TASK QUARTERLY 14 No 4, 377–396

VOICE COMMAND RECOGNITION USING HYBRID GENETIC ALGORITHM

MARTA WRONISZEWSKA AND JACEK DZIEDZIC

Faculty of Technical Physics and Applied Mathematics,
Gdansk University of Technology,
Narutowicza 11/12, 80-233 Gdansk, Poland
martapad@gmail.com, jaca@kdm.task.gda.pl

(Received 10 May 2010; revised manuscript received 16 December 2010)

Abstract: Speech recognition is a process of converting the acoustic signal into a set of words, whereas voice command recognition consists in the correct identification of voice commands, usually single words. Voice command recognition systems are widely used in the military, control systems, electronic devices, such as cellular phones, or by people with disabilities (*e.g.*, for controlling a wheelchair or operating a computer system).

This paper describes the construction of a model for a voice command recognition system based on the combination of genetic algorithms (GAs) and K-nearest neighbour classifier (KNN).

The model consists of two parts. The first one concerns the creation of feature patterns from spoken words. This is done by means of the discrete Fourier transform and frequency analysis. The second part constitutes the essence of the model, namely the design of the supervised learning and classification system. The technique used for the classification task is based on the simplest classifier – K-nearest neighbour algorithm. GAs, which have been demonstrated as a good optimization and machine learning technique, are applied to the feature extraction process for the pattern vectors. The purpose and main interest of this work is to adapt such a hybrid approach to the task of voice command recognition, develop an implementation and to assess its performance.

The complete model of the system was implemented in the C++ language, the implementation was subsequently used to determine the relevant parameters of the method and to improve the approach in order to obtain the desired accuracy. Different variants of GAs were surveyed in this project and the influence of particular operators was verified in terms of the classification success rate.

The main finding from the performed numerical experiments indicates the necessity of using genetic algorithms for the learning process. In consequence, a highly accurate recognition system was obtained, providing 94.2% correctly classified patterns. The hybrid GA/KNN approach constituted a significant improvement over the simple KNN classifier. Moreover, the training time required for the GA to learn the given set of words was found to be on a level that is acceptable for the efficient functioning of the voice command recognition system.

Keywords: voice command recognition, genetic algorithms, K-nearest neighbour, hybrid approach

1. Introduction

The aim of this work was to construct a model for a voice command recognition system based on the combination of genetic algorithms and the K-nearest neighbour classifier.

Speech recognition is a very complex task and in the last decades substantial effort has been undertaken in order to find efficient recognition methods [1]. Nowadays, real-time voice recognition systems are widely used in many commercial electronic devices. Typically in such cases, speech recognition is user-dependent and concerns a small vocabulary consisting of only a few commands that are used, for instance, for navigation. One of potentially important applications of voice command recognition is in systems for handicapped persons. These aspects make the development of recognition techniques noteworthy and encourage further research in this field.

However, significant variability is associated with the signal, even if the words are spoken by the same person. First, the acoustic realizations of phonemes are highly dependent on the context in which they appear. Second, the same word is never spoken by the same person in exactly the same way, sometimes it can have a different length or emphasis on other phonemes. Finally, changes in the environment in which the words are spoken can strongly affect the recognition process. The above problems have to be overcome during the construction of a successful recognition system.

Genetic algorithms (GAs) have been demonstrated as a good optimization and machine learning technique [2]. Due to their unusual ability of searching in a large space of possible solutions they have been used previously in combination with the K-nearest neighbour algorithm (KNN), forming a hybrid approach (GA/KNN). GA/KNN has been proven to be an efficient classifier [3, 4]. This approach has been used successfully in many practical applications such as biological measurements [3], the classification of soil samples [4], cancer diagnosis [5] or fraud detection [6]. Such a wide spectrum of possible uses inspired the authors to attempt to adapt the GA/KNN approach to voice command recognition.

The construction of the voice command recognition model described in this paper consists of two parts. The first one concerns the creation of feature patterns from spoken words. This is done by the discrete Fourier transform and subsequent frequency analysis. The second part constitutes the essence of the model, namely the design of the supervised learning and classification system. Patterns are divided into a training set, on which the algorithm learns; and a testing set, which serves to estimate the accuracy of the system. The main idea of the hybrid approach is an application of the GAs to the process of feature extraction in the pattern vectors, in order to improve the rate of correct classifications subsequently made by the KNN.

The model was implemented in C++ and a series of numerical experiments was performed in order to verify the influence of the parameters of the method

on the accuracy of the system, leading to a formulation of a satisfactory structure for the model.

This paper is structured as follows. Section 2 introduces basic concepts of genetic algorithms. In Section 3, the hybrid approach combining GAs and KNN is described. Section 4 is devoted to the construction of a voice command recognition system, including data pre-processing and supervised learning and classification. In Section 5, the details of the numerical experiments are described, followed by results and comments. Section 6 presents conclusions.

2. Genetic algorithms

Genetic algorithms constitute powerful tool for solving computational issues requiring searching [2, 7]. These kinds of problems are very common in optimization, machine learning or modelling and in predicting real-world phenomena. Genetic algorithms were first described by John Holland in the 1960s [8] and were subsequently studied by Holland and co-workers at the University of Michigan in the 1960s and 1970s [8].

The general idea is to use evolution as an inspiration for solving problems. Evolution can be seen as a method for searching among an enormous number of possibilities. Genetic algorithms use an evolutionary mechanism, the basic concept of which can be described as follows. A population consists of a number of individuals. Each of individuals has an unique code called a chromosome and presents a different level of adaptation to environment. In each generation, evolution promotes better fit individuals by allowing them to reproduce and transfer their genetic code to the next generation. Therefore, by natural selection highly fit individuals have a greater chance compared to poorly-fit individuals. During reproduction, the crossover operator exchanges subparts of two chromosomes, while mutation serves to change the value at randomly chosen locations in the genetic code. Following reproduction, a new generation is created, replacing the "old" population. Due to the higher probability of reproduction for well-fit individuals, evolution eliminates poorly-fit individuals over generations and tends towards saving the optimal (highest fit) individuals.

In GA terms, an individual, or rather its chromosome, represents a potential solution to the problem. The set of all possible individuals can be seen as a *search space*. This refers to the collection of candidate solutions to a problem. GAs operate on a *population*, which represents a subset of all possible solutions. Every individual is represented by a chromosome which can be either a vector (*e.g.*, a bit string) or a matrix, or even a tree structure (*cf.* genetic programming, [9]). The act of representing a candidate solution as a chromosome is called *encoding*. During the simulation, the whole population is subject to simulated evolution. First, the *fitness* of every individual in the population is evaluated. Fitness is calculated according to a criterion (fitness) function which depends on the problem under study. The value of the criterion function corresponds to the perceived success of the individual in solving the problem. Then, *selection* chooses solutions

with higher values of the fitness function giving them a greater probability of recombination. After a pool of individuals for reproduction is selected, *crossover* takes place. This operator randomly exchanges parts of chromosomes between the two parents. The next operator – *mutation* randomly flips some of the bits in the chromosome. Mutation and crossover occur with a certain probability. Finally, the old population is replaced by the newly obtained one. The above procedure describes the basic concept of a genetic algorithm. For a more thorough description the reader is referred to [7, 9].

The following procedure describes all steps of the simple GA:

Procedure:

1. Initialize the population. Randomly generate the set of candidate solutions (*i.e.* N chromosomes)
   Repeat steps 2–6 G times (G is the number of desired generations):
2. Calculate the fitness $f(x)$ of each chromosome in the population.
3. According to the fitness, select a pool of parents (one chromosome can be selected more than once to become a parent).
4. For each pair of parents, with a probability p_c (crossover probability), perform crossover. With a uniform probability, choose a point for the exchange of chromosome parts. If no crossover takes place, offspring are exact copies of their parents.
5. With a probability p_m (mutation probability) mutate the offspring in each locus in their chromosome.
6. Replace the current population with the new one.
7. Go to step 2.

3. GA/KNN for pattern classification

As mentioned earlier, the essential part of this work consists in designing a supervised learning and classification system. In this paper we propose a hybrid approach combining genetic algorithms and the nearest neighbour method for voice command recognition.

3.1. K-nearest neighbour algorithm

K-nearest neighbour algorithm (KNN) is a simple, yet quite useful tool for dealing with the classification problem [10]. The main idea can be expressed as follows: given a dataset of classified examples, an unclassified pattern should belong to the same class as its nearest neighbour (in case of $K=1$) or the same class as the majority of its nearest neighbours (in case of $K>1$) in the dataset. The distance between neighbours is usually measured in the Euclidean metric. Figure 1 illustrates the mode of action of the K-nearest neighbour algorithm in 2-dimensional space. KNN is a quick and effective method for classification and for many particular problems it is at least as good as more sophisticated classification algorithms [11]. It tends to fail, however, in cases in which the proportion of attributes that are significant in the classification process is small with respect to

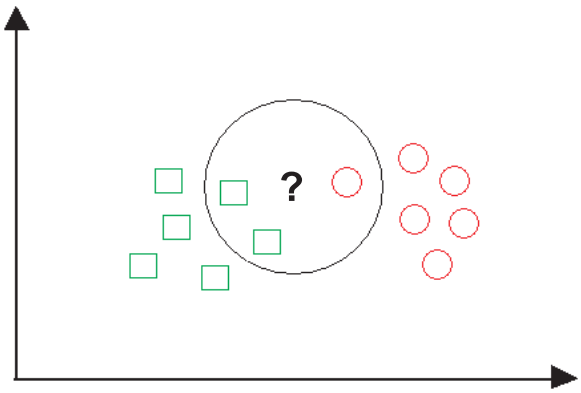

Figure 1. The K-nearest neighbours algorithm for $K=3$. The pattern denoted by the question mark is subject to classification. The three nearest neighbours to the unclassified pattern are inside the circle. One of them belongs to the red circle class and two of them belong to the green square class, therefore, according to the majority rule, the unknown pattern should be classified as a green square

the number of attributes that are irrelevant or misleading [12]. To address this problem feature extraction has been proposed [13].

3.2. Feature extraction

As mentioned earlier, the KNN algorithm is a very effective procedure but suffers in the presence of noisy or irrelevant features, because it treats all the attributes equally. Unfortunately, very often (particularly when the feature vector is highly dimensional) there is a large variation between the importance of attributes and this should be taken into account in the classification process.

This is accomplished by feature extraction techniques, where the closeness to the more important attributes becomes more critical than the closeness with respect to the less important ones. Feature extraction is accomplished by assigning weights $w=(w_1, w_2, \ldots, w_n)$ to every attribute in the pattern vector. A large weight assigned to a feature means that this feature is very important in classification and a low weight means that this feature is noisy or less relevant and should not be taken into account as much in the class determination procedure. The effect of such scaling is illustrated by the example in Figure 2.

The scaling should be applied to every pattern in the data set. In this case, the KNN algorithm is transformed into a so-called *weighted K-nearest neighbour algorithm* and the closeness between the two patterns x and y is measured (in the Euclidean metric) by:

$$dist_{x,y}=\sqrt{\sum_{i=1}^{n}(w_i(x_i-y_i))^2} \tag{1}$$

where w_i is the weight of the i^{th} attribute. Weights can be real or integer numbers from a defined interval.

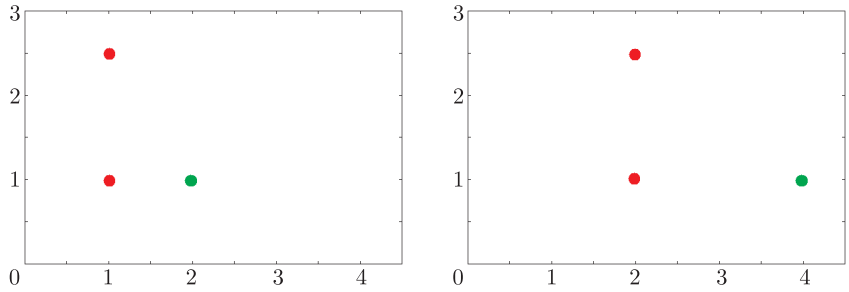

Figure 2. Consider two classes: red and green. Suppose, that the pattern $(1, 1)$ is subject to classification. In the left panel, according to the KNN rule with $K = 1$, this pattern would be incorrectly assigned to the green class. Scaling all pattern vectors by a weight vector $w = (2, 1)$ leads to correct classification as shown in the right panel

Feature extraction improves the performance of the KNN. Nevertheless, finding a suitable weight vector is a very complex task, impossible to solve directly, because an exhaustive search through all the combinations is computationally impractical. Because of the implicit parallelism inherent the GAs, they cope well with highly dimensional parameter spaces, thus they are a good candidate for improving the performance of the K-nearest neighbour algorithm, which is then applied to feature extraction.

3.3. Structure of GA/KNN

In this approach genetic algorithms are used to find a sufficiently good weight vector. Here, a population of GA defines a set of vectors $w = (w_1, w_2, \ldots, w_n)$, where n is the dimension of data patterns. Then, each vector w is multiplied by every sample pattern x from the dataset. Such transformed patterns are then classified by the KNN rule. The result of the classification is fed back as part of the fitness function, to guide the genetic search towards the best transformation. Therefore, the GA returns the nearly optimal weight vector, which leads to satisfactory classification of all samples in the data set – the *training set*. Figure 3 schematically illustrates the mode of action of the GA/KNN approach. When the process of supervised learning is over, it is possible to check if the constructed algorithm is able to correctly classify new, unknown patterns, taken from *testing set*.

4. Model for a voice command recognition system

The aim of our work is to design a system capable of recognizing voice commands. The basic steps that were undertaken to create such system can be expressed as follows. Several words spoken by the same man have been recorded and a dataset consisting of those words has been built. The task for the system was to recognize new utterances of the words spoken by the same person as one of the words recorded earlier. As the utterances of the same word, even if said by the same person, do not produce the exact same waveform, it was very important to make learning possible for the system by using a larger dataset. Therefore, each

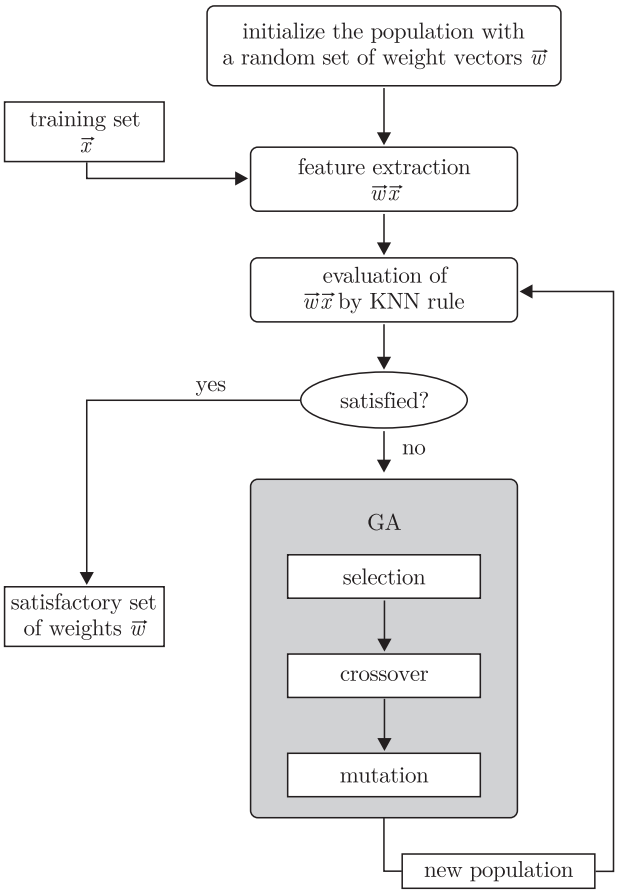

Figure 3. Mode of action of the hybrid GA/KNN approach

word spoken by the same person was recorded several times. The next step was to design a proper decision system capable of correctly classifying the new voice commands, based on the training data. Finally, the system was tested on new, unknown utterances and its accuracy was assessed.

4.1. Data pre-processing

All words were recorded in a noise-free environment using a standard microphone. They were stored in the waveform audio format (WAV) with a sampling rate equal of 44.1 kHz, which changes a continuous sound signal $x(t)$ into a discrete series $X(k)$, $k = 1, 2, \ldots, N-1$.

After reading the sounds from the WAV files into arrays, a set of words, each descried by a discrete series, is obtained:

$$V = \{X_1, X_2, \ldots, X_{WN}\} \tag{2}$$

where WN is the number of words.

The first step is to normalize the representations to the highest volume:

$$X_i'(n) = \frac{X_i(n)}{\max\limits_{0 \le n \le N-1} \{X_i(n)\}}, \quad n = 0, 1, 2, \ldots \quad i = 1, \ldots, WN \tag{3}$$

The second step consists in finding where each words begins. The starting point of a word is determined by finding a sample with an amplitude greater or equal to 3% of the maximum volume:

$$m_i = \min_{0 \le n \le N-1} \{n | X_i(n) \ge 0.03\} \tag{4}$$

Then all samples $X_i(n)$ where $n < m_i$ are discarded. The end of the word is also discarded but in such a way as to leave all words with the same length. Having checked all the recordings, it turned out that none of the the words had a length greater than one second. For this reason, all the recordings were truncated to one second. Assuming a sampling rate of 44.1 kHz, all the words of one second in length consist of 44 100 samples:

$$X_i''(n) = \{X_i(m), \ldots, X_i(N-1)\} \tag{5}$$

where $N = m + 44\,100$, $i = 1, \ldots, WN$.

Therefore, each of the words is represented as a vector of 44 100 numbers. The Fast Fourier Transform algorithm (FFT) was subsequently employed in order to transform every word from the time domain into the frequency domain. The size of the FFT was taken as equal to $N_{\mathrm{FFT}} = 2048$.

The frequency vector f is given by:

$$f(k) = \frac{k f_s}{N_{\mathrm{FFT}}}, \quad k = 0, 1, 2, \ldots, N_{\mathrm{FFT}} - 1 \tag{6}$$

where f_s is the sampling rate and N_{FFT} is the FFT step. Therefore, every time frequency vector will be in the range $[0, 44\,100]$ Hz.

To avoid aliasing, the frequency vector is truncated at the Nyquist frequency [14]:

$$f(k) = \frac{k f_s}{N_{\mathrm{FFT}}}, \quad k = 0, 1, 2, \ldots, N_{\mathrm{FFT}}/2 - 1 \tag{7}$$

reducing the frequency spectrum to the range $[0, 22\,050]$ Hz.

The output of the FFT, $\widehat{X}(k)$, is a vector of complex values. By transforming these values, a power spectrum of the sound in each frequency is obtained. The following formula yields the power vector in decibels:

$$\bar{X}(k) = 20 \log \left(\sqrt{\mathrm{Re}^2\left(\widehat{X}(k)\right) + \mathrm{Im}^2\left(\widehat{X}(k)\right)} \right) \ [\mathrm{dB}] \tag{8}$$

where $\widehat{X}(k)$ is the FFT output vector.

Averaging over the whole word is not likely to give sufficient information for the recognition process, instead we propose the following approach.

First, all words were divided into 10 equally long sections:

$$X_i^1 = \{X(0), \ldots, X(4409)\}$$
$$X_i^2 = \{X(4410), \ldots, X(8819)\}$$
$$\vdots$$
$$X_i^{10} = \{X(39\,690), \ldots, X(44\,099)\}$$

and the FFT was performed on each section of the word, to obtain $\widehat{X}_i^j(k)$, where $i = 1, 2, \ldots, WN$, $j = 1, \ldots, 10$, $k = 0, 1, \ldots, N_{\mathrm{FFT}} - 1$. The FFT output is then transformed according to the formula (8) to obtain the power spectra $\bar{X}_i^j(k)$.

Since the power vector of length $N_{\mathrm{FFT}}/2$ for every $1/10^{\mathrm{th}}$ of the word is still too long for an efficient recognition and not all frequencies provide important information about the sound, we have decided to calculate the average power over predefined frequency intervals. This was achieved by dividing the frequency vector into several ranges and computing the average value of the power of the sound over all frequencies in this frequency interval. Taking into account the fact that low frequencies carry more information [15], the vectors were divided into roughly geometrically growing intervals, finishing at a frequency above which only noise was seen to occur. After careful experimentation we have arrived at the following points separating the ranges (in kHz):

$$r_1 = 0.1,$$
$$r_2 = 0.3,$$
$$r_3 = 0.7,$$
$$r_4 = 1.5,$$
$$r_5 = 3.1,$$
$$r_6 = 6.3,$$
$$r_7 = 12.7.$$

Therefore, for each frequency range, the average value of power of the sound in this range is obtained as:

$$\left\langle \bar{X}_i^j \right\rangle_q = \frac{\sum_{k=fr_q}^{fr_{q+1}-1} \bar{X}_i^j(k)}{fr_{q+1} - 1 - fr_q}, \quad q = 1, \ldots, 6 \tag{9}$$

where

$$fr_q = \min_{1 \le k \le N} \{k | f_k \ge r_q\} \tag{10}$$

The next step is to normalize these values to a range of our choosing 0–10:

$$x_{\max} = \max_{1 \le i \le WN} \max_{1 \le j \le 10} \max_{1 \le q \le 6} \left\{ \left\langle \bar{X}_i^j \right\rangle_q \right\} \tag{11}$$

$$\left\langle \bar{X}_i^j \right\rangle_q = \frac{10}{x_{\max}} \left\langle \bar{X}_i^j \right\rangle_q \tag{12}$$

In this way 6 values in the range 0–10 for each $1/10^{\mathrm{th}}$ of the word are obtained, allowing us to express each word as a $6 \times 10 = 60$-dimensional vector

$x_i = \left(\langle \bar{X}_i^1 \rangle_1, \ldots, \langle \bar{X}_i^1 \rangle_6, \quad , \ldots \ldots, \quad \langle \bar{X}_i^{10} \rangle_1, dots, \langle \bar{X}_i^{10} \rangle_6\right)$, which is a reasonable size for the recognition techniques.

4.2. Hybrid GA for supervised learning and classification

The test data consisted of 7 different words from the Polish vocabulary:

$V = \{$napisz, pulpit, start, uruchom, wyślij, zadzwoń, zatrzymaj$\}$[1].

10 utterances of each word were recorded, yielding 70 samples in the training set. The hybrid genetic algorithm operates only on this set. Its task is to learn, based on the training set, which would ultimately lead to the capability of correct classification of unknown samples from the testing set, which consists of 175 utterances – 25 samples for each word. To make the estimation of the accuracy of the system more reliable, simulations on the model are performed taking a different training and test set each time.

For supervised learning and the classification of data patterns characterising words, the hybrid GA/KNN algorithm introduced in Section 3.3 was used. The task of the genetic algorithm consists in finding near-optimum weights for the scaling of all the feature vectors assigned to each word. The components of the weight vector, w, were chosen as integer numbers from the interval $[0, 127]$.

The GA was set up as follows:

- Initialization
  An initial set of the chromosomes was chosen randomly. The size of the population was set to 60, which balanced the computational effort and the obtained accuracy.
- Encoding
  Binary encoding was used and each weight component of the vector w is encoded on 7 bits, yielding chromosomes of 490 bits in length.
- Fitness function and selection
  We have experimented with different types of the fitness function. First, intuitively the fitness was taken to be equal to the number of correctly classified patterns, *i.e.* $fitness = N_C$. This led to acceptable results, however, the evolution in this case proved to be very slow, since the degree of similarity between individuals was high, leading to a flattened fitness landscape. An obvious development was to promote individuals producing K identical neighbours:

$$fitness = N_C + \sum_{i=1}^{N_T} b_i \tag{13}$$

  where N_T is the number of patterns and

$$b_i = \begin{cases} \Delta, & \text{if the number of the identical neighbours} \\ & \text{for } i^{\text{th}} \text{ pattern} = K, \\ 0, & \text{otherwise.} \end{cases} \tag{14}$$

  The parameter Δ was chosen to be 4.

1. Meaning {write, desktop, start, activate, send, call, stop}, respectively.

In order to choose individuals for reproduction, proportionate selection was used initially but due to the very small diversity in the population, it became obvious fitness scaling would become necessary [2]. Linear fitness scaling was applied, according to the procedure:

Procedure:

1. Find the maximum and minimum fitness value in the population:

$$f_{\max} = \max f(x_i) \quad i \in 1, 2, \ldots, N \tag{15}$$

$$f_{\min} = \min f(x_i) \quad i \in 1, 2, \ldots, N \tag{16}$$

2. Determine the average fitness in the population:

$$f_{\text{avg}} = \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{17}$$

3. Scale the fitness for each individual as follows:

$$f'(x_i) = a f(x_i) + b \tag{18}$$

where a and b are scaling parameters determined by the condition (20).

- check scaling condition (non-negative test):

$$f_{\min} > \frac{c f_{\text{avg}} - f_{\max}}{c-1} \tag{19}$$

where c is the number of expected copies desired for the best population member

- if above condition is satisfied then:

$$a = \frac{f_{\text{avg}}(c-1)}{f_{\max} - f_{\text{avg}}}, \quad b = -f_{\text{avg}}(a+1) \tag{20}$$

- else:

$$a = \frac{f_{\text{avg}}}{f_{\text{avg}} - f_{\max}}, \quad b = \frac{-f_{\min} f_{\text{avg}}}{f_{\max} - f_{\text{avg}}} \tag{21}$$

The application of fitness scaling did indeed improve the search but after further experiments ranking selection [2] turned out to be a better choice. It was implemented according to the following description. Each individual was ranked in increasing order of fitness from 1 to N, where N is the size of the population. Expected number of children of individual is given by $ExpVal(i) = Min + (Max - Min)\frac{rank(i)-1}{N-1}$, where Max is the expected value of the highest fit individual with rank N (this value is chosen by the user) and Min is the expected value of the lowest fit individual with rank 1. $rank(i)$ is the ranking number assigned to the individual i. Since we want to keep the population size constant from generation to generation and assume Max to be greater then 0, it is required that $1 \leq Max \leq 2$ and $Min = 2 - Max$. Baker recommended $Max = 1.1$ (derived experimentally) [16].

- Crossover and mutation
  Single-point crossover was used. After experiments with varying crossover rates, p_c was chosen as 0.9. The difference in the rate of evolution for varying p_c was not pronounced.
  The choice of the probability of mutation (p_m) turned out to significantly affect the algorithm. $p_m = 0.002$ was chosen.
  Elitism [7] was used to improve convergence.

5. Experiments and results

5.1. Basic model

The first experiment is the realization of the basic model of a classification system that arises directly from the techniques described in the previous sections. It plays a meaningful role as a reference and a source of basic conclusions.

The fitness function in the basic model was given by the formula (13)–(14). Proportionate selection with linear fitness scaling and elitism were used for the GA. The GA in this model tends to maximize the number of correct classifications, which is equal to 70 (the size of the training set) and to maximize the number of identical and correct neighbours which is equal to 3 for every sample. Therefore, the maximum possible value of the fitness function is $70 \times 5 = 350$ (in the case that all words are correctly classified and having all nearest neighbours from the right class).

The results for the basic model are compared with the simple KNN classification without evolutionary guided learning (*i.e.*, without feature selection). 10 experiments were performed and the average score was calculated. The experiments in this subsection, as well as all the others described here were performed on a desktop PC to reflect the application point of view. Table 1 presents the results for this model.

Table 1. Results for the basic model

Set (number of patterns)	Correctly classified patterns by KNN	Correctly classified patterns by GA/KNN
Training (70)	84.6%	95.7%
Test (175)	87.0%	93.1%
Training time (seconds)		26

It is immediately seen that GA/KNN outperforms KNN by a significant margin. The reason for this is that ordinary KNN is sensitive to the noise in the data and does not rate the relative importance of individual features for discrimination, in contrast to the hybrid approach.

5.1.1. Analysis of GA

Here we turn our attention to the behaviour of the GA for the test set. The experiments described below were performed to examine the influence of the parameters on the algorithm's efficiency.

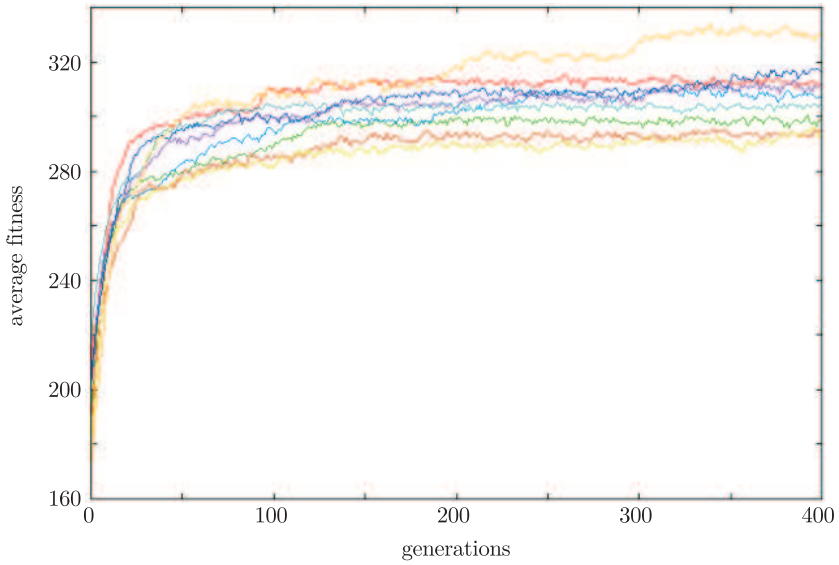

Figure 4. Population fitness for 10 runs with the basic model

The plot in Figure 4 shows how the average population fitness changes with the number of generations (for 10 simulations).

Initially, the rate of change is high, then the fitness stabilises with small fluctuations. This is a manifestation of the following facts: at the beginning, the evolution rate is very high, crossover is the dominant operator and the algorithm quickly converges towards the optimum solution. As the diversity decreases, mutation starts to play a more important role – the evolution slows down and only small fluctuations are observed.

An important finding is that elitism and linear fitness scaling turned out to be very important improvement to the basic model (*cf.* Figure 5).

As can be inferred from the plots, in the simple GA model the population fitness tends to stabilize after reaching a certain sub-optimal level, whereas with elitism and linear fitness scaling, the population keeps the best individuals unchanged (elitism), which guides the rest towards better solutions. Moreover, fitness scaling promotes diversity in the population, allowing even poorer individuals to reproduce and keeps selection pressure at sufficiently high level. This leads to a marked improvement in the rate of change of the fitness and the final fitness.

Another crucial aspect is the influence of mutation and crossover rates. Figure 6 shows the plots of the average fitness for different crossover probabilities (p_c) and Figure 7 for the different mutation probabilities (p_m).

From the above plots it can be seen how important the crossover operator for the GA is. With $p_c = 0$ evolution at the beginning is guided only by random changes in the chromosomes and very quickly stagnates at a low level. A crossover probability of 0.9 seems to represent a sufficient balance. It is high, yet leaves

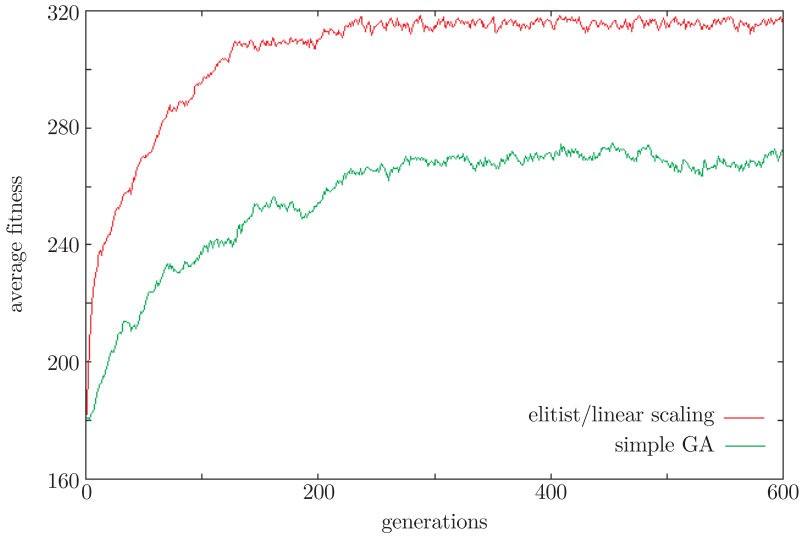

Figure 5. Comparison of simple GA (green) and the elitist linear fitness scaling model (red)

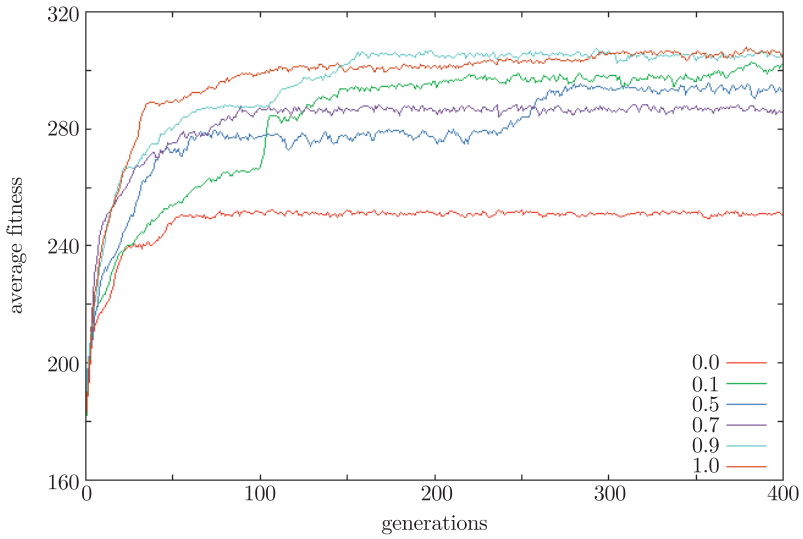

Figure 6. Fitness for different crossover rates

some chance for the best individuals, not to destroy their genetic code during reproduction.

Despite the fact that mutation is a secondary operator with less importance then crossover [2], its influence is still significant. With $p_m = 1$ the optimization process becomes a random search. On the other hand, an excessively small mutation probability is not desirable either, since it leads to stagnation in one of thr local optima. The value of $p_m = 0.002$ was found to be a good choice, preventing stagnation but at the same time not causing a problem with convergence.

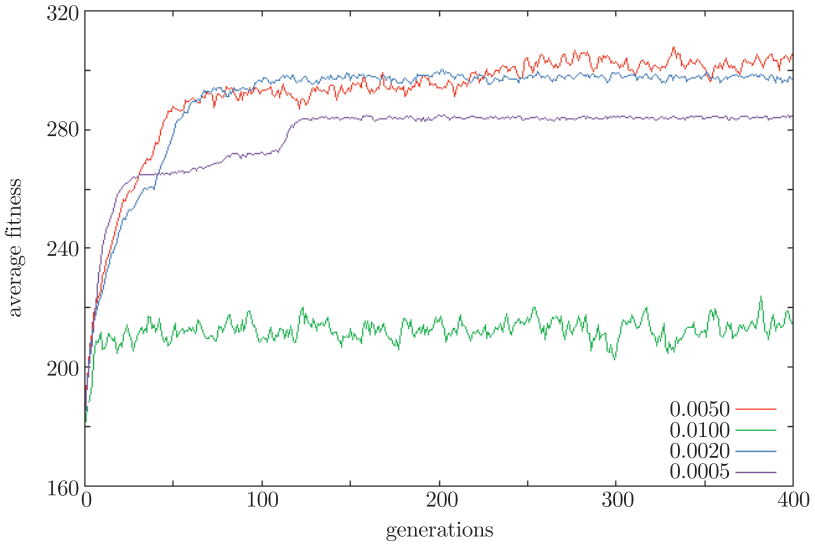

Figure 7. Fitness for different mutation rates

5.2. Error correction model

In this model, a slightly different approach was tested. Instead of taking the number of correctly classified patterns, misclassifications are counted $(N_T - N_C)$ and their ratio to the total number of patterns (N_T) forms a penalty term, which becomes a part of the fitness function. The second component of the fitness function affects the performance of the KNN algorithm by counting the number of neighbours which are from the wrong class and do not take part in the voting process $(S_{\min})$. Therefore, the fitness function is subject to minimization, as the decrease of misclassifications and number of different neighbours is desired. The formula for the fitness function becomes:

$$\mathit{fitness} = \gamma \frac{N_T - N_C}{N_T} + \delta \frac{S_{\min}/K}{N_T} \tag{22}$$

The parameters γ and δ tune the influence of each term on the fitness function. An increase in γ emphasises the number of correctly classified patterns, whereas larger δ forces the algorithm to look for such classifications that are as close to being unanimous as possible. In this experiment the rank selection for the GA was also introduced.

Table 2 presents results for this model for different values of γ and δ. The measure of the efficiency is the error rate in the classification process, *i.e.*, the ratio of missclassifications to the total number of patterns.

The influence of the parameters γ, δ is easily noticeable. With lower δ and higher γ, the algorithm focuses on the maximization of correctly classified patterns, leaving the number of different neighbours on quite a high level. Setting $\gamma = 5\delta$, proves to be the best choice for γ and δ parameters, leading to best results. The final result in this case is an error rate of 5.8%, corresponding to a final accuracy of 94.2%, which represents an improvement over the simple model.

Table 2. Results for the error correction model

Parameters	Error rate (training)	Different neighbours	Error rate (testing)
$\gamma = 1.0,\ \delta = 0.5$	1.7%	5.5%	7.2%
$\gamma = 1.0,\ \delta = 0.2$	1.1%	4.4%	5.8%
$\gamma = 1.0,\ \delta = 1.0$	2.4%	5.3%	8.2%
$\gamma = 2.0,\ \delta = 0.2$	0.8%	6.4%	7.5%
Training time (seconds)			26

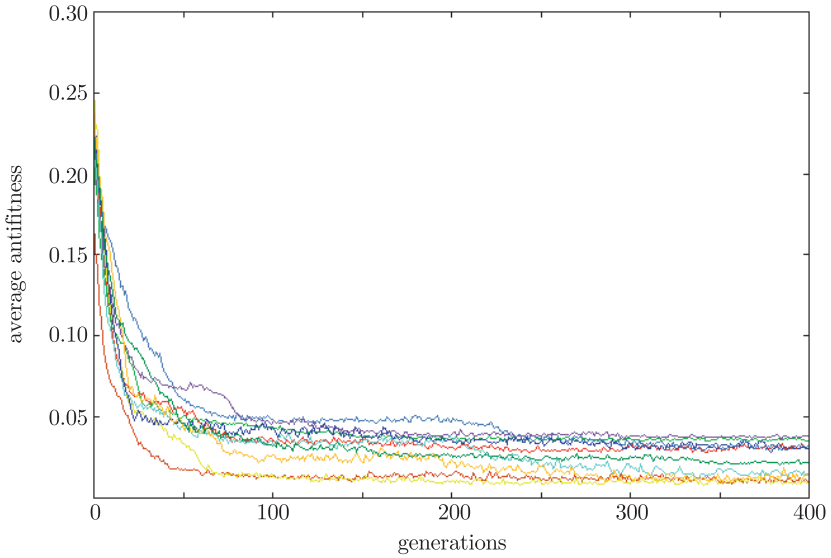

Figure 8. Antifitness for the error correction model with $\gamma = 1.0$ and $\delta = 0.2$ for 10 experiments/initial populations

Figure 8 shows the plots of the average antifitness for 10 simulations.

5.3. Varying K parameter model

The aim of this model is to examine the impact of the different number of neighbours in the KNN algorithm. Tests were performed with different values of K. Table 3 presents recognition results for $K = 1,5$ and 7.

All the parameters for the GA in this model are the same as in the basic model. The fitness function takes into account the number of correctly classified patterns and in case of $K = 5$ and $K = 7$ it additionally rewards the individuals which produce the highest number of correctly classified neighbours.

Curiously, $K = 1$ yields a better classification score for the pure KNN algorithm, than $K = 3$. However, this does not imply better performance of the GA – we can see that learning and feature extraction leads to only a slight improvement over KNN eventually producing a worse final result than the basic model. Obviously, in case of $K = 1$ the fitness function takes into account only the number of correctly classified patterns, since there is no room for optimization

Table 3. Results for the varying K model

Number of neighbours	Correctly classified patterns by KNN	Correctly classified patterns by GA/KNN	Training time
$K = 1$	89.7%	90.2%	24
$K = 5$	86.2%	88.5%	27
$K = 7$	76.5%	82.2%	29

of the number of neighbours. The population has a small level of diversity and with one neighbour, the GA learns perfectly on the training set but shows low capability for generalization when it comes to the testing set.

From the other results, it can be inferred that an increase in the number of neighbours causes a decrease in the classification score. For $K = 5$, the number of correctly classified patterns by the simple KNN algorithm is lower than for $K = 1$ or $K = 3$. In case of $K = 7$, the percentage of correct classifications is even lower. However, in this last case the GA plays an essential role, since it improves on the plain KNN significantly. The reason for this can be expressed by contrasting this with the situation for $K = 1$. Here, the GA tends to increase the number of correct neighbours and since there are more neighbours, it gives a better degree of generalization.

This experiment showed that for the system under study the number of neighbours equal to 3 is the perfect choice and consequently this value was used in subsequent experiments.

5.4. Changing the size of the training set

This experiment investigated the influence of the size of the training set on the results of the classification. The program was executed for sizes of the training set ranging from 4 to 15. Figure 9 presents the results.

It is seen that an increase in the training set size lead to increase of the classification score. As expected, for the training set the correctness rate is acceptable from the beginning. Intuitively, fewer utterances require fewer conditions to be satisfied, in order to find a reasonable good weight vector correctly classifying training patterns. However, achieving perfect discrimination for a small training set hinders generalization – the algorithm learns only the features typical for a small subset of patterns, which are not necessarily representative for new samples – lower classification result for the testing set speaks well for this fact. What is significant, however, is that up to 10 utterances of each word, the percentage of correctly classified patterns grows very fast. A further increase of the training set size improves the score only marginally, yielding diminishing returns.

The conclusions from the above results cannot be formulated in isolation. They are strongly connected with the processing time required for the classification process, which differs with the number of words in the training set. Clearly, the time required for training grows with the number of patterns in the training set. This growth is significant, so one has to be very careful in increasing the

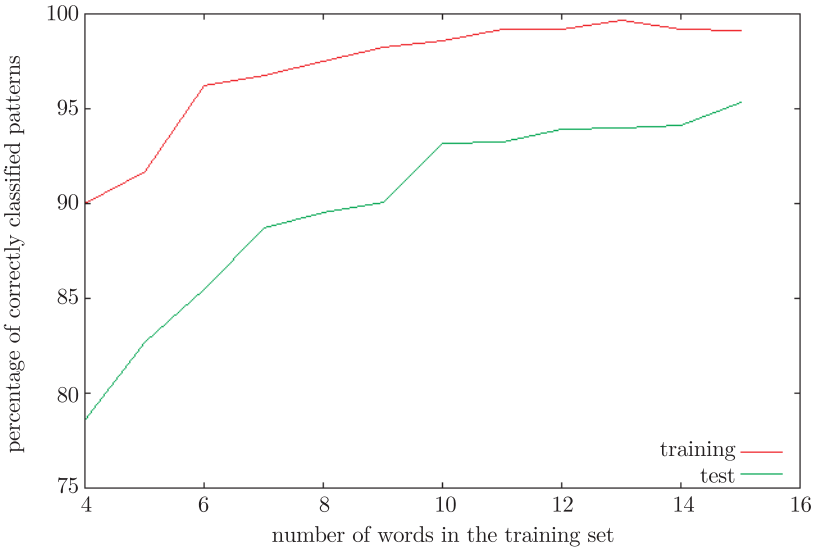

Figure 9. Percentage of correctly classified patterns as a function of the number of utterances of each word in the training set

training set size, keeping in mind the fact that this will lead to an increase in the learning time.

6. Conclusions

The main goal of this paper was to describe a voice recognition system based on the hybrid genetic algorithm and K-nearest neighbour classifier and to determine the performance of the constructed method. Having determined the influence of the simulation parameters on the classification score, a satisfactory construction of the model was proposed, providing the overall accuracy of 94.2% correctly classified patterns in 26 seconds.

6.1. Overall performance

The first experiment established the most important inference for this model – the necessity of using genetic algorithms for the learning process. The proposed GA/KNN approach was a significant improvement over the simple KNN classifier. Clearly, the basic KNN is sensitive to the noise in the data and does not rate the relative importance of individual features for discrimination. The introduction of a GA for feature extraction resulted in a significant increase in the accuracy of the system. The GA/KNN outperforms the KNN algorithm for voice command recognition, which is a very meaningful conclusion, as simple KNN is still widely used in the area of voice recognition. Moreover, the training time required for the GA to learn on the given set of words turned out to be on a reasonable level – 26 seconds, which is acceptable for an efficient functioning of a voice command recognition system.

A further result, provided by the last experiment, proved that a training set size of 10 utterances of each word is an equilibrated value. This amount reconciled two opposite issues that have to be taken into account – large enough size of the training set to enable generalization and as small as possible number of repetitions, in order to avoid the necessity of recording an excessive number of samples of one word and excessive computational overload. The influence of the K parameter has been assessed. We found that an increase in the number of neighbours causes a decrease in the classification score. However, with $K = 1$, the algorithm did not produce better results either, since it was not able to develop generalization capability in the learning process. As a consequence, the number of neighbours equal to 3 was found to be the best choice.

Finally, it is worth mentioning that the relatively high performance of the simple KNN algorithm is the result of quite discriminant data set, which is a consequence of the careful choice made in the construction of pattern vectors. The discrete Fourier transform and spectral analysis were found to be an efficient component of the recognition system.

6.2. GA's performance

Since a GA played an essential role in the voice command recognition system, it is important to highlight the most meaningful results of its performance.

First of all, the nearly optimum score for the training set achieved in the very short time confirmed the capability of the GA for searching in a large space of possible solutions. Series of experiments with changing conditions proved the GA to be a very adaptive method.

The results for the basic and error correction model provided further conclusions. Comparison of the average fitness for the different crossover probabilities determined the importance of the crossover operator. Using crossover is absolutely necessary for the correct functioning of the algorithm. Similarly, although with less impact, the mutation operator introduces randomness into the evolution process, preventing stagnation in local optima and resulting in a better algorithm score.

It is worth mentioning that the determination of a selection method turned out to be an important issue in the GA as well. Results for the proportionate selection and ranking selection methods showed better performance of ranking. This technique helped overcome the low diversity in the population, yielding a better final score.

Another finding is that for this model, like for many other applications of GAs, the simple GA was not sufficient. Introduction of elitism and linear fitness scaling led to significant improvements.

6.3. Further work

Further work can be devoted to the following issues.

One idea would be to emphasize the influence of the nearest neighbour in the class determination by the KNN. It is likely that the vote of the closest neighbour

should have more influence than that of any other k^{th} neighbour. Taking this fact into account requires the construction of a new fitness function for the GA.

Using an alternative metric for measuring the distance between patterns, *e.g.* the Minkowski metric [17] could also be considered.

More effort could be put into the verification of the influence of the parameters used in the construction of the pattern vectors. The first, obvious candidate is the choice of the frequency ranges over which average powers were calculated. However, the methods of dividing words and cutting the samples could be revised as well.

Finally, the impressive performance of GAs encourages one to go further and use them as a direct classification technique. Some results have already been achieved in this field [18], however it is worth a try to develop and adapt such genetic classifiers for voice command recognition systems.

References

[1] Rabiner L and Biing-Hwang J 1993 *Fundamentals of Speech Recognition*, Prentice Hall, Englewood Cliffs, NJ
[2] Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, New York
[3] Kelly J D and Davis L 1991 *Proc. 4th Int. Conf. Genetic Algorithms and their Applications (ICGA)*, San Mateo, CA, pp. 377–383
[4] Pei M, Goodman E D, Punch W F and Ding Y 1998 *Proc. IDEAL98*, Hong Kong, pp. 371–384
[5] Jain R and Mazumdar J 2003 *Australasian Physical and Engineering Sciences in Medicine* **26** (1) 6
[6] He H, Hawkins S, Graco W and Yao X 2000 *J. Adv. Comp. Intelligence and Intelligent Informatics* **4** (2) 130
[7] Mitchell M 1996 *An Introduction to Genetic Algorithms*, MIT Press, Cambridge
[8] Holland J H 1975 *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor
[9] Koza J 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press
[10] Cover and Hart P 1967 *IEEE Trans. on Information Theory* **13** (1) 21
[11] Shakhnarovich G, Darrell T and Indyk P 2006 *Nearest-Neighbor Methods in Learning and Vision*, MIT Press
[12] Beyer K, Goldstein J, Ramakrishnan R and Shaft U 1999 *Lecture Notes in Computer Science* **1540** 217
[13] Bot M C J 2001 *Proc. 4th European Conf. Genetic Programming*, Como, Italy, pp. 256–267
[14] Shannon C E 1949 *Proc. IRE* **37** (1), pp. 10–21
[15] Titze I R 1994 *Principles of Voice Production*, Prentice Hall
[16] Baker J E 1987 *Proc. 2nd Int. Conf. Genetic Algorithms and their Application*, Hillsdale, pp. 14–21
[17] Gavrilova M L 2002 *J. Supercomputing* **22** (1) 87
[18] Bandyopadhyay S, Murthy C A and Pal S K 1995 *Pattern Classification with Genetic Algorithms* **16** (8) 801