Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com
by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

CHAPTER 1.3.2

A NOTE ON GENETIC ALGORITHMS FOR LARGE-SCALE FEATURE SELECTION

W. SIEDLECKI and J. SKLANSKY
University of California, Irvine, Irvine, CA 92717, USA

We introduce the use of genetic algorithms (GA) for the selection of features in the design of automatic pattern classifiers. Our preliminary results suggest that GA is a powerful means of reducing the time for finding near-optimal subsets of features from large sets.

Keywords: Feature selection, genetic algorithms, classifier, search, multidimensional data.

1. Introduction

We introduce a form of genetic algorithm (GA) for selecting a small subset from an initially large set of coordinates of the feature space in the design of a pattern classifier. Reducing the dimensionality of the feature space not only decreases the cost and time of feature extraction in the operation of the classifier, but it also raises the credibility of the estimated performance of the classifier. Our initial experiments indicate that GA is a powerful tool for feature selection when the dimensionality of the initial feature set is large—specifically, greater than 20.

There are two versions of the problem of feature selection in the design of pattern classifiers, each version addressing a specific objective and leading to a distinct type of optimization. In one version the objective is to find a subset that yields the lowest error rate of a classifier. This version of the problem leads to unconstrained combinatorial optimization in which the error rate is the search criterion. In the second version of the problem we seek the smallest subset of features for which the error rate (or perhaps some other measure of performance) is below a given threshold. This version leads to a constrained combinatorial optimization task, in which the error rate serves as a constraint and the number of features is the primary search criterion.

In the search for subsets of features each subset can be coded as a d-element bit string or binary-valued vector (d is the initial number of features), $a = \{\alpha_1, \ldots, \alpha_d\}$,

Reprinted with permission from *Pattern Recognition Letters*, Vol. 10 (1989) 335–347. ©1989, Elsevier Science Publishers B.V. (North-Holland).

where α_i assumes value 0 if the i-th feature is excluded from the subset and 1 if it is present in the subset. We refer to a as a *feature selection vector* and α_i is a *feature selection variable*. Thus the search space for the feature selection problem is a space of d-element bit strings.

The search space can be conveniently represented in the form of a lattice (an example of the 4-dimensional lattice is shown in Fig. 1). In the lattice, which is an undirected graph, nodes correspond to points in the search space, and every pair of nodes is connected by a link if and only if the set of features represented by one of the linked nodes is an immediate subset (or superset) of the set represented by the other node. The top node in the lattice represents the full set of features and the bottom node corresponds to the empty set. The nodes in the lattice are grouped in levels: all nodes at the same level have the same number of features. By convention, the empty set node is placed at level 0, and the full set node occupies level d.

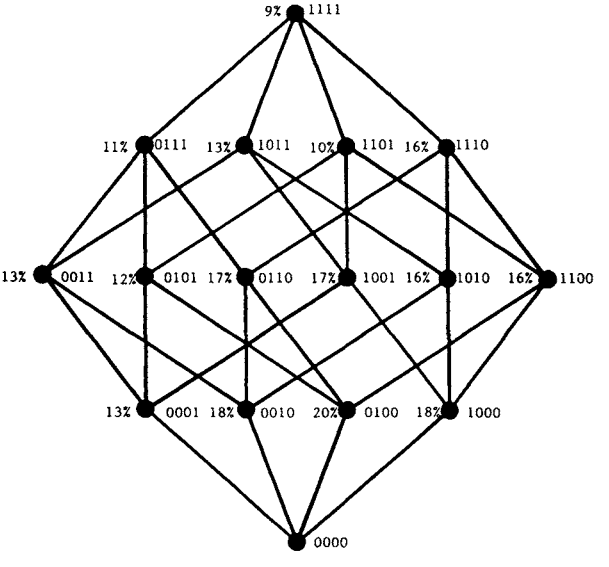

Fig. 1. A 4-dimensional feature selection lattice.

We say that a function $e(\cdot)$ has the *monotonicity property*, or is *monotonic*, if for every pair of linked nodes in the lattice, a and b, the following is true:

$$\text{if } l(a) < l(b) \text{ then } e(a) > e(b), \tag{1.1}$$

where $l(\cdot)$ is the level function,

$$l(a) = \sum_{i=1}^{n} \alpha_i. \tag{1.2}$$

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

The research on feature selection dates back to the early sixties (for an overview and biographical notes see [1] or later [2]). The most recent advances in this area are attributed to Narendra and Fukunaga [3], who introduced and tested the use of branch and bound, and Foroutan and Sklansky [4], who introduced the concept of *approximate monotonicity* and studied the use of branch and bound for selecting features for piecewise linear classifiers. In both cases the branch and bound method was used to minimize the number of features provided that a certain additional constraint was satisfied. The constraint induces backtracking in the branch and bound algorithm and, consequently, limits the size of the search space (which in our case is the *feature selection lattice*) to a *feasible region.*

Narendra and Fukunaga used probabilistic separability measures as the constraint while Foroutan and Sklansky used the error rate of the piecewise linear classifier. Although the branch and bound technique was reported to achieve about a 99% reduction over exhaustive search [4], it suffers from two major problems. The first problem becomes evident when we realize that in order for the branch and bound algorithm to find the optimal solution, it has to search the entire feasible region. If the constraint criterion does not have the monotonicity property, branch and bound is likely to prune a feasible part of the search space. In the worst case the feasible region may be even disconnected, i.e., it may consist of several parts separated by the infeasible area, and then the ordinary branch and bound procedure has no means to access and explore all disconnected feasible parts of the feature selection lattice. This problem was addressed by Foroutan and Sklansky in [4] and to some extent was overcome in the case of the error rates of piecewise linear classifiers, which were demonstrated to be only slightly nonmonotonic with respect to subset inclusion so that certain modifications to the backtracking rule in branch and bound would enable (with a certain likelihood) the algorithm to explore the entire feasible region.

The second major problem in the use of branch and bound for feature selection is that this algorithm performs an exhaustive search in the feasible region. The size of this region may have different values, depending on the value of the threshold selected by the user, but generally is uncontrollable and grows at the same rate as the size of the entire search space, i.e., as 2^d, where d is the initial number of features. While search by branch and bound in up to 20-dimensional feature space is still feasible it becomes rapidly impractical as this dimensionality is approached and exceeded.

Since the evaluation of the discriminatory power of a subset of features requires an estimate of the error rate of a classifier optimized for that subset, the complexity of the combinatorial search encountered in feature selection is amplified by the time required for designing the optimum classifier and estimating its error rate. We refer to such complex evaluation problems as *large-scale feature selection.* For such cases we need further search time reductions of two or more orders of magnitude. Toward this end we developed a feature selection procedure based on the concepts of genetic

algorithms. Preliminary tests on this procedure, reported here, suggest that this might be a way to achieve the desired reductions in search time.

2. The Concept of Genetic Algorithms (GA)

Although research in genetic algorithms has a twenty-year history it has been just recently that theoretical advances and several spectacular successes in practical application attracted more attention to this field and caused its rapid growth. (For details on genetic algorithms the reader is encouraged to refer to the classical book by Holland [5] or to the recent book by Goldberg [6].)

The ordinary genetic algorithm is an optimization procedure working in binary search spaces, i.e., the search spaces consisting of binary strings, but after some coding it can be also applied to continuous search spaces. Unlike classical hill-climbers it does not evaluate and improve a single solution but, instead, it analyzes and modifies a *population* (that is, a set) of solutions at the same time. The power of this intrinsic parallelism of genetic search is amplified by the mechanics of population modification, allowing the genetic algorithms to attack successfully even NP-hard problems (see [5] for more details).

In the genetic algorithm a solution, i.e., a point in the search space, is represented by a finite sequence of 0's and 1's, called a *chromosome*. (In the application of GA to feature selection, each chromosome represents a subset of features, the k-th bit denoting the presence or absence of the k-th feature.) The algorithm manipulates a finite set of chromosomes, the *population*, in a manner resembling the mechanism of natural evolution. In this mechanism, the chromosomes are allowed to *mate* or *crossover*, and to *mutate*. The mating of two chromosomes produces a pair of offspring chromosomes which are syntheses of their parents. A mutation of a chromosome produces a near identical copy with some components of the chromosome altered.

The optimization process is carried out in cycles called *generations*. During each generation a set of new chromosomes or bit strings $\{a_i\}$ is created through crossover, mutated and evaluated. Since the population size is finite, only a predefined number of the (best) chromosomes survives to the next cycle of reproduction. Despite its limited size, the population is capable of fast adaptation which results in rapid optimization of the criterion function (*score*).

A high-level algorithmic description of the basic method is given below. Though many variations of this basic method exist, our description captures its primary characteristics.

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

1. Construct an initial population set $\Pi = \{a_i\}_{i=1,\ldots,n}$
2. **For** $i \leftarrow 1$ **to** *Number_of_generations* **do**
   (a) Initialize mating set $M \leftarrow \emptyset$ and offspring O.
   (b) **For** $j \leftarrow 1$ **to** n **do**
       Add $f(a_i)/\bar{f}$ copies of a_i to M.
   (c) **For** $j \leftarrow 1$ **to** $n/2$ **do**
       Select a pair a_j and a_k from M and do $O = O \cup crossover(a_j, a_k)$ with probability P_c.
   (d) **For** $i \leftarrow 1$ **to** n **do**
       For $j \leftarrow 1$ **to** d **do**
           Switch the j-th bit in $a_i \in O$ with probability P_m.
   (e) Update the population $\Pi \leftarrow Combine(\Pi, O)$.

In the above algorithm f is the so-called *fitness* function and

$$\bar{f} = \sum_{i=1}^{n} f(a_i)/n.$$

In the form of genetic algorithms that were originally designed for modeling biological evolution, the crossover operator, $crossover(\cdot,\cdot)$, implements exchanges of information among chromosomes. In particular, if a chromosome is represented by a binary string (as in feature selection), crossover can be implemented by randomly choosing a point, called the *crossover point*, at which two chromosomes exchange their parts to create two new chromosomes. For instance, given two strings, 00100101 and 10111010, the crossover operator can cut them in the middle and, as a result, $crossover$(00100101, 10111010) will produce two new chromosomes: 00101010 and 10110101.

When the crossover operator provides new solution points for further evaluation, it serves two complementary search functions. First, it creates new structures (e.g., the substring ##1111##, where # means 'don't care'), which were not present in parent chromosomes. These structures, when evaluated and accepted, create a new track toward the optimal solution. Second, this operator retains old structures within new solution points (for instance, 0010####), which, if accepted within the newly created solution, makes the presence of this structure in the population stronger. As a result, this structure has greater chance to survive and proliferate within the future solutions.

Mutation is a secondary search operator which increases the variability of the population. In our examples involving bit strings a mutant can be created by changing at random one or more bits in the structure.

The new population is created by combining the old population and the offspring, which is symbolically denoted as $\Pi \leftarrow Combine(\Pi, O)$. There are a number of possible implementations of this procedure, ranging from more radical, like $\Pi \leftarrow O$, to less restrictive, like "select n best chromosomes from Π and O." The

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

fitness function is another key element in the efficient application-oriented version of the genetic algorithm. The execution and performance of genetic search is also determined by a number of parameters, some of them specific for distinct implementations of genetic algorithms. However, the *population size, crossover rate* and *mutation rate* are common for all implementations. The crossover rate is the probability of accepting an eligible pair of chromosomes for crossover. The mutation rate is the probability of switching bits in the chromosomes. The crossover rate usually assumes high values, close or equal to one, while the mutation rate is typically small (1 to 15%).

3. Applying GA to Feature Selection

In our research on the use of genetic algorithms in feature selection we followed the formulation of the search problem utilized by Narendra and Fukunaga [3], and later by Foroutan and Sklansky [4] in their work on the branch and bound procedure. This approach assumes that we seek the smallest or the least costly subset of features for which the classifier's performance does not deteriorate below a certain specified level. When the error of a classifier is used to measure the performance, a subset is defined as *feasible* if the classifier's error rate is below the so-called *feasibility threshold.* We search for the smallest subset of features among all feasible subsets.

In this form feature selection is a constrained optimization problem, not readily handled by genetic algorithms. To make the constrained optimization suitable for genetic search we introduce the following *penalty function*:

$$p(e) = \frac{\exp((e-t)/m) - 1}{\exp(1) - 1}, \tag{3.1}$$

where e is the error rate, t is the feasibility threshold and m is a scale factor (referred to as the 'tolerance margin'). Note that the penalty function is monotonic with respect to e. If $e < t$ then $p(e)$ is negative and, as e approaches zero, $p(e)$ slowly approaches its minimal value:

$$p(0) = \frac{\exp(-t/m) - 1}{\exp(1) - 1} > -\frac{1}{\exp(1) - 1}. \tag{3.2}$$

Note also that $p(t) = 0$ and $p(t + m) = 1$. For greater values of the error rate the penalty function quickly rises toward infinity. We add this penalty function to the number of features in the evaluated subset to produce the score $J(a)$:

$$J(a) = l(a) + p(e(a)), \tag{3.3}$$

where $a = (\alpha_1, \ldots, \alpha_d)$ is a bit string representing a feature subset; α_i is a binary number set to 0 if the i-th feature is not present in the subset and set to 1 if it is included in the subset; and $l(a)$ is the level in the lattice occupied by a, defined in (1.2). Note that $l(a)$ represents approximately the cost of extracting features (assuming that the costs of extracting the individual features are equal).

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Recall the properties of the penalty function defined in (3.1) and note the following:

(a) Feature subsets (i.e., chromosomes in the terminology of GA) for which the error rate is below the feasibility threshold receive a small reward (i.e., a negative penalty).

(b) Feature subsets at the same level are validated according to the error rates associated with them: the ones with the lower error rates are better adapted (adaptation is measured by the score (3.3)).

(c) Feature subsets for which the error rate is above the threshold t but below $t + m$ receive a small penalty (between 0 and 1), which makes them about as well adapted as the subsets at one level higher. As a result, it is possible that a subset at level k will be considered better than a subset at level $k + 1$.

(d) Feature subsets for which the error rate exceeds $t + m$ receive a relatively high penalty (over 1) and cannot compete with subsets at the next higher level in the feature selection lattice.

Based on these properties of the score function (3.3) we anticipated that the genetic algorithm that would use it to validate the population of chromosomes should rapidly converge to the feasibility/infeasibility border in the feature selection lattice, and thereafter to operate in the vicinity of this border. In the next section we describe experiments which confirmed this anticipated behavior of our genetic algorithm.

Going back to the definition of the ordinary genetic algorithm given in Section 2, we need to define the fitness function and the $Combine(\cdot,\cdot)$ operator. Let $\Pi = \{a_1, \ldots, a_n\}$ denote a population of feature selection vectors. Thus each feature selection vector represents a chromosome. Since in our case we look for a minimum of the score (3.3), we define the fitness function as

$$f(a_i) = (1 + \varepsilon) \max_{a_j \in \Pi} J(a_j) - J(a_i), \tag{3.4}$$

where ε is a small positive constant which assures that $\min f(a_i) > 0$, i.e., even the least fit chromosome is given a chance to reproduce. The $Combine(\cdot,\cdot)$ operator selects the best n chromosomes from $\Pi \cup O$.

4. Experimental Study

The intent of the experiments reported here was first to evaluate GA as a tool for feature selection in competition with classical procedures for small scale problems, and then to evaluate it for large scale problems. By 'small scale problems' we refer to cases where the feature selection task can be handled in a reasonable time by exhaustive search or by a branch and bound search, as in the Foroutan–Sklansky formulation. In practice the initial number of features, d, in small scale problems never exceeds 20. We refer to feature selection problems in which the initial number of features exceeds 20 as *large scale.*

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

In an ideal experimental situation it would be desirable to test our feature selection procedure on a group of real data sets derived from various applications (e.g., medical, industrial or military), because that would give the results of such testing credibility and statistical validity. However, for practical reasons any extensive testing involving real data is prohibitive. Therefore, we followed the following path in our experiments:

(a) First, we compared sequential search, branch and bound and our procedure on simulated data of moderate dimensionality. Where possible, we included exhaustive search in these tests. The speedups obtained by replacing the training of a classifier and the estimation of its error rate by a simulation of the error rate were in the range of 10^3 to 10^4.

(b) Next, to verify the results obtained on the simulated data we performed limited tests on real data.

4.1. *Experiments on Simulated Data*

To simulate the conditions encountered by feature selection procedures in practice we replaced the true error rate function with a model of the classifier's error rate. Building adequate approximations of the error rates of some types of classifiers is possible, but in practice it does not lead to substantial savings in the computation time. For instance, there is an analytic formula for the error rate for the optimal classifier for two Gaussian distributions in the feature space with equal covariance matrices. However, the computation or approximation of this formula requires inverting the common covariance matrix, which yields the time complexity of $O(d^3)$. Not only is this complexity too high for our purposes, but the assumed form of class distributions in the feature space is so restrictive that the error rate function is hardly representative of any real application.

To resolve this dilemma we modeled the error rate as a function of the feature selection vector, accounting for various interactions among the features and various defects of the classifier training procedures — but without assuming any forms of underlying distributions in the data.

Let $a = (\alpha_1, \ldots, \alpha_d)$ denote a chromosome or feature selection vector representing a feature subset in a d-dimensional space. Recall that in this vector $\alpha_i = 0$ means that the i-th feature is not included in the subset and $\alpha_i = 1$ means that the i-th feature belongs to the subset. Let $\tilde{\alpha}_i$ denote $1 - \alpha_i$.

The error rate function can be represented as

$$e(a) = \sum_{j=1}^{2^d} m_j(a) e_j, \tag{4.1}$$

where

$$m_j(a) = \begin{cases} 1 & \text{if } j = \text{bin}(a), \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

and bin(a) is a function that transforms the binary vector a into a corresponding decimal equivalent (e.g., bin(011) = 3). But this form merely provides a means for expressing the error rate as a function. It does not reflect various phenomena observed in the behavior of error rate function encountered in practice. Consequently, instead of the above formulation we have built our error rate model around the k-degree algebraic form

$$e = e_0 + \sum_{i_1,\ldots,i_k} w_{i_1,\ldots,i_k} \tilde{\alpha}_{i_1} \ldots \tilde{\alpha}_{i_k} \tag{4.3}$$

where e_0 is a constant and $w_{i_1,\ldots,i_k}$ are constants. For $k = 1$ this definition reduces to a linear form

$$e = e_0 + W^T \tilde{a}, \quad \text{where} \quad W = [w_1, \ldots, w_d]^T. \tag{4.4}$$

For $k = 2$, Eq. (4.3) becomes a quadratic form

$$e = e_0 + \tilde{a}^T W \tilde{a}, \quad W = [w_{ij}], \quad i, j = 1, \ldots, d. \tag{4.5}$$

By Eq. (4.3) a constant $w_{i_1,\ldots,i_k}$ is added to the base error rate e_0 if the features corresponding to $\alpha_{i_1} \cdots \alpha_{i_k}$ are not included in the current feature subset. Since Bayes theory states that the error rate function should increase when features are dropped (this effect is known as the *monotonicity property* of the *Bayes classifier*), the constants $w_{i_1,\ldots,i_k}$ should be nonnegative. In that sense these constants represent the penalty for dropping the information contained in features. Hence we call these constants *penalty factors.*

In practice classifiers do not behave exactly as their Bayesian models. Training procedures used to construct the classifiers are sensitive to various anomalies in the data. The data itself may be underrepresented—which may result from a small sample size, unfavorable size/dimensionality ratio or both. These properties of the data affect the chances for the success or failure by the training procedures to construct the optimal classifier for the given data. Consequently, when a feature or a group of features is dropped from the data, the error rate of a classifier trained on the remaining features may *decrease.* In our error rate model we simulate several possible causes of deviations from strict monotonicity in real classifiers. We discuss these simulated deviations below.

We allow a certain fraction of the penalty factors $w_{i_1,\ldots,i_k}$ to be negative. These negative constants represent the improvements in the performance of a training procedure when noise-inducing features are removed. Such an effect occurs often in the k-NN rule, which requires comparisons of distances to perform classification. If a feature with low discriminatory power has much larger variance than other features with inherently high discriminatory power, it is very likely to suppress the influence of the otherwise strong features on the overall classification results. When this irrelevant feature is dropped, the features with high discriminatory power can

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

be more fully utilized by the k-NN procedure and yield smaller error rates. The negative penalty factors occur with a probability which we call the *defect probability.*

Beside strictly data-dependent effects we also observe in practice *training defects.* This class of defects covers situations when the training procedure does not produce the best classifier for the data. An example here would be any iterative procedure (e.g., the *window training procedure* [9]), where the error rate varies stochastically and for which the stopping rule is inaccurate. The error in estimating the error rate based on the evaluation of the trained classifier on the test data also varies stochastically. We simulate these training defects by imposing an extra variance on the error rate obtained from (4.5):

$$e = e_0 + \sum_{i_1,\ldots,i_k} w_{i_1,\ldots,i_k} \bar{\alpha}_{i_1} \cdots \bar{\alpha}_{i_k} + \varepsilon, \tag{4.6}$$

where ε has a normal distribution $N(0, \varepsilon_0)$. We call the constant ε_0 the *estimation error level.*

Additionally we include a number of local defects associated with specific subsets of features that are significantly larger than those generated by (4.5). Those additional defects are determined by indicating their absolute value and the combinations of features for which they occur. Thus our full representation of the error rate is as follows:

$$e = e_0 + \sum_{i_1,\ldots,i_k} w_{i_1,\ldots,i_k} \tilde{\alpha}_{i_1} \ldots \tilde{\alpha}_{i_k} + \varepsilon + \sum_{j=1}^{s} v_j \hat{\alpha}_{i_1} \ldots \hat{\alpha}_{i_d}, \tag{4.7}$$

where $\hat{\alpha}_i$ can stand for α_i, $\tilde{\alpha}_i$ or a 'don't care' case. Note that while the first sum consists of k-element binary feature code multiplications, the second one may use even all features to determine defects. Those features that are marked as 'don't care' are dropped from the products. Thus the products describing the local defects may have a varying number of features involved.

We implemented and tested the error rate model described above. In particular we generated a quadratic-type error function for 24 features. Beside the GA we tested three other procedures: exhaustive search, (p, q)-search introduced by Stearns [7] as a generalization of sequential search techniques, and the branch and bound as modified by Foroutan and Sklansky [4]. Our objective was to find the smallest (i.e., containing the smallest number of elements) subset of features for which the error rate does not exceed a given threshold. Below we describe the results of our test on these four algorithms.

Exhaustive search

The results of the exhaustive search of the 2^{24}-node feature selection lattice are given in Table 1. In this table we list minimum and maximum errors for each level in the feature selection lattice and feature subsets (the rightmost column)

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Table 1. Exhaustive search results.

Level	Min. error	Max. error	Best subset (min. error)
1	0.349	0.400	00000000000000000010000
2	0.320	0.397	00101000000000000000000
3	0.297	0.386	00101000000000000010000
4	0.273	0.370	01100000000110000000000
5	0.248	0.354	01100100000110000000000
6	0.228	0.347	01100100000110000010000
7	0.211	0.338	01100100000110000010100
8	0.195	0.326	01100100000110000010110
9	0.182	0.314	01100100000110100010110
10	0.169	0.302	01100100000110110010110
11	0.158	0.290	01100100000110110011110
12	0.148	0.279	01100100000110110111110
13	0.139	0.269	01100111000110010010111
14	0.130	0.259	01100101110011001010111
15	0.122	0.250	01100101110011001110111
16	0.116	0.241	01100101110011010111110111
17	0.111	0.235	01100101110011010111111111
18	0.113	0.229	01000111111011010111111111
19	0.109	0.224	01101111110111001111111111
20	0.106	0.218	01101111110111010111111111
21	0.106	0.214	01101110111111111111111111
22	0.108	0.208	01101111111111111111111111
23	0.113	0.168	10111111111111111111111111
24	0.124	0.124	11111111111111111111111111

corresponding to the minimum errors. It took 95,214 seconds (approximately 26 hours) of CPU time on a MicroVAX II to complete the search. Note that any additional feature would increase this more than twofold (the number of nodes would double and the time needed to compute a single error rate value would increase slightly).

Branch and bound

Recall that the threshold t, used to induce backtracking in branch and bound, determines the size of the feasible region and that if the constraint criterion (e.g., the error rate) has the monotonicity property, branch and bound performs exhaustive search in the feasible region. The Foroutan–Sklansky modified branch and bound (BB) algorithm, which allows for certain departures from strict monotonicity, requires additionally that we set the tolerance margin m (see [4]). Effectively, the size of the feasible region is determined by $t + m$. To avoid an excessively large feasible region we chose the threshold $t = 12.5\%$ and $m = 1\%$. From Table 1 we see

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com
by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

that the feasible region would contain no more than

$$n_f = \sum_{i=14}^{24} \binom{24}{i} \approx 7 \cdot 10^6$$

nodes, since exhaustive search found a 14-element feature subset with an error rate of 13.0% and no subset at the level 13 had an error rate smaller than 13.9%. Recall that the level in the lattice at which a node is placed is equivalent to the number of features the corresponding subset contains.

The Foroutan–Sklansky modified branch and bound algorithm did not find a feasible subset at the level 15 in the feature selection lattice. After executing 134,122 criterion function evaluations, which yields 99.2% savings over exhaustive search, it completed its search. The results of branch and bound search are given in Table 2. In this table we show only the levels that were visited by the branch and bound procedure. The rightmost column, titled 'Excess error,' contains the amount by which the error rate obtained from branch and bound exceeds the minimal achievable error rate at the same level. Note that the branch and bound algorithm found optimal subsets only at the levels 20, 23 and 24.

Table 2. The Foroutan–Sklansky BB-search.

Level	Min. error	Best subset (min. error)	Excess error
14	0.142	00101100000110011111111	0.012
15	0.132	01100111010011010111110	0.010
16	→ 0.124	11110110001010110111110	0.008
17	0.119	11100100111110011111101	0.008
18	0.114	11100100111111010111111	0.001
19	0.110	11100100111111010111111	0.001
20	0.106	01101111110110111111111	0.000
21	0.114	10111111111111010111110	0.008
22	0.113	10111111111110111111111	0.005
23	0.113	10111111111111111111111	0.000
24	0.124	11111111111111111111111	0.000

As we see from Table 2, the best subset satisfying the error-rate constraint was located at the level 16 (marked by an arrow). The best subset found by branch and bound at the level 15 has an error rate of 13.2 %, which is 1% higher than the minimal achievable error at this level. While the one-percent difference between the error rate corresponding to the optimal subset and the error rate of the best subset actually detected by branch and bound at the level 15 may not seem significant in practice (as the estimated error rates are always corrupted by the imperfect estimation process itself), it nevertheless signals a deeper problem in the actual performance of this procedure. Apart from the already huge number of visited nodes, the Foroutan–Sklansky modification of branch and bound still did not deal

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

adequately with the nonmonotonicities in the feature selection lattice despite the fact that $t + m$ should have forced it to search all subsets at the level 14 and 15.

(p,q)-search

The (p, q)-search procedure also failed to find the optimal solution. We tested several settings for p and q (e.g., (2,1), (1,0), (0,1), (3,2), etc.) but for none of them was the excess error rate less than 1 %. The results of (1,0)-search, which is equivalent to sequential backward selection, are given in Table 3(a). The results of (2,1)-search are shown in Table 3(b). The number of criterion function evaluations was 301 for (1,0)-search and 433 for (2,1)-search.

Table 3(a). Sequential backward selection — (1,0)-search.

Level	Min. error	Best subset (min. error)	Excess error
1	0.371	00100000000000000000000	0.022
2	0.325	00100100000000000000000	0.005
3	0.311	00100100000010000000000	0.014
4	0.281	00100110000010000000000	0.008
5	0.267	00100111000010000000000	0.019
6	0.252	10100111000010000000000	0.024
7	0.241	10100111001010000000000	0.030
8	0.234	10100111001110000000000	0.039
9	0.208	10100111011110000000000	0.026
10	0.195	10100111011111000000000	0.026
11	0.192	10100111111111000000000	0.034
12	0.187	10101111111111000000000	0.029
13	0.164	10111111111111000000000	0.025
14	0.153	10111111111111000000100	0.023
15	0.143	10111111111111000010100	0.021
16	0.135	10111111111111000010110	0.019
17	0.128	10111111111111000110110	0.017
18	→0.123	10111111111111000111110	0.010
19	0.118	10111111111111010111110	0.009
20	0.116	10111111111111010111110	0.010
21	0.114	10111111111111011111110	0.008
22	0.113	10111111111111011111111	0.005
23	0.113	10111111111111111111111	0.000
24	0.124	11111111111111111111111	0.000

Genetic search

The margin used by the penalty function was set to $m = 1\%$ as in the branch and bound procedure. Although when these tests were conducted we did not know any theoretical justification for assigning any particular value to m, this value seemed

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Table 3(b). The Stearns algorithm — (2,1)-search.

Level	Min. error	Best subset (min. error)	Excess error
1	0.349	00000000000000000010000	0.000
2	0.325	00100100000000000000000	0.005
3	0.302	00100100000000000010000	0.005
4	0.281	00100110000010000000000	0.008
5	0.260	00100110000010000010000	0.012
6	0.252	10100111000010000000000	0.024
7	0.234	10100111000010000010000	0.023
8	0.234	10100111001110000000000	0.039
9	0.208	10100111011110000000000	0.026
10	0.195	10100111011111000000000	0.026
11	0.181	10100111011111000010000	0.023
12	0.187	10101111111111000000000	0.039
13	0.164	10111111111111000000000	0.025
14	0.153	10111111111111000000100	0.023
15	0.143	10111111111111000010100	0.021
16	0.135	10111111111111000010110	0.019
17	0.128	10111111111111000110110	0.017
18	→0.123	10111111111111000111110	0.010
19	0.118	10111111111111010111110	0.009
20	0.116	10111111111111010111110	0.010
21	0.114	10111111111111011111110	0.008
22	0.113	10111111111111011111111	0.005
23	0.113	10111111111111111111111	0.000
24	0.124	11111111111111111111111	0.000

to work properly on a wide range of tests. In addition we tested other values of the threshold and did not see any significant correlation between t and m.

The genetic algorithm implemented to satisfy the specification given in the previous section executed a moderate number of criterion function evaluations (1000 to 3000) before it found the optimal (!) solution. We tested a broad range of values of parameters (*crossover rate, mutation rate* and *population size*) that control the performance of the genetic algorithm. For almost all cases the optimal solution was found in the first 1000 to 3000 criterion function evaluations. Each test consisted of 25 runs of genetic search for the same parameter setting but for different initial populations. To suppress the dependency of the results on the initial population, the results of these runs were averaged and also standard deviations were calculated. Each of the runs performed up to 3000 error evaluations (that is, it visited up to 3000 nodes). The results of these comparative tests are presented in Table 4. In this table the population size is listed in the first column and the mutation rate in the next column. The crossover rate for all listed tests was equal to 1.0. The following six columns list the error rates (columns titled E14, E15 and E16) and their standard deviations (V14, V15, V16) for the levels of interest, that is, 14, 15 and 16. As

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Table 4. Genetic algorithm: an overview.

n	mutation rate	E14	V14	E15	V15	E16	V16
10	0.10	0.135	0.004	0.127	0.004	0.121	0.004
10	0.15	0.134	0.004	0.126	0.004	0.120	0.004
10	0.20	0.134	0.004	0.126	0.004	0.121	0.004
14	0.10	0.134	0.004	0.127	0.004	0.120	0.004
14	0.15	0.134	0.004	0.126	0.004	0.119	0.004
14	0.20	0.133	0.004	0.126	0.004	0.120	0.004
20	0.05	0.134	0.004	0.127	0.004	0.120	0.004
20	0.10	0.135	0.004	0.127	0.004	0.120	0.004
20	0.15	0.133	0.004	0.126	0.004	0.120	0.004
20	0.20	0.134	0.004	0.127	0.004	0.120	0.004
30	0.05	0.135	0.004	0.128	0.004	0.121	0.004
30	0.10	0.133	0.004	0.125	0.004	0.119	0.004
30	0.15	0.133	0.004	0.126	0.004	0.120	0.004
30	0.20	0.134	0.004	0.127	0.004	0.121	0.004
40	0.05	0.134	0.004	0.127	0.004	0.120	0.004
40	0.10	0.134	0.004	0.127	0.004	0.121	0.004
40	0.15	0.134	0.004	0.127	0.004	0.120	0.004
40	0.20	0.135	0.004	0.127	0.004	0.122	0.004
50	0.05	0.135	0.004	0.128	0.004	0.121	0.004
50	0.10	0.135	0.004	0.127	0.004	0.121	0.004
50	0.15	0.135	0.004	0.127	0.004	0.122	0.004
50	0.20	0.135	0.003	0.127	0.003	0.122	0.004
76	0.05	0.136	0.004	0.128	0.004	0.121	0.004
76	0.10	0.135	0.004	0.128	0.004	0.122	0.004
76	0.15	0.137	0.004	0.128	0.004	0.122	0.003
76	0.20	0.137	0.004	0.129	0.004	0.123	0.004
		Minimal achievable errors at these levels					
		0.130		0.122		0.116	

we see, the error obtained by the best performing search procedures does not exceed the minimal error rate by more than 0.4 %. The corresponding standard deviations are equal to 0.4 %, which implies that in several cases the optimum feature subsets were found by our genetic algorithm.

As an example, the result of search based on a 10-element population, with the crossover rate equal to 100% and mutation rate equal to 15 % is presented in Table 5. Note that the excess error at the optimal level, marked by an arrow, is only 0.1 %.

As we mentioned before, we expected the genetic algorithm to concentrate on searching the boundary between the feasible and infeasible nodes in the feature selection lattice. This effect is confirmed in Table 5. The second column in this table lists the number of nodes (feature subsets) visited at each level. We see here that most of the visited nodes are at levels 13 to 16.

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com
by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Table 5. GA: an example run.

Level	Visited nodes	Min. error	Best subset (min. error)	Excess error
8	1	0.226	00100100110011010000000	0.031
9	12	0.192	01100000100010011010100	0.010
10	52	0.177	01100101000011010011010	0.008
11	169	0.163	01100101000011011001011	0.005
12	349	0.149	01100111100011000011011	0.001
13	551	0.139	01100101110011001101011	0.000
14	702	0.131	01100101110011011011011	0.001
15	624	→0.123	01100101110011011011011	0.001
16	422	0.116	01100101110011011111011	0.000
17	208	0.119	11110100011011011111111	0.008
18	89	0.124	11111110011111011001101	0.011
19	34	0.119	11100111110011011111111	0.010
20	7	0.115	11111110011111011111111	0.009
21	6	0.116	11111111010111011111111	0.010
22	4	0.126	11111111111111111001111	0.018

4.2. *Experiments on Real Data*

In order to verify the power of our genetic feature selection procedure we devised a 5-NN classifier trained on digitized infrared imagery of real scenes. This data was provided by the U.S. Army. It consisted of 150 30-dimensional feature vectors, each vector belonging to one of two classes. The purpose of this experiment was to compare our method with sequential methods, particularly with the Stearns (p, q)-search procedure, which at this dimensionality of the problem is the only algorithm able to give solutions in reasonable time. The k-NN rule is known to react strongly to undersampling (i.e., the situation when the training set is small)[1] and the scaling problem (i.e., the problem of scaling features). To avoid the latter we scaled all features to a unit standard deviation. The influence of undersampling manifests itself in significant nonmonotonicity of its error rate function. This effect is even stronger if the parameter k (the number of neighbors checked for classification) is fixed. Consequently, we expected that the resulting error rate function would be ill-behaved and would present difficulties to search algorithms.

Our procedure in this experiment was as follows. First, we ran sequential backward selection, which is equivalent to (1,0)-search in Stearns' notation. It is important to note here that sequential algorithms produce a path through all the levels of the feature selection lattice. The error rates along this path can be drawn versus the levels in the lattice (see Fig. 2). Based on the observation of the path through the feature selection lattice found by backward selection we chose the threshold $t = 0.145$ for our genetic feature selector. By selecting $t = 0.145$ we intended to

[1]For discussions of undersampling and related phenomena we refer the reader to [1] and [8].

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

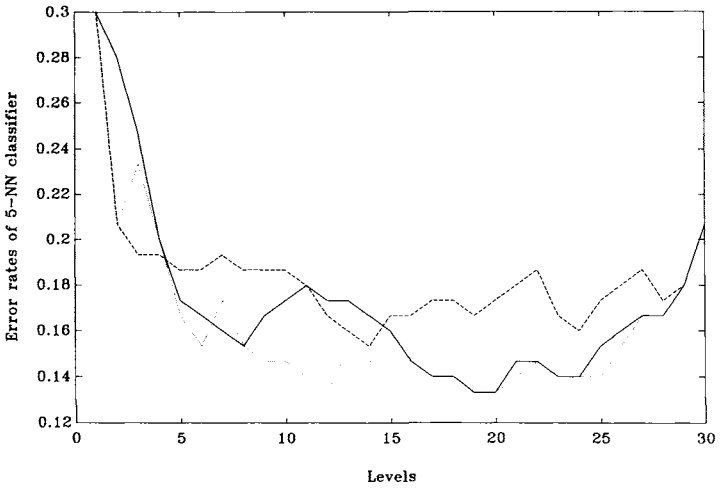

Fig. 2. Paths obtained from sequential procedures: the solid line denotes (1,0)-search (sequential backward selection), the dashed line denotes (0,1)-search (sequential forward selection), and the dotted line denotes (2,1)-search.

reduce the size of the feasible region, hereby making it more difficult for the genetic algorithm to find. In particular, according to the result obtained from backward selection the feasible region was placed between levels 17 and 20, and also at levels 23 and 24.

Our next step was to execute the genetic algorithm. Based on the results of simulations with the aid of the modeled error rate functions described in the previous section we chose the population size equal to 40, the mutation rate equal to 0.1 and we began search at level 2, creating the initial population with the aid of a random number generator. We ran our genetic algorithm 9 times. In each run the genetic algorithm was terminated when the number of tested subsets exceeded 2000.

Our final tests were done with the aid of the Stearns (2,1)-search algorithm and sequential forward selection (i.e., (0,1)-search in Stearns'-notation). The number of nodes (feature subsets) tested was 465 by forward and backward selection, and 1395 by (2,1)-search.

In Fig. 2 we summarize the results obtained from sequential search procedures. In this figure the solid line corresponds to the path obtained from (1,0)-search, the dashed line comes from (0,1)-search and the dotted line denotes (2,1)-search. As expected the (2,1)-search procedure provided the best results (the lowest error rates) for all levels except 3, 4 and 7. The (1,0)-search algorithm found the best feasible subset consisting of 17 features. The (2,1)-search algorithm found a smaller feasible subset, consisting of 11 features. The (0,1)-search algorithm did not find a feasible subset at all.

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com
by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

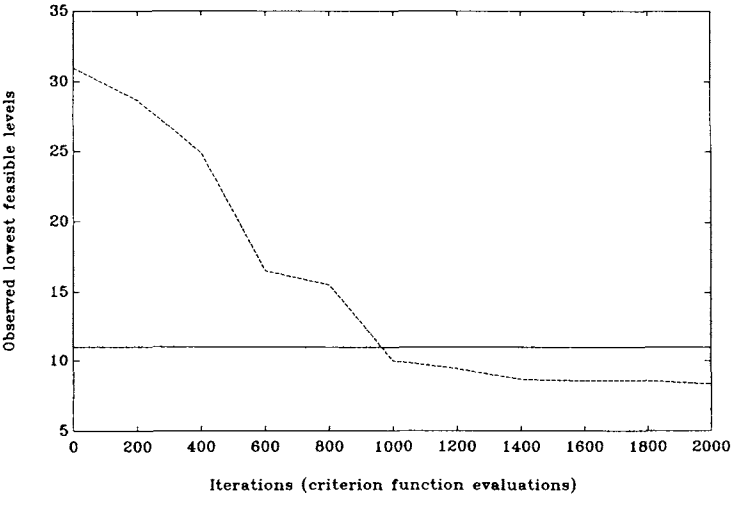

Fig. 3. An averaged convergence curve of genetic search for 5-NN rule.

An average of 9 runs of our procedure starting with different initial populations is presented in Fig. 3. In this figure we plot the observed lowest feasible level versus the number of executed iterations. The solid line denotes the lowest feasible level, equal to 11, found by (2,1)-search. The dashed line denotes the progress of genetic search. As we see from the figure, the genetic feature selection procedure found a feasible subset at the level 11 after approximately 1000 iterations. This compares favorably with the number of iterations executed by (2,1)-search. On average, the lowest feasible level detected by our method was equal approximately to 8. In one run the genetic algorithm found a node with error rate 0.133 at the level 7, that is, it reduced the size of the best feasible subset of features by four features.

5. Concluding Remarks

In this communication we discussed the use of genetic algorithms for selecting features for statistical pattern classifiers when the initial number of features exceeds 20. In our formulation, which follows the formalism used earlier by Narendra and Fukunaga [3] and later by Foroutan and Sklansky [4], the feature selection problem is a constrained optimization problem. In accordance with this formulation we adapted the ordinary genetic algorithm to the problem of feature selection by incorporating a penalty function into the criterion function or 'score.'

Our feature selection procedure based on the modified genetic algorithm was subject to numerous experiments described in detail in Section 4. We showed, on both real and modeled error rate functions, that our feature selection procedure outperforms all other nonexhaustive methods, in particular sequential search and branch

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com
by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

and bound. In particular, on the 24-dimensional feature selection lattice with modeled error rate our method exhibited better results than branch and bound while yielding computational savings of two orders of magnitude over branch and bound. On 30-dimensional real data the genetic algorithm also outperformed sequential search (branch and bound was not tested because of its prohibitive time complexity), finding smaller feature subsets after exploring similar number of nodes. This advantage is more prominent when the dimensionality of the initial feature space is large, since the sequential procedures have quadratic time complexity and genetic search exhibits slightly more than linear increase of time complexity.

The success of the feature selection procedure based on the genetic algorithm can be attributed to the following facts:

(a) Branch and bound performs exhaustive search in the feasible region.

(b) Sequential search performs locally exhaustive search by testing all connected subsets on its way through the lattice. There is also no theoretical indication for what values of p and q the (p, q)-search procedure is going to yield the best results, so that multiple runs are needed.

(c) In contrast, the genetic algorithm performs directed scanning of the lattice and searches only the most promising parts of the lattice. Since during genetic search a set of candidate solutions is processed at the same time, the genetic algorithm is more likely to find the optimal solution than any other method that modifies and evaluates a single solution at a time. This inherent parallelism of the genetic algorithm results also in a very high efficiency of search, which, in terms of the number of visited nodes, rivals that of sequential search.

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com
by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.

Acknowledgements

We are indebted to Richard Sims of the U.S. Army Missile Command for discussions and comments on the use of genetic algorithms for the design of classifiers. The research reported here was supported by the U.S. Army Research Office under Contract DAAG29-84-K-0208.

References

[1] P. A. Devijver and J. Kittler, *Pattern Recognition: A Statistical Approach*, Prentice-Hall, London, 1982.

[2] W. Siedlecki and J. Sklansky, On automatic feature selection, *Int. J. Pattern Recognition and Artificial Intelligence* **2** (1988) 197–220.

[3] P. M. Narendra and K. Fukunaga, A branch and bound algorithm for feature subset selection, *IEEE Trans. Computers* **26** (1977) 917–922.

[4] I. Foroutan and J. Sklansky, Feature selection for automatic classification of non-Gaussian data, *IEEE Trans. Syst. Man Cybern.* **17** (1987) 187–198.

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

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

[7] S. D. Stearns, On selecting features for pattern classifiers, *Proc. of the Third Int. Conf. on Pattern Recognition*, Coronado, CA, 1976, 71–75.

[8] A. K. Jain and R. Chandrasekaran, Dimensionality and sample size considerations in pattern recognition practice, in P. R. Krishnaiah and L. N. Kanal (eds.), *Handbook of Statistics*, Vol. 2 (*Classification, Pattern Recognition and Reduction of Dimensionality*), North-Holland, New York, 1982.

[9] J. Sklansky and G. N. Wassel, *Pattern Classifiers and Trainable Machines*, Springer-Verlag, New York, 1981.

Handbook of Pattern Recognition and Computer Vision Downloaded from www.worldscientific.com
by AUSTRALIAN NATIONAL UNIVERSITY on 06/01/19. Re-use and distribution is strictly not permitted, except for Open Access articles.