# Feature Selection using Genetic Algorithms and Probabilistic Neural Networks

**Content Areas: neural networks, genetic algorithms, data mining**
Tracking Number: A694

## Abstract

Selection of input variables is a key stage in building predictive models, and an important form of data mining. As exhaustive evaluation of potential input sets using full non-linear models is impractical, it is necessary to use simple fast-evaluating models and heuristic selection strategies. This paper discusses a fast, efficient, and powerful non-linear input selection procedure using a combination of Probabilistic Neural Networks and repeated bitwise gradient descent. The algorithm is compared with forward elimination, backward elimination and genetic algorithms using a selection of real-world data sets. The algorithm has comparative performance and greatly reduced execution time with respect to these alternative approaches. It is demonstrated empirically that reliable results cannot be gained using any of these approaches without the use of resampling.

## 1 Introduction

In many machine learning domains, the objective is to infer a model that allows one or more output (dependent) variables to be predicted given the values of input variables (independent variables, or *features* – we will use the terms input variable and feature interchangeably throughout this paper). A wide variety of modeling techniques can be used to form the prediction, including conventional statistical models such as linear (least squares) models and logistic regression, clustering algorithms, neural networks, fuzzy logic and neuro-fuzzy techniques, and decision trees. Irrespective of the modeling technique, a key issue is to determine which of the available input variables should be used in modeling - this is known as *feature selection*. Typically, the model must be inferred from a set of historical data, $D$, that includes a number, $N$, of cases (or vectors), $c_j$, each containing values for an output variable, $o_j$, together with the associated vector of $V$ input variables, $x_j$, and the feature subset must be selected on the basis of this same data set.

Feature selection is non-trivial for a number of reasons. First, variables are seldom entirely independent – there may be redundancy (where two or more variables are correlated so that it is not necessary to include all of them in modeling – the most extreme example occurs when a variable is simply replicated), and interdependence (where two or more variables between them convey important information that is obscure if any of them is included on its own – the well-known two-spirals problems demonstrates the case of two interdependent variables). Second, it may actually be beneficial to discard variables that have some low level of genuine information, as the "curse of dimensionality" implies that smaller models generalize better, and we often encounter the problem where the number of cases available is small with respect to the number of variables.

As a consequence of these problems, the only way to select the optimal feature subset with certainly is to evaluate all possible combinations, of which there are $2^V$ for a $V$ variable problem. This means building $2^V$ models, and if the modeling process itself is subject to experimental variability (for example, knowing the input variables to a neural network, we must still determine the number of hidden units, and train many times to avoid local minima) then each of the $2^V$ evaluations of variable subsets may itself be extremely computationally expensive. Even if exhaustive evaluation is possible, the variable subset selected may be dependent on the training data used, which is itself a sample from an unknown distribution, and therefore the results are unreliable.

In reality, exhaustive evaluation is not practical for more than a few input variables. It is common practice to apply heuristic algorithms based on a smaller number of evaluations, such as forward stepwise and backward stepwise selection.

We may also reduce the computational burden by performing feature selection using some quick to evaluate model; for example, by using a linear model for feature selection even if the model that will ultimately be deployed is non-linear [Jain and Zongker, 1997].

This paper discusses the application of a feature selection algorithm using a combination of repeated bitwise gradient descent and Probabilistic Neural Networks [Speckt, 1990]. The algorithm is compared with forward stepwise, backward stepwise, and genetic algorithms. The algorithms are evaluated using a selection of real-world data sets drawn from the UCI machine learning repository [Blake *et. al.*, 1998]. The new algorithm is shown to be effective in selecting feature

subsets, and extremely efficient. In addition, it can be used to differentiate between important and ambiguous variables.

## 2. Evaluation of Feature Subsets

The feature selection task can be conveniently represented as a binary string search problem. A feature selection algorithm searches for a binary string, *S*, with the number of bits equal to the number of candidate input features, *V*; $s_i=0$ indicates that a feature should not be used; $s_i=1$ indicates that it should be used. Such a string is sometimes referred to as a mask. Any given mask can be evaluated by building a model using the indicated combination of inputs, and assessing its performance. Feature selection algorithms therefore have two key parts: a search algorithm that generates candidate mask strings, and an evaluation algorithm that assigns a performance rating to the strings (the performance may be used by the search algorithm to guide the generation of new candidate masks for evaluation).

This section describes the evaluation algorithm used in this paper; the next section describes the new search algorithm and the benchmark search algorithms.

Probabilistic neural networks (PNNs) are simple non-linear modeling techniques that have modest computational requirements for a reasonably small data set. Probabilistic neural networks are used for classification problems [Speckt, 1990], where the objective is to assign cases to one of a number of discrete classes. The output of the model is an estimate of the class membership probabilities. This paper concentrates on the application of feature selection in classification problems; however, the techniques describe extend trivially to the case of regression problems (where the output is a continuous variable) using Generalized Regression Neural Networks [Speckt, 1991], a closely related technique with similar performance characteristics.

Probabilistic Neural Networks estimate the probability density functions (p.d.f.s) using the training data set in a very direct fashion, and then assign class membership probabilities to new cases by using the p.d.f. estimates. The class p.d.f.s are estimated by adding together kernel functions (typically Gaussians) located at each case in the training set. Intuitively, the presence of a case in the training set can be taken as evidence of some probability density at that point, and of (somewhat lower) probability density at nearby points. Where there is a cluster of training cases belonging to the same class, the probability estimate for that class will be high as a number of overlapping kernel functions are added together.

The PNN estimates the probability that a new case, *x*, belong to class *i* as:

$$f_i(x) = \frac{1}{2\pi^{v/2}\sigma^{v}} \frac{1}{k_i} \sum_{j=1}^{k_i} \exp\left( \frac{(x - x_{ij})^T (x - x_{ij})}{2\sigma^2} \right)$$

where $x_{ij}$ is the $j^{th}$ training case belonging to class $i$, $k_i$ is the number of training cases in class $i$, and $\sigma$ is the smoothing factor, which is determined experimentally.

The PNN is constructed as a neural network using three layers: input layer, pattern units, and summation/output units. The input layer distributes inputs to the next layer. The pattern units each contain a weight vector that is a copy of a case from the training set. These units calculate the squared Euclidean distance of the pattern unit vector from the input, divide this by $2\sigma^2$, and then calculate the exponential of the negative of this. They thus form a Gaussian function centered at the training case. The third layer contains one unit for each class, and each of these units is connected only to pattern units of that class. The weights on these connections are all one. At the output layer, the activations are normalized to sum to one; thus, the constant in the formula above can be ignored. Modifications may also be made to account for known disparities between prior class distributions and the distribution in the training set, and to incorporate a loss matrix if the cost of misclassification varies from class to class.

The only variable that needs to be optimized in a PNN is the smoothing factor. This is easily done experimentally. The data set, *D*, is divided into two subsets: a training set, *T*, and a test set, *X* (typically with equal numbers of randomly selected cases). A line search algorithm is used to select the smoothing factor, by building a PNN for each smoothing factor, and assessing its performance using a composite error function applied to the test cases. The error function

$$E_\sigma = \sum_{X} \sum_{O} (f_i(x) - o_i)^2$$

might be the correct classification rate, the sum of the test case cross-entropies, or the sum-squared error function. This paper uses the latter, as it is commonly applied in the neural network community:

PNNs are not too sensitive to the precise choice of smoothing factor, and it is sufficient to optimize the parameter once before commencing the feature selection process; the computational burden is therefore negligible.

To evaluate a feature subset, we use the same error function, in this case with a fixed smoothing factor, but with the input variables varying.

From the point of view of feature selection, the PNN has significant advantages over other forms of neural network, and even over linear modeling. First, there is no "training algorithm" to speak of. A PNN is "trained" by recording the training cases in the hidden layer, and setting the connections to the output layer to indicate the class. It is not even necessary to do this - execution of a PNN can be simulated directly within the data set, so that there is no training phase

at all. This contrasts with neural networks such as standard multilayer perceptrons, which require an extensive period of training (Speckt [1990] quotes a 200,000 times speedup compared with back propagation on one particular problem). Second, once the smoothing factor has been fixed there are no variable training parameters, so that repeated execution is not necessary. Third, the PNN is non-linear and is capable of modeling arbitrarily complex problems. It is therefore a more intuitively appealing option than the commonly-selected alternative, linear modeling, if the problem domain is known or suspected to be non-linear.

The computational cost of evaluating a feature set using a PNN is, however, quite high. The execution time of the network is proportional to $N_T V$, where $N_T$ is the number of training cases. To evaluate a feature set, the PNN is executed on each of the test cases, so the total evaluation time is proportional to $N_T N_X V$, where $N_X$ is the number of verification cases. As $N_T$ and $N_X$ sum to $N$, the total number of cases available (typically half are used in training and half in verification), the execution time is proportional to $N^2 V$. This cost becomes quite significant if the number of cases is large, and is certainly much greater than the execution time for a linear or multilayer perceptron model. Nonetheless, given that training is the dominant element in the use of most neural networks, the disadvantage in execution time is greatly outweighed by the advantage in training time.

One approach to reducing computational cost is to subsample cases during evaluation. Execution speed may be increased by selecting a sub-sample of the cases, dividing this into a training and test set, and evaluating on that. As the computational cost is proportion to $N^2$, halving the sample size quarters the evaluation time, so the effect can be very significant. This is a valuable alternative if the data set contains many cases. Of course, reducing the number of cases used to form the model will also make the selection procedure more prone to random errors. However, the data sets we have used in this paper have relatively few cases for the number of variables, and we have not employed subsampling.

## 3. Selection algorithms

The PNN can be used, as described in the previous section, to evaluate an input variable mask. This evaluation capability can be plugged into any algorithm that searches for binary strings. This paper compares a very simple approach – bitwise gradient descent from a random starting point - with three popular approaches to feature selection: forward stepwise, backward stepwise, and the genetic algorithm.

In forward stepwise selection, a feature subset is iteratively built up [Jain and Zongker, 1997]. On the first iteration, $N$ models are tested, each of which uses a single input variable (corresponding to masking strings consisting of all zeros with a one in a single position). The mask with the lowest error is selected, indicating that the variable that gives the best performance on its own should be selected first. On each subsequent iteration, each of the unused variables is added to the model in turn, and the variable that most improves the model is selected. The algorithm terminates when adding an extra variable results in no improvement in performance (it is also possible to consider versions where the algorithm terminates if the improvement falls below some threshold, indicating an acceptable complexity/performance trade-off).

In backward stepwise selection, the algorithm starts by building a model that includes all available input variables. On each iteration, the algorithm locates the variable that, if removed, most improves the performance (or causes least deterioration). The algorithm terminates when removing a variable results in no deterioration in performance (it is also possible to consider versions where the algorithm terminates if the deterioration rises above some threshold).

A problem with forward selection is that it may fail to include variables that are interdependent, as it adds variables one at a time. However, it may locate small, effective, subsets quite rapidly, as the early evaluations, involving relatively few variables, are fast. In contrast, in backwards selection interdependencies are well-handled, but early evaluations are relatively expensive. In either case, the maximum number of possible evaluations is $V(V-1)/2$, which is potentially quite substantial.

The genetic algorithm [Goldberg, 1989] is a well-known approach for selecting binary strings, and a number of authors have suggested its use for feature selection [Yang and Honavar, 1998; Raymer *et.al.*, 1996]. An important aspect of the genetic algorithm is that it is explicitly designed to exploit epistasis (that is, interdependencies between bits in the string), and thus should be well-suited for this problem domain. However, Genetic Algorithms typically require a large number of evaluations to reach a minimum (a population of 100 strings, evaluated over 100 generations, for a total 10,000 evaluations is commonplace). This implies that the Genetic Algorithm is only likely to require less evaluations than forward or backward stepwise algorithms if the number of variables is very large (100 or more). We also note that 10,000 evaluations is sufficient to exhaustively evaluate all possible combinations of up to 13 variables. However, the genetic algorithm might achieve better results than forward or backward selection for feature sets of between 14 and 20 variables.

This paper introduces a simple and effective approach for feature selection called bitwise gradient descent. The algorithm starts with a randomly initialized string. It then "flips" each bit in the string in turn, retaining the changed bit only if the change causes a reduction in error. The total number of evaluations is only $V$ - substantially less than the other algorithms described above. If the variables are not interdependent, then this approach will yield an optimum solution. Even if they are some modest interdependencies involving a couple of variables, repeated application is likely to discover these. In addition, repeated application of the algorithm gives valuable information about the importance of the individual variables, as will be described below.

## 4. Resampling

A key issue in all the feature selection algorithms is the division of the available data into the training and test subsets. Some variables may be of great individual importance, and will be selected by any of the algorithms describe above. However, if variables are of marginal importance, or are mutually redundant, then their presence in the selected subset may be strongly influenced by the division between the training and test subsets. This problem is clearly present in all the data sets used in these experiments - repeating any of the feature selection algorithms with different training/test subset selections invariably produces different results.

A practical solution to this problem is to repeat the feature selection process a number of times, counting the number of occasions on which each variable is selected, and to regard the frequency distribution of the variables as the output of the feature selection procedure. This is more informative and more useful than a straightforward single evaluation, the results of which are extremely suspect.

It is the combination with resampling that makes bitwise gradient descent particularly effective. The algorithm is efficient enough to be repeated a moderately large number of times, and resampling helps to avoid problems where particular interdependencies between variables are entirely missed.

## 5. Experiments

Four data sets were selected from the UCI machine learning repository [Blake *et. al.*, 1998]. These are all real-world problem domains, and all have a large number of input variables and a relatively small number of cases. The data sets were not artificially chosen to demonstrate particular issues such as interdependency between features, reflect a range of variable types, and include missing value[1]. The data sets are briefly described below:

**Anneal.** 798 cases, 37 variables (9 numeric, 29 nominal), 6 output classes, no missing values.

**Horse colic.** 368 cases, 27 variables (3 numeric, 24 nominal), 3 output classes, 30% missing values. Prediction of survival of horses with colic.

**Ionosphere.** 351 cases, 34 variables (all continuous, 2 output classes, no missing values. Distinguish radar measurements which show structure. [Sigillito *et. al.*, 1989]

**Sonar.** 208 cases, 60 variables, 2 output classes. Distinguish rocks from mines on sea bed. [Gorman and Sejnowski, 1988]

All data sets are randomly divided into training and test subsets of equal size. Nominal variables were encoded using standard binary encoding (two-state) or one-of-N encoding (3 or more state) techniques. Numeric variables were normalized into the range [0,1] using the minimax procedure.

[1] Missing values are substituted using the mean training value of numeric variables, and the training set distributions for nominal variables.

The bitwise gradient descent algorithm was repeated 20 times for each data set, with a random starting string and random division into training and test cases on each execution; the variable selection frequencies are discussed below. The forward and backward stepwise selection algorithms were each repeated 5 times (the reduced number of experiments is due to the greatly increased execution time) with random division into training and test cases on each execution.

As the genetic algorithm is itself a stochastic population-based algorithm, it was speculated that a similar frequency-based approach could be applied to the final population of the algorithm. The algorithm was repeated twice for each of the Ionosphere and Horse Colic data sets (as a consequence of the extreme execution time, only a limited number of experiments were conducted), and the results compared for consistency. There are a large number of control parameters that can be altered in a genetic algorithm. For the purposes of these experiments, the following factors were selected: a standard genetic algorithm with elitism, mutation rate average 1.0 per string, crossover one-point, rate 0.3, selection by expected value roulette method, fitness linearly normalized for constant bias (fittest member of each population 5 times fitter than least fit member). These correspond to the SUGAL settings [Hunter, 1998] *replacement uniform, replacement_condition unconditional, replacement_rate 1.0, elitism on, mutation invert, mutation_rate 1.0, mutation_rate_type per_chromosome, crossover onepoint, crossover_rate 0.3, selection integral_roulette, normalisation reverse_scale, bias 5.0*. These settings were selected "by eye" on the basis of some initial experiments.

Once completed, the individual runs of each algorithm were assembled into frequency tables, giving the percentage of runs of each algorithm that each feature was selected. These frequency tables were examined in graphical format; see figure 1. Due to space considerations in this paper, only extracts from some of the frequency distributions have been included in detail. Table 1 summarizes the overall performance, showing the average percentage disparities between the feature selection frequencies of the forward and backward stepwise algorithms, and of the forward and bitwise algorithms. If the results of the algorithms were entirely unrelated, we would expect figures of approximately 50%. The figures are substantially lower than this.

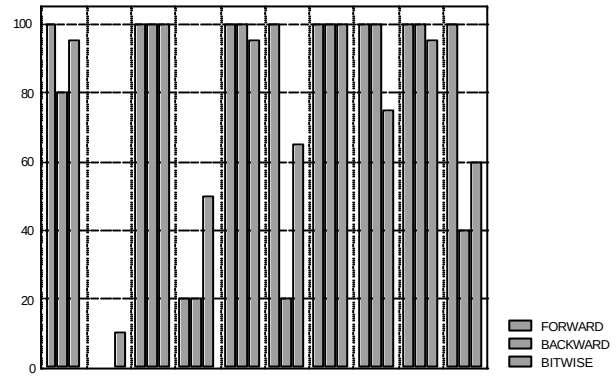
| | **Ionos** | **H. Colic** | **Sonar** | **Anneal** |
|---|---|---|---|---|
| **Fwd-Bwd** | 25.3 | 23.0 | 25.7 | 2.6 |
| **Fwd-Bit** | 22.9 | 27.6 | 37.3 | 12.1 |
| **Bwd-Bit** | 17.9 | 23.1 | 20.9 | 11.1 |

**Table 1: Average disparities in frequencies per bit**

### 5.1. Comparative performance

Figure 1 shows the frequency of feature selection by forward stepwise, backward stepwise and bitwise gradient descent for the first ten features in the Ionosphere data set.

Figure 1: Feature selection frequency, Ionosphere




This is typical of most of the data sets – the algorithms give consistent results on most of the features. The bitwise gradient descent algorithm consistently selects features that are selected by both of the other algorithms, and consistently rejects features that they would also reject. Where there is a noticeable difference between the bitwise gradient descent frequencies and the stepwise algorithms, there also tend to be differences between the stepwise algorithms. On the Anneal data set, the agreement is striking, with the algorithms selecting the same feature subset with close to 100% consistency. Here, the bitwise gradient descent algorithm does have significantly different frequencies to the other algorithms, as marked by the disparity in table 1. However, on closer analysis this disparity is due to the bitwise algorithm selecting frequencies in the range below 20% or above 80%, as opposed to consistent 0% and 100% from the other algorithms. In practice, this would not affect the feature set selected. The Horse Colic and Sonar data sets are more challenging – there the choice of variables seems to be, to a significant extent, arbitrary. This is indicated by a tendency for the selection frequencies to be less markedly extreme (often in the range 30-70%) and, unsurprisingly, this ambiguity is reflected in disparities between the frequencies recorded by the three methods; see figure 2.

It is difficult to find any evidence that the bitwise gradient descent algorithm fails to find interdependent features in any of the data sets. We would expect, in that case, to find features that are selected with close to zero frequency by forward and bitwise selection and with high frequency by backward selection. This may reflect the fact that complete interdependence is, in reality, very rarely encountered, so that the putative advantage of backward selection in being able to handle interdependence is largely theoretical.

## 5.2. Sensitivity to data set division

A far more pertinent issue is the sensitivity of all the algorithms to the division of the data set into training and test cases. For example, in the Sonar data set forward selection chooses anywhere from 48 to 56 variables, in five tests, and between 31 and 54 variables in backward selection! In the Ionosphere data, forward selection yields between 13 and 32 variables, and backward selection from 25 to 33. This implies that to apply any of the algorithms without resampling is extremely deceptive.

However, if the feature selection algorithm is run repeatedly with resampling, and a frequency table assembled, the information yielded is extremely useful. It is possible to identify definitely useful or useless variables, and to distinguish these from the ambiguous variables. If a very large number of the variables are ambiguous, then we may conclude that there is a high level of correlation between variables, and perhaps look to perform some feature extraction, such as principal component analysis, before continuing with the next stage of the analysis.

## 5.3. The genetic algorithm

The genetic algorithm is computationally extremely demanding compared with the other techniques. A naive way to use the genetic algorithm in feature selection is to run it for the requisite number of generations, then to select the best member of the final population as the result of the algorithm. The entire algorithm can then be run a number of times, just as with the other algorithms, to check for consistency.

However, this approach ignores the information available in the final population, which contains a large number of candidate solutions. One might expect that the genetic algorithm would heavily select bits for features that are definitely useful or useless, while bits that are ambiguous would be subject to contrary selective pressures and thus remain more diverse. To test this theory, the genetic algorithm was run on two of the data sets, and the frequency distributions of the bits in the final population compared between these two runs.

Figure 2: Feature selection frequency, Horse Colic

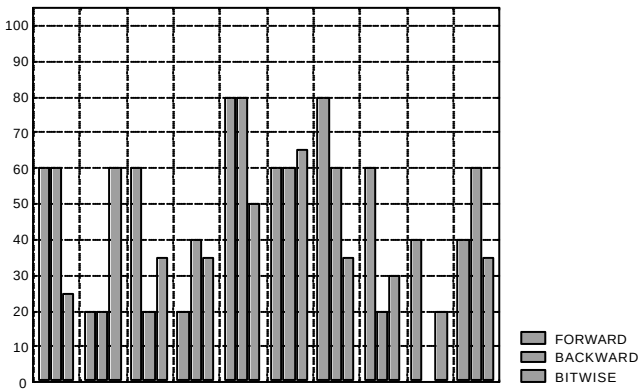


Figure 3: Genetic Algorithm, Ionosphere

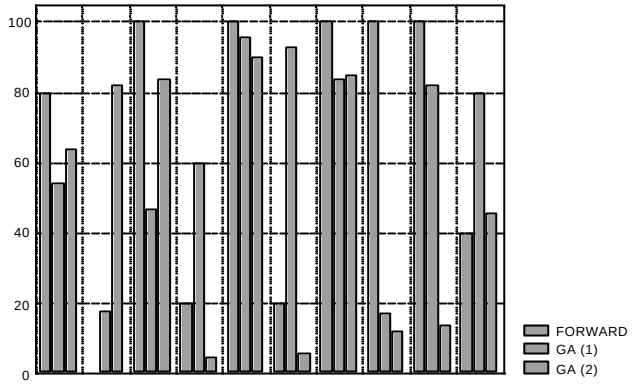



Figure 3 shows the results on the Ionosphere data set, which are clearly disappointing (the forward selection frequencies from figure 1 are repeated, for ease of comparison). It is clear that the two runs of the genetic algorithm produce quite radically different frequency distributions for each bit, which contrasts unfavorably with the consistent results produced by the other three algorithms. This may be due to "parasitical" effects, where bits with a low fitness contribution are propagated because they are located on strings which have high fitness because of more influential bit settings, or it may be a result of "genetic drift" (the tendency of unused bits to drift towards the extremes over time).

Whatever the reason, the genetic algorithm is clearly less useful as a frequency-based feature selection algorithm, although it should be emphasized that if treated in the conventional fashion (the best string being selected as the single output of the algorithm) the results are comparable with the other algorithms – although with significantly greater execution time.

## 6. Conclusion

This paper has presented a new, efficient, feature selection algorithm based on repeated bitwise gradient descent combined with Probabilistic Neural Networks. The algorithm has acceptable computational requirements (taking only a few minutes for twenty runs on all the data sets used in this paper, on a Pentium 266; this compares with up to an hour for each run of the stepwise algorithms, and up to ten hours for a single run of the genetic algorithm). Comparison with standard forward and backward stepwise algorithms shows that the new algorithm yields equally good results, at far greater speeds.

An analysis of resampling effects shows that it is critical to perform feature selection a number of times, and to base feature selection on frequencies of feature selection rather than a single run. The proposed algorithm is well-suited for this task.

## References

[Blake *et. al.*, 1998] Blake, C., Keogh, E. and Merz, C.J. *UCI repository of machine learning databases.* http://www.ics.uci.edu/~mlearn/MLRepository.html. Irvine, CA: University of California, Dept. Information and Computer Science.

[Goldberg, 1989]. Goldberg, D. E. *Genetic Algorithms*. Reading, MA: Addison Wesley, 1989.

[Jain and Zongker, 1997] Jain, A. and Zongker, D. Feature Selection: Evaluation, Application and Small Sample Performance *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19 (2), 1997.

[Sigillito *et. al.*, 1989] Sigillito, V. G., Wing, S. P., Hutton, L. V. and Baker, K. B. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10, 262—266, 1989.

[Speckt, 1990] Speckt, D.F. Probabilistic Neural Networks. *Neural Networks 3 (1)*, 109--118, 1990.

[Speckt, 1991] Speckt, D.F. A Generalized Regression Neural Network. *IEEE Transactions on Neural Networks 2 (6)*, 568—576, 1991.

[Hunter, 1998]. Hunter, A. Crossing over Genetic Algorithms: the SUGAL Generalised GA, Heuristics 4 (2), 179—192, 1998.

[Raymer et.al., 1997] Raymer, M.L., Punch, W.F., Goodman, E.D., Sanschagrin, P.C. and Kuhn, L.A. Proc. Simultaneous Feature Extraction and Selection Using a Masking Genetic Algorithm. *7th Int. Conf. on Genetic Algorithms*, 561—567, Morgan Kaufmann, San Francisco, June 1997.

[Gorman and Sejnowski, 1988] Gorman, R.P. and Sejnowski, T.J. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks 1 (1)*, 75—89, 1988.

[Yang and Honavar, 1998] Yang, J. and Honavar, V. Feature Subset Selection Using a Genetic Algorithm. *IEEE Int. Systems and their Applications 13 (2)*, 44—49, 1998.