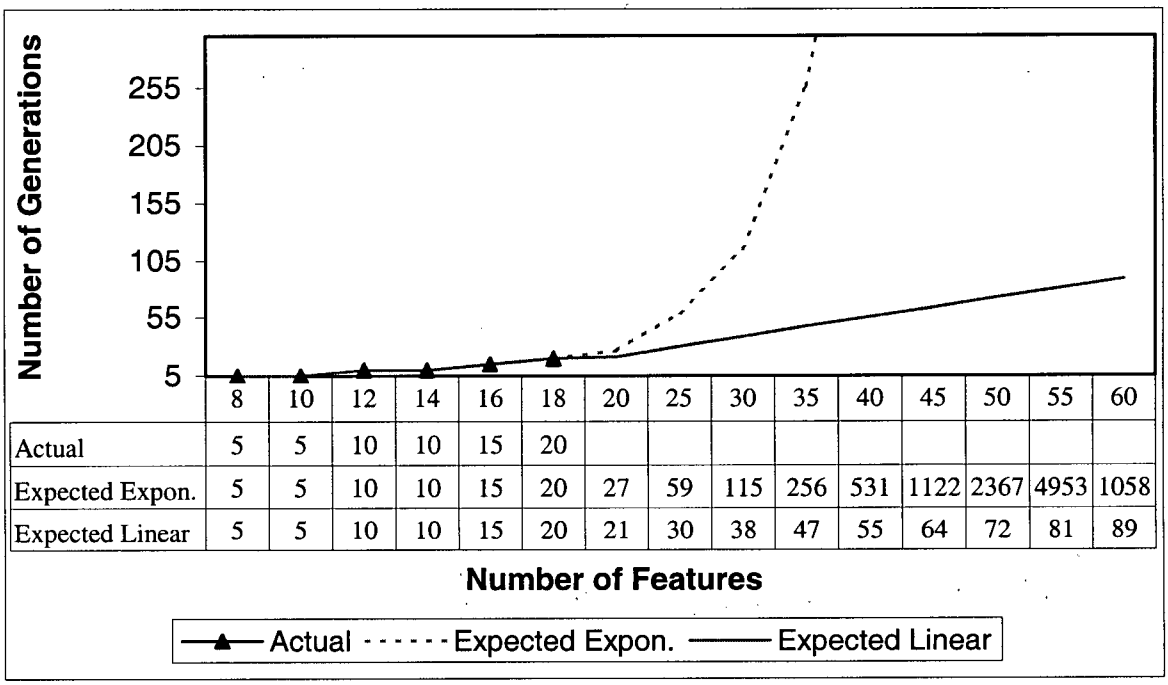
Number of Features	**Best Exhaustive (optimal and near-optimal)**		**Exhaustive Run Time**	**Best GA**	**Average GA (for 5 runs)**	**Number of Generations**	**GA Run Time (single run)**
8	74	73.8	2 minutes	74	73.68	5	2 minutes
10	75.2	75	13 minutes	75.2	74.96	5	3 minutes
12	77.2	77	47 minutes	77	76.92	10	5 minutes
14	79	78.8	3 hours	79	78.2	10	5.5 minutes
16	79.2	79	6 hours	79.2	78.48	15	8 minutes
18	79.4	79.2	1.5 days	79.4	78.92	20	11 minutes

Table 4: Number of generations to convergence.

The following observations can be made:

- In every case, the GA-based FS was able to find the optimal or near to optimal values found by exhaustive search. However, the GA took much less time to find the same optimal or near-optimal values returned by the exhaustive search. With only 8 features, the run times for both exhaustive and GA searches are identical. However, for 18 features, the run time for the exhaustive search was 1.5 days compared to 11 minutes for the GA.
- As the number of features increases, the number of generations required to find the optimal or near-optimal values increases as well.

Figure 14 below displays the relationship between the numbers of generations needed to reach optimal or near-optimal values with a GA, and the number of features in the original set. The tiny triangles represent actual data from Table 4(above). There are also two curves in Figure 14. The solid line represents a linear best-fit curve that fits, and extrapolates, the data points. This solid line is obtained by regression analysis and using the linear best-fit method. This method calculates a straight line that best fits your data. The extrapolated segment of this curve (beyond 18 features) represents the most optimistic projection of GA run time duration. The dotted curve represents an exponential curve that fits, and extrapolates, the same data points. Also using regression analysis, the best exponential curve that fits the data is calculated. The extrapolated segment of this curve (again, beyond 18 features) represents the most pessimistic projection of GA run time duration. The reason for use of extrapolation is the obviously impossible amount of time required to run exhaustive searches of large feature spaces using a single-processor computer. In any case, one can safely conclude that the time needed for a GA-based search is bound, on the lower side by the (optimistic) best-fit curve, and on the upper side, by the (pessimistic) exponential curve.



	8	10	12	14	16	18	20	25	30	35	40	45	50	55	60
Actual	5	5	10	10	15	20									
Expected Expon.	5	5	10	10	15	20	27	59	115	256	531	1122	2367	4953	1058
Expected Linear	5	5	10	10	15	20	21	30	38	47	55	64	72	81	89

Figure 14: Number of generations to convergence as a function of number of features.

Using exponential extrapolation, the number of generations expected for 60 features is 10585, while it is only 89 for linear extrapolation. The true value should be something in between. If we assume that the true value is midway between 89 and 10585 (which is 5337), then running the GA will be computationally expensive. In such cases, using methods of parallel GA, such as DVeGA (Moser, 1999), becomes necessary.

Finally, we conclude that GA-based feature selection is a reliable method of locating optimal or near-optimal feature sub-sets. These techniques also save time, relative to exhaustive searches. However, their effective use in large feature spaces is dependant upon the availability of parallel processors to speed up the GA work.

6.6 Verification Experiments

In this section, some of the experimental work (comparative study and evaluation) described in this chapter and in chapter 5 is repeated using our own extracted features instead of the publicly accessible character databases. This step is necessary to verify and prove the results we obtained earlier.

6.6.1 Pre-Processing and Feature Extraction

The image files that we used for the verification of the results are from the USPS Office of Advanced Technology Database of Handwritten Digits, produced by the Center of Excellence for Document Analysis and Recognition (CEDAR). Using these handwritten images, we have extracted our own features and build a feature set of 40 features. The code used for extracting those features is written using Matlab. We first performed some pre-processing as needed. The aim of pre-processing is an improvement of the image data that suppresses unwanted distortions or enhances some image features important for further processing. The pre-processing steps that we have done are:

1- Noise removal: Remove isolated pixels (1's surrounded by 0's).

2- Image resizing: Get the bounding box of the image (which is the smallest rectangle that can contain the region) and shift the bounding box into the middle of the image so that the image size is 128x128.

3- Thinning (when needed): Remove pixels on the boundaries of objects without allowing objects to break apart (i.e. thinning the image to 1 pixel width). Extracting some features requires the thinned image rather than the whole image.

The features used are shown in Table 5 and explained below.

Seven Hu Moments	Filled Area	Solidity
Area	Convex Area	Orientation
Bounding Box	Number of Holes	Eccentricity
Major Axis Length	Number of End Points	Centroid
Minor Axis Length	Circularity	Eight Extrema Points

Table 5: The extracted feature set from handwritten digits images.

1. Hu Moments: For a function *f(x)*, we can compute the mean value of the function using: $\mu = \frac{\sum_{x=1}^{N} xf(x)}{\sum_{x=1}^{N} f(x)}$. We can also describe the variance by: $\sigma^2 = \frac{\sum_{x=1}^{N} f(x)(x-\mu)^2}{\sum_{x=1}^{N} f(x)}$.

   A third statistical property, called *skew*, describes how symmetric the function is: Skew = $\frac{\sum_{x=1}^{N} f(x)(x-\mu)^3}{\sum_{x=1}^{N} f(x)}$. All of these are examples moments of the function. One can define moments about some arbitrary point, usually either about zero or about the mean. The n-th moment about zero, denoted as m_n, is $m_n = \sum_{x=1}^{N} x^n f(x)$, where m_0 is the total value of the function. The mean μ is the first moment about zero divided by the zero-th moment: $\mu = \frac{m_1}{m_0}$. The n-th moment about the mean, denoted as μ_n and called the n-th central moment is $\mu_n = \sum_{x=1}^{N} (x-\mu)^n f(x)$. The zero-th central moment μ_0 is, again, the total value of the function. The first central moment μ_1 is always 0. The second central moment μ_2, when normalized by the total value

μ_0 is the variance: $\sigma^2 = \frac{\mu_2}{\mu_0}$. The third central moment μ_3, when normalized is the skew: skew=$\frac{\mu_3}{\mu_0}$. The fourth central moment μ_4, when normalized is the kirtosis: kirtosis=$\frac{\mu_4}{\mu_0}$. If we have an infinite number of central moments, we can completely describe the shape of the function. A set of seven invariant moments can be derived from the second and third moments (Gonzalez & Woods, 1992). This set of moments is invariant to translation, rotation, and scale change (Hu moments).

2. Area: The number of pixels in the shape.
3. Bounding Box: A 1-by-4 vector, which represents the smallest rectangle that can contain the region. The format of the vector is [*x y* width height], where *x* and *y* are the *x*- and *y*-coordinates of the upper-left corner of the rectangle, and width and height are the width and height of the rectangle.
4. Major Axis Length: The length (in pixels) of the major axis of the ellipse that has the same second-moments as the region.
5. Minor Axis Length: The length (in pixels) of the minor axis of the ellipse that has the same second-moments as the region.
6. Filled Area: The number of on pixels in filled image.
7. Convex Area: The number of pixels in the convex image
8. Number of Holes: The number of holes in the shape.
9. Number of End Points: The number of end points in the shape.

10. Circularity: Circularity measures the ratio of the perimeter divided by the area ($C=\frac{P^2}{4\Pi A}$ where P is the perimeter and A is the area) (Parker, 1994).
11. Solidity: The proportion of the pixels in the convex hull that are also in the region. It is computed as Area/ConvexArea.
12. Orientation: The angle (in degrees) between the x-axis and the major axis of the ellipse that has the same second-moments as the region.
13. Eccentricity: The ratio of the length of the longest chord of the shape to the longest chord perpendicular to it. This is one way to define it; another way to define it is as follows. The eccentricity of the ellipse that has the same second-moments as the region. The eccentricity is the ratio of the distance between the foci of the ellipse and its major axis length. The value is between 0 and 1. (0 and 1 are degenerate cases; an ellipse whose eccentricity is 0 is actually a circle, while an ellipse whose eccentricity is 1 is a line segment).
14. Centroid: the x and y coordinates of the center of mass of the region.
15. Extrema: 8-by-2 matrix, which contains the extremal points in the region. Each row of the matrix contains the x and y coordinates of one of the points; the format of the vector is [top-left top-right right-top right-bottom bottom-right bottom-left left-bottom left-top]. Figure 15 shows these extremal points.

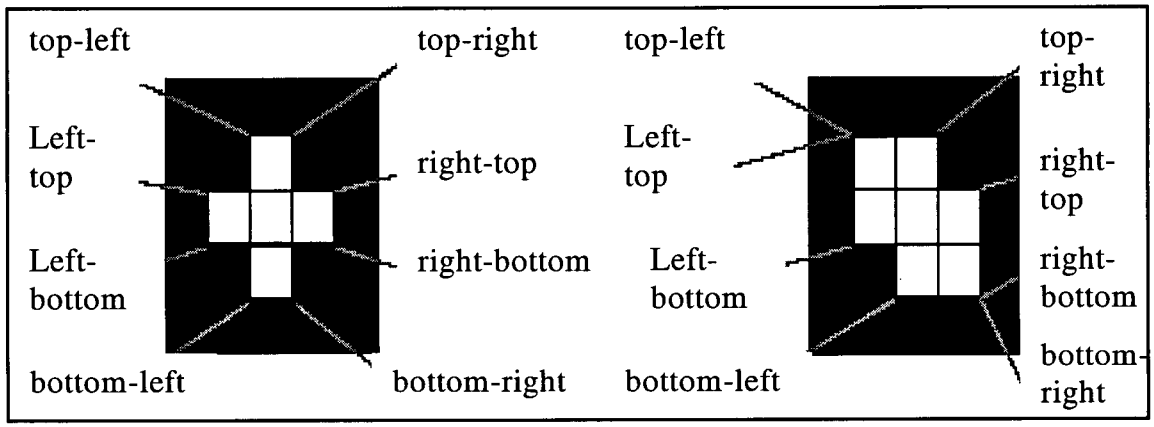


Figure 15: Exteremal points.

6.6.2 Verification of Comparison 1

We have repeated two of the previous experiments done previously using our own database, which contains 40 features. The first experiment that was repeated using our own extracted features is comparison 1 (see section 5.6.1 in chapter 5), which is studying the effect of varying the number of values that weights can take on the number of selected features. The error estimation method used is the leave-one-out cross validation applied to the training data itself. The number of training samples used is 500 and the resultant weights are assessed on a validation set of size 250 that is never used before during training. In addition, we have performed random partitioning for the data samples. We randomly selected different training/validation samples each time provided that the train and validation samples are completely different. Also, the random partitioning is stratified, meaning that all the classes (digit classes) are equally represented (i.e. the number of training/validation samples for each class are equal). This process was repeated five times and the average accuracy is calculated. The average results are shown in Table 6.

Method of Selection/ Weighting	Increment Value	Number of Levels	Probability of Zero	Accuracy of Training	Accuracy of Validation	Number of Zero Features
1-NN	-	-	-	70.48	70.96	-
FS	1	2	½	77.68	74.72	21
3FW (three discrete values: 0, 0.5, 1)	0.5	3	1/3 (0.33)	78.32	73.6	14
5FW (five discrete values: 0, 0.25, 0.5, 0.75, 1)	0.25	5	1/5 (0.2)	78.44	73.6	9
33FW	0.03125 (1/32)	32+1	1/33	78.16	73.36	0

Table 6: Recognition accuracy and number of zero features for various selection and weighting schemes.

Though we only intended by repeating this experiment to prove the previously obtained results from comparison 1 (see section 5.6.2), the results shown in Table 6 illustrate the followings:

- Comparing the results of the 1-NN to the FS and the FW methods we find that a lot of features were eliminated, while the classification accuracy increased. This means that there are many irrelevant features and eliminating those features helped in increasing the classification rate.
- As previously shown in section 5.6.2, we can notice from the table that FS far outperforms FW in terms of the number of zero features. As the number of values a weight can take increase, the number of eliminated features decrease. These results are consistent with those previously obtained in comparison 1 in section 5.6.2.

- FS is better than the other feature weighting settings (3FW, 5FW and 33FW) in terms of the number of eliminated features. Moreover, FS has a somewhat better validation classification accuracy than FW. This result is consistent with the results of comparisons 2 (described in section 5.6.4), which deals with the usefulness of FS in the presence of irrelevant features.

6.6.3 Verification of Evaluation 1

The second experiment that we repeated using our own extracted features is evaluation 1, which is the study of the convergence of GFS to an optimal or near-optimal set of features. Like we did before in evaluation 1 (in sections 6.2 and 6.3), we run the exhaustive search to explore the space of 2^d where d is the number of features and recorded the optimal solution and near optimal reached by the exhaustive search. Also, we run the GFS five times and reported the best and average accuracies obtained. We used a single train/test set of size 500/250 and the number of features used were 16,18 and 20, for both exhaustive and GA searches.

Running the exhaustive search for the whole 40 feature set would be computationally impossible. Therefore, we choose smaller subsets of 20,18 and 16 features out of the whole 40-feature set, to run the exhaustive search. We have only run the exhaustive search for a maximum of 20 features. However, based on the actual run times, the estimated time to run the exhaustive search for 22 features will be 150 hours (6.25 days). The results shown in Table 7 where the maximum classification accuracy obtained using the exhaustive search is recorded. In addition, the best and average values reached by GFS are also shown.

Number of features	Best Exhaustive (optimal and near-optimal)		Exhaustive run time	Best GA	Average GA (for 5 runs)	No of generations	GA run time (single run)
16	75.2	74.8	2:30 hours	75.2	74.08	15	1 1/2 minutes
18	75.2	74.8	10:45 hours	75.2	73.84	15	1 min. 40 sec.
20	75.2	74.8	2 days	75.2	74.24	20	2 1/2 minutes

Table 7: Classification accuracy rates achieved by GA and exhaustive search.

Looking at the results in Table 7, we can see the following:

- The optimal values reached by the exhaustive searches for all the 16, 18 and 20 features are the same. This confirms our previous observation that some of the features are redundant/irrelevant, by which eliminating them did not affect the optimal classification rate obtained.
- It is clear that GFS can reach an optimal or near optimal solutions, which were found by the exhaustive search. These results are consistent with the previously obtained results from evaluation 1 in section 6.3.

Finally, it is important to note that the classification accuracies obtained using our extracted 40 features are around 70% using 1-NN and around 75% after GFS. Though these rates are not high for a typical handwritten digits recognition task, they were satisfactory for us, since our main purpose was to prove that GFS could reach the optimal solutions reached by the exhaustive search. Enhancing the classification accuracy for our

extracted features may require the addition of more features and applying more pre-processing steps such as slant and slope corrections.

Chapter 7 Conclusions and Future Research

7.1 Conclusions

The objective of the research was to apply genetic algorithms for the problem of feature weighting for character recognition application. In addition, we expected that because feature weighting is the general case of feature selection, it should perform better than feature selection, at least in some situations. So, after the employment of GFW to character recognition application, we were interested in comparing the performance of both GFS and GFW also in the context of character recognition applications to test the validity of this hypothesis. To achieve these objectives we built a pattern recognition experimental bench that contains a genetic-based feature selection and weighting module. Then we carried out two sets of studies, which in turn produced some unexpected but justified results.

The first set compares the performance of Genetic Algorithm (GA)-based feature selection to GA-based feature weighting, under various conditions. The second set of studies evaluates the performance of the better method (which turned out to be feature selection) in terms of optimal performance and time. The results of these studies show the superiority of GFS over GFW in terms of a) the number of eliminated features, as well as b) recognition accuracy, in situations where irrelevant or/and redundant features are present. Nevertheless, GFS succeeds in finding optimal or near-optimal solutions, in all experiments. In addition, results show that GA is an effective method for feature selection. However, their scalability to highly dimensional problems, in practice, is still

an open problem. The following sections summarize the lessons learnt from this research effort.

7.1.1 Genetic Feature Selection verses Genetic Feature Weighting

Genetic feature selection is clearly superior to genetic feature weighting in terms of feature reduction. The main reason for this superiority appears to be the small number of weight values that feature selection uses, which is only 2 (zero and one), compared to the number of weight values used by feature weighting (potentially infinite).

In the presence of *irrelevant* features, feature selection and *not* feature weighting is the technique *most* suited to feature reduction. Furthermore, it is *necessary* to use some method of feature selection before a 1-NN or similar nearest-neighbor classifier is applied, because the performance of such classifiers degrades *rapidly* in the presence of irrelevant features. Since it has been shown that GA-based feature selection is effective in eliminating irrelevant features, it is reasonable to try a GA (at least) as a method of feature selection before classification is attempted using nearest-neighbor classifiers.

The classification accuracy of a 1-NN classifier does not suffer so much as a result of having a significant number of redundant features in the (original) feature set. However, due to the other problems associated with redundant features, such as increased dimensionality (and hence computational cost), and the arbitrariness of procedure, as well as slightly worse classification accuracies, it is recommended that a GA-based feature selection method be used to eliminate as many of the redundant features as possible. It is clear that the most suited feature selection/weighting methods for such a task are FS and 3FW.

Despite the fact that feature weighting has the best training classification accuracies, feature selection is better in generalization (i.e. make better predictions for the samples that

are not in the training set than feature weighting), and hence more suited to real-world applications (in which most data is new). This is because FW over-fits the training data thus, resulting in reduced predictions (than feature selection) for the new samples. Therefore, it is advisable to use 2 (FS) or 3 (3FW) weight levels at most, as Kohavi et al. (1997) recommend.

7.1.2 Performance of Genetic Feature Selection

Genetic feature selection is a reliable method of locating optimal or near-optimal feature subsets. These techniques also save time, relative to exhaustive searches. The question of how well our method will scale-up to highly dimensional feature spaces remains an open problem. However, their effective use in large features spaces is dependant the availability of parallel processors.

7.2 Future Research

We present below a list of research problems, carefully justified, that we believe future research in GFS should address. It is our belief that solutions to such problems will help with the automation of genetic feature selection/weighting in pattern recognition applications.

- **Research Problem 1.** Feature selection, and therefore feature weighting, is NP-complete. Hence, although feature selection has shown very promising results, practical applications are limited by the dimensionality of the solution search space. Moser [Moser and Murty 00] examined the scalability of Distributed Vertical Genetic Algorithms (DVeGA) to very large-scale feature selection applications with more than 500 features. His application succeeded in reducing the dimensionality while simultaneously maintaining high accuracy. Crucially, Moser's "experiments showed that GA scale well to domains of large complexity in feature selection" [Moser and Murty 00]. So a possible

future research is to try and use the idea of distributed genetic algorithms (or parallel processing in general) in the way we implement our own GFS/GFW optimizer to make highly dimensional feature spaces feasible.

- **Research Problem 2**. A possible future research is to understand how to generalize the lessons gained from the successful application of the GFS optimizer (to a particular symbol set) to new and different symbols sets. Examples of symbols sets that can be tried are hand-written English characters, mathematical notations and any (pre-segmented) black & white or gray-scale 2D line drawing. Indeed, the real power of the GFS optimization approach we are proposing will not be fully realized until the experimental bench starts working successfully with different symbol sets. Furthermore, it should do so without recourse to extensive and lengthy trial and error tuning. This will help in building and configuring a character recognition software product tailored for a specific symbol set, and with minimal help from pattern recognition experts.

- **Research Problem 3.** We have shown that FS, in general, is superior to FW in terms of the number of eliminated features, as well as accuracy of recognition, especially in cases where irrelevant/redundant features are present in the original feature set. However, other combinations of FS and FW could perform better than FS or FW alone. For example, Raymer, Punch, Goodman, Kuhn & Jain (2000) have applied simultaneous feature weighting and selection using genetic algorithm via a masking technique. They obtained better results on validation samples than with FW alone and state that operating FS and FW simultaneously allow the GA to find better interactions between features than just operating FS and FW independently (Raymer et al., 2000). Hence, researchers may wish

to investigate further that approach by comparing the simultaneous feature weighting and selection suggested in (Raymer et al., 2000) and our method of FS and FW that are operating independently. In addition, researchers could also apply different weighting schemes, such as local weighting, and combine both global and local weightings to compare FS and FW.

- **Research Problem 4.** Referring to section 4.3, there appears to be no record of any study, which compares the performance of GA with simulated annealing for the feature selection problem. In addition, the only existing work (Zhang & Sun, 2002) that compares GA with tabu search for feature selection problem need to be verified using true classifier error rate and real datasets (See section 4.3 for details). Therefore, a fruitful area for future research is to compare the performance of the three stochastic search algorithms (GA, simulated annealing and tabu search) in the context of feature selection problem.

References

Aha, D. W. (1992). Tolerating noisy, irrelevant, and novel attributes in instance-based Learning algorithms. *International Journal of Man-Machine Studies, 36,* 267-287.

Aha, D. W., & Bankert, R. L. (1994). Feature selection for case-based classification of cloud types: An empirical comparison. In D. W. Aha (Ed.) *Case-Based Reasoning: Papers from the 1994 Workshop* (Technical Report WS-94-01). Menlo Park, CA: AAAI Press. (NCARAI TR: AIC-94-011)

David Beasley, David R. Bull, & Ralph R. Martin. (1993). An Overview of Genetic Algorithms: Part 1, Fundamentals. University Computing, Inter-University Committee on Computing, *15(2),* 58-69.

Blum, A.L., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, pp. 245-271.

Brill, F., Brown, D., & Martin, W. (1992). Fast genetic selection of features for neural network classifiers. *IEEE Transactions on Neural Networks, 3(2),* 324-328.

Dasarathy, Belur V. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, Los Alamitos, CA: IEEE Computer Society Press.

Dash, M., & Liu, H. (1997). Feature selection for classification. *Intelligent Data Analysis, 1,* 131-156.

Davis, L. (1991). *Handbook of GA.* Van Nostrand Reinhold.

De Jong, K. (1975). An analysis of the behavior of a class of genetic adaptive systems. Doctoral dissertation, University of Michigan, Dissertation Abstracts International, 36(10), 5140B, University Microfilms No. 76-9381.

Demiroz, G., & Guvenir, H. A. (1996). Genetic algorithms to learn feature weights for the nearest neighbor algorithm. In *Proceedings of the 6 th Belgian--Dutch Conference on Machine Learning* (BENELEARN--96), pp. 117-126.

Devijver, P., & Kittler, J. (1982). *Pattern Recognition: A Statistical Approach.* Prentice Hall.

Duda, R. O., Hart, P. E., & Stork, D. G. (2000*). Pattern Classification.* John Wiley Interscience.

Estevez, P.A., & Fernandez, M. (1999). Selection of features for the classification of wood board defects. *In Proceedings of the Ninth International Conference on Artificial Neural Networks, 1,* 347-352.

Ferri, M., & Piccioni, M. (1992). Optimal selection of statistical units: An approach via simulated annealing. *Computational Statistics and Data Analysis, 13*, 47 – 61.

Fung, G., Liu, J., & Lau, R. (1996). Feature selection in automatic signature verification based on genetic algorithms. In *Proceedings of International Conference on Neural Information*, pp. 811-815.

Gaborski, R. S., & Anderson, P. G. (1993). Genetic algorithm selection of features for handwritten character identification. In *the International Conference on Artificial Neural Networks & Genetic Algorithms ANNGA 93*, Innsbruck, Austria.

Geman, S., Bienenstock, E., & Doursat, R. (1995). Neural networks and the bias/variance dilemma. *Neural Computation, 4,* 1-58.

Glover F. (1986). Future paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research, 5,* 533-549.

Goldberg DE. (1989). *Genetic algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA.

Gonzalez, R. C., & Woods, R. E. R. (1992). *Digital image processing*. Addison-Wesley.

Handels, H., Ross, T., Kreusch, J., Wolff, H. H., & Poppl, S.J. (1999). Feature selection for optimized skin tumor recognition using genetic algorithms. *Artificial Intelligence in Medicine, 16(3),* 283-297.

Howe, N., & Cardie, C. (1997). Examining locally varying weights for nearest neighbor algorithms. *Case-Based Reasoning Research and Development: Second International Conference on Case-Based Reasoning*, D. Leake and E. Plaza, eds., Lecture Notes in Aritificial Intelligence, Springer, pp. 455-466.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

Hussein, F., Kharma, N., & Ward, R. (2001). Genetic Algorithms for Feature Selection and Weighting, in proceedings of the *Sixth International Conference on Document Analysis and Recognition ICDAR'01*, pp. 1240-1244.

Jack, L.B., & Nandi, A.K. (2000). Genetic algorithms for feature selection in machine condition monitoring with vibration signals. *IEE Proceedings, Vision, Image and Signal Processing, 147(3),* 205-212.

Jain, A., & Zongker, D. (1997). Feature selection: Evaluation, application, and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(2),* 153-158.

Jain, Anil K., Duin, Robert P.W., & Mao, Jianchang (2000). Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(1),* 4-37.

Kelly, J., & Davis, L. (1991). A hybrid genetic algorithm for classification. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pp. 645-650.

Kharma, N., & Ward, R. (1999). Character Recognition Systems for the Non-expert, in *IEEE Canadian Review, 33,* pp. 5-8.

Kharma, N., Hussein, F., & Ward, R.K. (2002a). Using Genetic Algorithms for Feature Selection and Weighting in Character Recognition Systems. *Evolutionary Computation Journal.*

Kharma, N., Hussein, F., & Ward, R.K. (2002b). Genetic Algorithms for Feature Selection and Weighting in Character Recognition Systems, a Summary Review and Comparative Study. *The Sixteenth International Conference on Pattern Recognition*, Québec City, (submitted).

Kim, G., & Kim, S. (2000). Feature selection using genetic algorithms for handwritten character recognition. In *Proceedings of the Seventh International Workshop on Froniers in Handwriting Recognition*, pp. 103-112.

Kohavi, R., & Sommerfield, D. (1995). Feature subset selection using the wrapper method: overfitting and dynamic search space topology. *Proceedings of the First International Conference on Knowledge Discovery and Data Mining, KDD'95,* Montreal, Canada, pp. 192-197.

Kohavi, R., & John, G. (1997) Wrappers for feature subset selection. *In Artificial Intelligence journal, special issue on relevance, 97(1-2),* 273-324.

Kohavi, R., Langley, P., & Yun, Y. (1997). The utility of feature weighting in nearest-neighbor algorithms, *European Conference on Machine Learning*, ECML'97.

Komosinski, M., & Krawiec, K. (2000). Evolutionary weighting of image features for diagnoses of CNS tumors. *Artificial Intelligence in Medicine. 19,* 25-38.

Kudo, M., & Sklansky, J. (2000). Comparison of algorithms that select features for pattern classifiers. *Pattern Recognition, 33(1),* 25-41.

Martin-Bautista, M.J., & Vila, M.A. (1998). Applying genetic algorithms to the feature selection problem in information retrieval. In *Lecture Notes On Artificial Intelligence (LNAI), 1495.* Springer-Verlag.

Matthew, W. (1999). *Galib (2.4.4)* Massachusetts Institute of Technology MIT. http://lancet.mit.edu/ga/

Michalewicz, Z., & Fogel, D. B. (1998). *How to solve it.* Springer-Verlag.

Moser, A. (1999). A distributed vertical genetic algorithm for feature selection. *In Proceedings of the Fifth International Conference on Document Analysis and Recognition,* Open Research Forum.

Moser, A., & Murty, M. (2000). On the scalability of genetic algorithms to very large-scale feature selection. *Real World Applications of Evolutionary Computing: Proceedings, 1803:* 77-86.

Murphy, P., & Aha, D. (1994). Repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, CA. http://www.ics.uci.edu/~mlearn/MLRepository

Pandya, Abhijit S., & Macy, Robert B. (1996). *Pattern recognition with neural networks in C++.* Boca Raton, FL: CRC Press.

Parker, J. R. (1994). *Practical Computer Vision Using C.* John Wiley & Sons, Inc.

Punch, W., Goodman, E., Pei, M., Chia-Shun, L., Hovland, P., & Enbody, R. (1993). Further research on feature selection and classification using genetic algorithms. *In Proceedings of the Fifth International Conference on Genetic Algorithms,* pp. 379-383.

Raymer, M., Punch, W., Goodman, E., Kuhn, L., & Jain, A. (2000). Dimensionality reduction using genetic algorithms. *IEEE Transcations on Evolutionary Computation, 4(2),* 164-171.

Sadiq, S. M., & Youssef, H. (1999). *Iterative Computer Algorithms with Applications in Engineering.* IEEE Computer Society.

Sahiner, B., Chan, H.P., Wei, D.T., Petrick, N., Helvie, M.A., Adler, D.D., & Goodsitt, M.M. (1996). Image feature selection by a genetic algorithm: Application to classification of mass and normal breast tissue. *Medical Physics, 23(10),* 1671-1684.

Shi, D., Shu, W., & Liu, H. (1998). Feature selection for handwritten Chinese character recognition based on genetic algorithms. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, 5,* 4201-4206.

Siedlecki, W., & Sklansky, J. (1988). On automatic feature selection. *International Journal of Pattern Recognition, 2,* 197-220.

Siedlecki, W., & Sklansky, J. (1989). A note on genetic algorithms for large-scale feature selection. *IEEE Transactions on Computers, 10,* 335-347.

Smith, J.E., Fogarty, T.C., & Johnson, I.R. (1994). Genetic selection of features for clustering and classification. *In IEE Colloquium on Genetic Algorithms in Image Processing and Vision.*

Theodoridis, S., & Koutroumbas, K. (1998). *Pattern Recognition.* Academic Press.

Trier, O.D., A.K. Jain, & T. Taxt (1996). Feature extraction methods for character recognition - A survey, *Pattern Recognition, 29,* 641-662.

Vafaie, H., & De Jong, K. (1993). Robust feature selection algorithms. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*, pp. 356-366.

Van Laarhoven, P.J.M., & Aarts, E.H.L. (1987). *Simulated Annealing: Theory and Applications.* D. Reidel: Dordrecht.

Wang, Y.K., & Fan, K.C. (1996). Applying genetic algorithms on pattern recognition: An analysis and survey. In *Proceedings of the Thirteen International Conference on Pattern Recognition, 2,* 740-744.

Weideman, W. E., Manry, E. M., & Yau, H. C. (1989). A comparison of a nearest neighbor classifier and a neural network for numeric handprint character recognition. *Proceedings of the International Joined Conference on Neural Networks,* Washington, DC, *I,* 117-120.

Weiss, S., & Kulikowski, C. (1991). *Computer Systems that Learn.* San Mateo, CA: Morgan Kaufmann.

Wettschereck, D., Aha, D. W., & Mohri, T. (1997). A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review, 11,* 273-314.

Wilson, D. Randall, & Martinez, R. Tony (1996). Instance-based learning with genetically derived attribute weights, *Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks* (AIE'96), pp. 11-14.

Zhang, H., & Sun, G. (2002). Feature selection using tabu search method, *Pattern Recognition, 35(3),* 701-711.