Some Probabilistic Modelling Ideas for Boolean Classification in Genetic Programming

Jorge Muruzábal[1], Carlos Cotta-Porras[2] and Amelia Fernández[2]

[1] University Rey Juan Carlos, 28933 Móstoles, Spain
[2] University of Málaga, 29071 Málaga, Spain

Abstract. We discuss the problem of boolean classification via Genetic Programming. When predictors are numeric, the standard approach proceeds by classifying according to the *sign* of the value provided by the evaluated function. We consider an alternative approach whereby the *magnitude* of such a quantity also plays a role in prediction and evaluation. Specifically, the original, unconstrained value is transformed into a probability value which is then used to elicit the classification. This idea stems from the well-known logistic regression paradigm and can be seen as an attempt to squeeze all the information in each individual function. We investigate the empirical behaviour of these variants and discuss a third evaluation measure equally based on probabilistic ideas. To put these ideas in perspective, we present comparative results obtained by alternative methods, namely recursive splitting and logistic regression.

1 Introduction

Consider the problem of boolean classification via Genetic Programming (GP). The goal is to learn the relationship between a boolean response y and a set of predictors $x_1, x_2, ..., x_n$ on the basis of a body of known classifications. Let $\mathbf{x}$ and D denote the vector containing all predictors and the $M \times (n+1)$ training data matrix respectively. The GP approach proceeds by evolving a population of individuals or functions $\phi = \phi(\mathbf{x})$ according to some evaluation or fitness measure Θ and some set of basic functions Γ. All functions ϕ ever considered by the algorithm are combinations of members of Γ, inputs x_k and random constants; their fitness is given by $\Theta(\phi, D) \in \mathbb{R}$. The well-known procedures of selection and recombination are applied as usual for a fixed number of generations. Once this simulated evolution process is over, the best function found is ready for validation and, if appropriate, deployment on new cases $\mathbf{x}$.

The nature of predictors x_k is closely related to both the function set Γ and the evaluation method Θ. If all predictors are boolean, then one can naturally use logical operators in Γ [1]: all ϕ produce boolean values and no transformation is needed in Θ. In this paper, however, we consider the case of numeric predictors $x_k \in \mathbb{R}$. Now ϕ outputs real numbers and some kind of transformation is needed. The standard approach [2] proceeds by implementing a *wrapper* that classifies each $\mathbf{x}$ in D according to the *sign* of $\phi(\mathbf{x})$; the resulting predictions and evaluation measure are called *deterministic* for reasons that will become apparent below.

This idea returns to the previous boolean case but has a number of disadvantages. For example, it does not matter whether $\phi(\mathbf{x}) = +.1$ or $\phi(\mathbf{x}) = +1,000$, the prediction $\hat{y}$ will be 1 in either case. This is fine as long as $\hat{y} = 1$ is correct, but otherwise ϕ would seem much less valuable in the latter case. Another problem is that several unrelated functions may share the same signs over (the majority of) D. This may introduce a substantial amount of confusion in the evolutionary process. Finally, this approach yields no information about the *confidence* that we should place on our predictions; we can look at proximity to the border $\phi(\mathbf{x}) = 0$, but there is no absolute scale on which our assessment could be based. Overall, it seems a bit wasteful to carry out the search in the space of functions ϕ when all the system needs is a much simpler boolean decision criterion. It would be nice if we could profit from every bit of information in ϕ.

In this paper we study an alternative approach inspired by standard logistic regression modelling [3]. A distinctive feature of this approach is that the magnitude of $\phi(\mathbf{x})$ also plays a role. To define a new evaluation measure, the idea is to use first an appropriate function G to produce an estimate $\hat{\pi}(\mathbf{x}) = G \circ \phi(\mathbf{x})$ of the target conditional probability $\pi(\mathbf{x}) = P(Y = 1/X = \mathbf{x})$, then introduce a random mechanism to exploit the *probabilistic* information in $\hat{\pi}(\mathbf{x})$. Here X and Y denote the obvious random variables in the underlying statistical model. The random decision $\hat{y}$ is simply the result of a coin flip with probability of heads $\hat{\pi}(\mathbf{x})$. From this point on, the new evaluation measure works exactly as before. Note that constraints on G make only a few functions suitable for the present purpose. To illustrate their desired behaviour: if $\phi(\mathbf{x}) = +.1$, $\hat{\pi}(\mathbf{x})$ is slightly above .5 and will often yield $\hat{y} = 1$ *as well as* $\hat{y} = 0$; conversely, if $\phi(\mathbf{x}) = +1,000$, then $\hat{\pi}(\mathbf{x}) \approx 1$ and $\hat{y} = 1$ nearly always. This seems to be more in line with our intuition in this context.

We talk of *stochastic* predictions and evaluation in this case; let Θ_d and Θ_s respectively denote the deterministic and the stochastic evaluation measures. We provide some evidence below suggesting that Θ_s is better than Θ_d for some purposes and that the situation may be reversed for others. Specifically, Θ_s is better than Θ_d in the sense of providing more accurate results with "well-behaved" (or deterministic) training data; Θ_d seems more parsimonious when there is noise in the data.

As regards interpretation, $\hat{\pi}(\mathbf{x}) \in (0, 1)$ is an extremely simple and informative summary for future $\mathbf{x}$. In particular, we shall explore below another way of exploiting the *whole set* $\{\hat{\pi}(\mathbf{x}), \mathbf{x} \in D\}$ for evaluation purposes. This will be seen as an attempt to quantify the overall *fit* of the selected ϕ in a well-defined sense.

The organization of the paper is as follows. Section 2 contains a brief description of the logistic regression paradigm. Section 3 presents our experimental setup and main empirical results. Section 4 links up with some related work. Section 5 closes with some discussion and an outline of future research.

2 A review of the logistic regression paradigm

In logistic regression (LR) modelling, the boolean random variable Y is assumed to depend on the available predictors x_1, x_2, ..., x_n in the following way. First, the contribution of these predictors is totally captured by a linear combination

$$f(\mathbf{x}, \boldsymbol{\omega}) = \omega_0 + \omega_1 x_1 + ... + \omega_m x_m.$$

Here we may have all or some of the original variables together with some other (simple) functions thereof (eg. squares). An important point is that the user needs to specify exactly what m variables come into play. Now, it is assumed that, for some *link* function G and some $\boldsymbol{\omega} \in \mathbb{R}^{m+1}$, Y follows, conditionally on $\mathbf{x}$, a Bernoulli distribution with parameter $G \circ f(\mathbf{x}, \boldsymbol{\omega})$. A link function is a one-to-one mapping from $\mathbb{R}$ onto the unit interval $(0, 1)$. While several choices for G are available, the most widely used is the inverse-logistic or sigmoid transformation

$$G(y) = 1/\left[1 + exp\left(-y\right)\right].$$

This particular choice will also be assumed throughout this paper. We note in passing that this model can be seen as a particular case (namely, the no-hidden-layer case) of the standard neural network model for classification tasks [5].

Jordan [6] has put together a number of useful results concerning this function. Most relevant for our purposes here is the fact that, if we model our classification problem in the usual way[3], then the posterior probability of either class can *always* be expressed as $G \circ \xi(x)$ for some function ξ. It follows that $G(y) = 1/\left[1 + exp\left(-y\right)\right]$ is the most natural choice for our transformation (which, while not bayesian, aims to provide estimated "posterior" probabilities).

Returning now to the LR paradigm, the key assumption is that the target function

$$\pi(\mathbf{x}) = P(Y = 1/X = \mathbf{x})$$

can be approximated by (some member of) the parametric form

$$h(\mathbf{x}, \boldsymbol{\omega}) = 1/\left[1 + exp\left(-f(\mathbf{x}, \boldsymbol{\omega})\right)\right].$$

It is then possible to formulate the likelihood function

$$L(\boldsymbol{\omega}/D) = \prod_{j=1}^{M} h(\mathbf{x}_j, \boldsymbol{\omega})^{y_j} \left[1 - h(\mathbf{x}_j, \boldsymbol{\omega})\right]^{1-y_j}$$

and estimate $\boldsymbol{\omega}$ by the method of maximum-likelihood. The LR algorithm implementing this method is well-studied and widely available [4]. While the resulting estimate $\hat{\boldsymbol{\omega}}$ is not guaranteed to maximize L, it typically provides sensible answers.

[3] That is, incorporating a priori probabilities for the two classes and conditional densities for the input vectors $\mathbf{x}$

The above problem may also be viewed alternatively by noting that $f(\mathbf{x}, \hat{\boldsymbol{\omega}})$ aims to approximate the *logistic* transformation of $\pi(\mathbf{x})$, that is,

$$\lambda(\mathbf{x}) = logit(\pi) = log[\pi/(1-\pi)] \in \mathbb{R}.$$

This *logit* function λ will also be the target for functions ϕ in our GP context. Indeed, in our simulations we specify λ and generate the response y associated to $\mathbf{x}$ on the basis of $G \circ \lambda(\mathbf{x})$.

Fitted probabilities for individual $\mathbf{x}_j \in D$ are given by $\hat{\pi}_j = h(\mathbf{x}_j, \hat{\boldsymbol{\omega}})$. In ordinary LR, however, each prediction $\hat{y}_j \in \{0, 1\}$ simply reflects the sign of the corresponding $f(\mathbf{x}_j, \hat{\boldsymbol{\omega}})$. Thus, following the terminology introduced earlier, the LR paradigm typically behaves deterministically (just like the standard GP approach).

As regards the practical aspects of the LR paradigm, it is clear that the analyst will rarely carry out a single fit (or estimation of $\boldsymbol{\omega}$). Upon examination of the first fit, the analyst may remove some variables and incorporate others. More often than not, however, we let the computing package carry out an *automatic* exploration of the space of all linear combinations of a specified set of variables along with their two-at-a-time products and squares [7]. For example, in S-PLUS this exploration is carried out via the `step` command. The algorithm then selects the "best-fitting" model and returns the associated vector of estimated coefficients $\hat{\boldsymbol{\omega}}$ together with some indication of their significance given the remaining variables. This selection process is based on a scalar quantity that combines the quality of fit itself (or *deviance*, see [4]) with the number of terms (or ω's) in the model. However, this measure is less readily interpretable. Instead, the set of fitted probabilities $\hat{\pi}_j$ can be used to validate the model as shown next.

The Chapman data refer to past occurrence of cardio-vascular disease (CVD) amongst 200 males examined by certain hospital. There are six predictors including weight, height, age, blood pressure, etc. The percentage of positive response is only 13%; see [3] for further details. The preferred model is given in Table 1; thus, the key equation is given by

$$f(\mathbf{x}, \hat{\boldsymbol{\omega}}) = -9.255 + 0.053\ AGE + 0.018\ WEIGHT + 0.007\ CHOLESTEROL.$$

The interpretation is immediate: the older and heavier the patient, the higher the chance of having had CVD (and similarly for the cholesterol level, although the latter's coefficient is quite small). While the cholesterol variable is not overwhelmingly significant, it is seen to contribute to a sensible model as follows.

variable	**coefficient**	**t-value**
Intercept	−9.255	−4.49
Age	0.053	2.55
Weight	0.018	4.91
Cholesterol	0.007	0.79

Table 1. Selected model for the Chapman data, see [3].

Note first that the confusion matrix arising from this fit (given in Table 2) might lead us to think that the model is not good: $\hat{y}_j = 1$ in only two occasions, 24 positive cases go unpredicted! The following argument shows that this is not the case: the postulated model provides an excellent fit.

	zero	one
zero	174	0
one	24	2

Table 2. Confusion matrix for the training sample under the model shown in Table 1. Columns are predicted values $\hat{y}_j$; rows are observed values y_j.

Christensen [3] first groups individuals in categories according to their CVD probabilities $\hat{\pi}_j$, see Table 3. For example, the first category includes all individuals with $\hat{\pi}_j \in [0, 0.1)$; we find 99 cases here[4]. Now, if the midpoint .05 is selected as representative probability of this group, then we would naturally expect $99 \times 0.05 = 4.95$ positive cases. This figure is to be contrasted with the true number of CVD occurrences in the group: out of the 99 individuals with predicted probability below .1, we find 5 positive cases. Hence, the fit is very good in this first group. Indeed, the fit is rather good through all 6 groups.

Interval	[.0,.1)	[.1,.2)	[.2,.3)	[.3,.4)	[.4,.5)	[.5,.6)
E_i	4.95	9	5.5	3.5	3.15	1.1
O_i	5	10	2	5	2	2
Number of cases	99	60	22	10	7	2

Table 3. Groupwise expected (E_i) and true (O_i) number of CVD occurrences for the Chapman data based on the model shown in Table 1.

The explanation of this phenomenon is as follows. Since the LR method is based on a simple *linear* cut, the model can not tell whom exactly had CVD in each subinterval or risk group. However, the observed behaviour of each group as a whole is quite in line with the model's probabilistic assessment. This validation example is particularly interesting in that it refers to a case where the distribution of 0's and 1's is rather *skewed*. Skewed data sets (containing only a few 0's or 1's) are difficult for the conventional GP approach since any constant with the right (dominant) sign will have high fitness. Letting the evaluation function measure the overall fit in this way should be helpful to get rid of such free-riders.

We will return to Christensen's idea at the end of Section 3. A basic observation is that the previous assessment of fit depends in no way on the LR paradigm and can therefore be "exported" as soon as the analog of the $\hat{\pi}_j$'s are provided. It can also be used on test data as a validation tool.

[4] Of course, the fit can also be measured with a higher or lower number of subintervals.

3 Empirical work

In this Section we present the bulk of our empirical results. We first describe the basic experimental setting, then proceed to discuss some specific problems.

3.1 Experimental setup

We have extended the standard symbolic regression GP paradigm implemented in the `lil-gp` system [8]. Our modifications allow for more than one predictor at a time ($n > 1$) and incorporate all details required to test the ideas described above. The modified system currently runs on a LINUX-based PC; source code is available from the authors.

Due to the exploratory nature of this research, we choose to keep the situation as controlled as possible: all experiments reported below are based on simulated data. In our runs, we always use the same system parameters, see Table 4. Since the focus is on comparative results and not on optimal choices for these parameters, we have simply adopted `lil-gp`'s default options. In particular, we always use a single population and maintain the same reproductive plan throughout each run. We execute ten independent runs under each configuration; a summary of the runs is often given in the standard box-plot form. Either evaluation measure Θ_s or Θ_d is based on the number of correct predictions on the training sample as standardized fitness [2]:

$$\Theta(\phi, D) = M - \sum_{j=1}^{M} |y_j - \hat{y}_j| \,,$$

where $\hat{y}_j = \hat{y}_j(\phi, \mathbf{x}_j)$ may be computed either deterministically or stochastically as explained earlier.

System Parameter	**Value**
Maximum number of generations	200
Population size	500
Training sample size	500
Test sample size	500
Initial population generation	half-and-half
Maximum tree depth	5-10
Ephimeral random constant	Yes
Crossover/Reproduction	80/20
Mutation	None

Table 4. Execution parameters. See `lil-gp`'s documentation [8] for details.

Maximum tree depth is usually 10 but we sometimes switch to 5 in order to reduce the complexity of the output functions. We experiment with two function sets: Γ contains the arithmetic operators alone, $\Gamma = \{+, -, *, /\}$, where / stands for protected division as usual. Function set $\mathit{\Gamma}^*$ contains also the protected logarithm $rlog(x)$, the exponential function $exp(x)$, and the trigonometric functions $sin(x)$ and $cos(x)$.

In our simulations, we work with $n = 5$ predictors; out of these, 2 or at most 3 are relevant, the rest contribute noise. To create our training sets D, we first draw, for each predictor, independent, uniformly distributed variates in $(-5, 5)$. Then we specify the "true" logit function $\lambda(\mathbf{x})$ and generate the associated responses accordingly. Just like in the case of predictions $\hat{y}$, these responses can be generated either deterministically (the "no noise" situation) or stochastically (the "noise" situation). The latter formulates a more realistic and typically harder problem. Results from both options are presented below.

As usual, for each problem we create a second sample to test generalization ability. This test sample itself can also be created either deterministically or stochastically; deterministic test samples are used except where otherwise noted. Lastly, test performance in turn can be determined in either of these two ways. Because the alternative techniques that we study make their predictions deterministically, so do we with our GP functions. This ensures the fairness of our comparisons throughout.

The outline of the rest of this Section is as follows. We first illustrate the "confusion of signs" problem mentioned in the introduction. Next, always under $\mathit{\Theta}_s$ and $\mathit{\Theta}_d$, we analyze two problems well-suited for other techniques. We also approach these problems with their "natural" methods and discuss comparative performance. Specifically, we try the logistic regression and recursive splitting algorithms implemented in the S-PLUS system (V4.5), see [7]; all run parameters are fixed to their default values in either case. We only use deterministic training data so far. In Section 3.5 we introduce a third evaluation measure based on overall fit. Finally, we explore performance with noisy training data.

3.2 A basic confounding problem

Consider the problem of learning the logit function $\lambda_1(\mathbf{x}) = x_1 - 2x_2 + 5x_1x_2$. When both the generation of data D and the evaluation of functions ϕ are done deterministically, a basic confounding problem arises during learning[5]. As mentioned, this problem is due to the emergence of alternative functions which partially or totally match the signs of λ_1 over D. The result is that learning is impaired and we may end up with a totally mistaken function. For example, the system converged once to

```
 (/ (rlog (exp x2))
    (* x1 0.86686)).
```

In another run, we found a function containing 10 instances of the "successful" string `(/ x1 x2)`. Likewise, some variations of the sign-matching function x_1x_2 were discovered early and the system got stuck; this was the case of

[5] Actually, training data for this problem were generated stochastically (as shown in Fig. 1a). However, since the slopes near the axes are so sharp here, these data do not differ much from deterministically generated data.

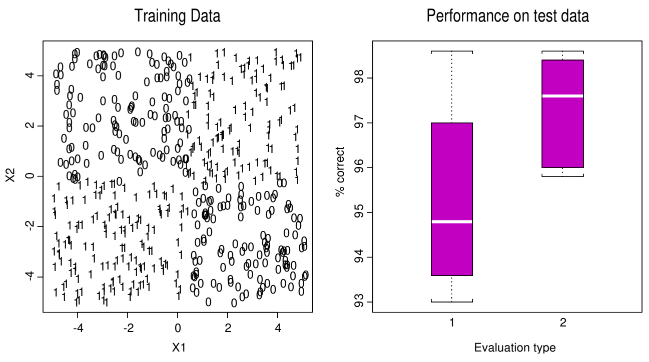

Fig. 1. Learning λ_1 with function set Γ^* and maximum depth set to 10. (a) Training data in the relevant $x_1 \times x_2$ space. (b) Boxplots (each based on ten runs) of performance results under the two evaluation measures Θ_d and Θ_s. In this and the following boxplots, standard GP (Θ_d) is identified as evaluation type 1. Recall that the box itself runs from the lower to the upper quartile, thus capturing central or typical behaviour. The white line represents the median. The "whiskers" stretch out to cover less typical values, whereas atypical values are shown individually (see eg. Fig. 2). Note also that the vertical scale changes from boxplot to boxplot.

```
(/ (* x2 x1)

   (exp x4)),
```

found at initialization and never improved upon! Naturally, this behaviour under Θ_d does not persist under Θ_s. In any case, performance on test data (see Fig. 1b) shows that Θ_s never declines below 95.8% and is thus more likely to prevent a poor fit.

3.3 The logistic regression problem

Consider now the logit function $\lambda_2(\mathbf{x}) = x_1 - 2x_2 + x_3 + 2x_1x_2 - 3x_2x_3$. This reflects exactly the type of structure that logistic regression (LR) searches for; it should therefore provide a nice basis for evaluation of the GP approach.

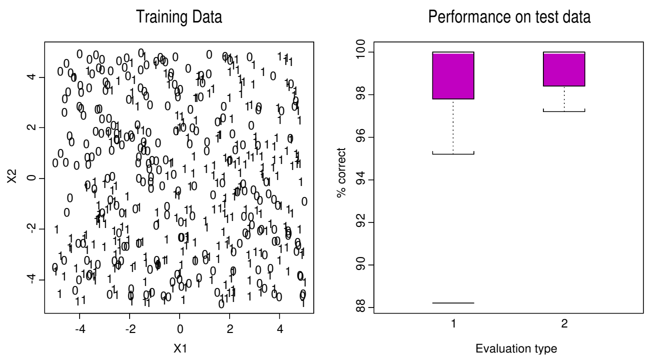

Fig. 2. Learning λ_2 with function set Γ and maximum depth set to to 5. (a) Partial view of the training data. (b) Boxplots of performance results (see Fig. 1). Recall that Θ_d and Θ_s are identified respectively with evaluation type 1 and 2.

variable	**coefficient**	**t-value**
Intercept	1.18	1.56
x_0	-0.44	-1.63
x_1	2.60	3.01
x_2	-4.36	-3.17
x_3	2.12	3.21
x_4	0.38	1.49
$x_2 \times x_3$	-7.32	-3.33
$x_1 \times x_2$	5.24	3.28
$x_2 \times x_4$	-0.36	-2.57
$x_0 \times x_2$	0.30	2.39
$x_0 \times x_1$	0.12	1.52

Table 5. Logistic regression fit for λ_2 (similar to Table 1).

Fig. 2b shows the results on the test sample. It appears that this problem is easy since the algorithm often finds the correct function (the median success rate is 100% in both cases). However, the lower tail of the distribution is again better for Θ_s.

As regards performance by the LR approach, the best fit (produced by the `step` command as discussed earlier) is summarized in Table 5. Note that we do not recover the desired function exactly. In particular, noise variables x_0 and x_4 show up several times. While the overall significance of such terms is not large in general, the -2.57 t-value of $x_2 \times x_4$ stands out. This is in contrast to the perfect match often achieved by the GP approach. On the other hand, all key terms are found significant, and indeed their coefficients are about 2.4 times their target values. Accordingly, performance on test data is 97.2%. Hence, as expected, the

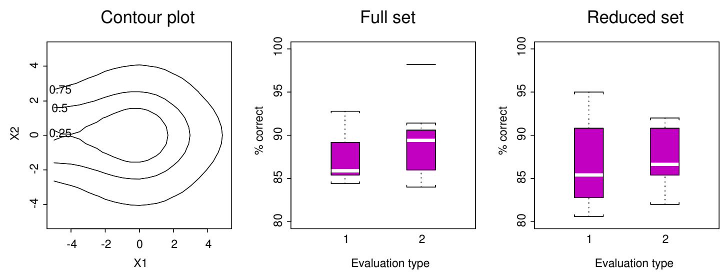

Fig. 3. Learning λ_3. (a) Contour plot of $G \circ \lambda_3$. (b) Performance results for the full function set Γ^* and maximum depth set to 10. (c) Performance results for Γ and maximum depth set to 5.

model fits the data well, yet the GP approach under Θ_s does better.

3.4 The recursive splitting problem

Let us now switch to the problem of learning

$$\lambda_3(\mathbf{x}) = 1.5 - \exp\{-\frac{x_1 + x_2^2}{5}\} + \log(\frac{1 + x_1^2 + x_2^2}{25}).$$

This function illustrates one of the worst-case scenarios for the LR method: while the basic structure is still a sum, summands comprise complicated functions of the input variables. However, the problem is relatively easy for the recursive splitting method. Indeed, the contour plot in Fig. 3a shows that parallel decision boundaries should not be too hard to find. But let us look first at performance by the GP algorithm.

To begin with, consider the case of Γ^* so that the exact target function can be found in principle. The results in Fig. 3b show that Θ_s yields again better results. Not only the best absolute result is achieved in this case, but the median percentage is about 4 points higher. Now we repeat the previous experiment with Γ instead of Γ^*; the system is thus somewhat handicapped but it is still interesting to compare performance. As shown in Fig. 3c, Θ_s leads once again to the best median, although the best overall individual was now evolved under Θ_d.

Consider now performance by the recursive splitting method. We find a relatively simple decision tree with 13 nodes achieving a 96.6% success rate on the test sample. Note that GP performs slightly worse, but it does provide a function! Obviously, things would be much harder for this method were the target region tilted towards, say, the main diagonal. We will examine below performance in a related problem of interest, namely, learning in the presence of noise.

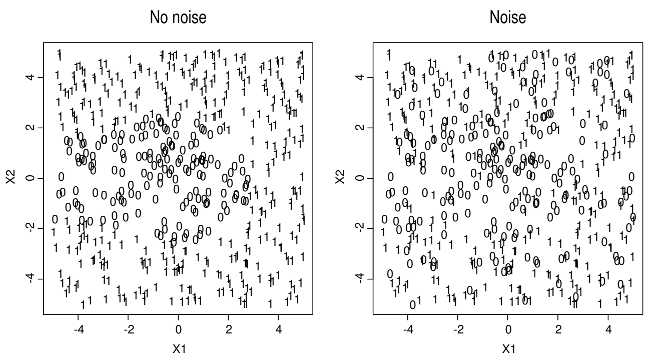

Fig. 4. Training data from λ_3.(a) Deterministic generation follows Fig. 3a closely. (b) Stochastic generation eliminates rigid borders (0's and 1's may occur anywhere).

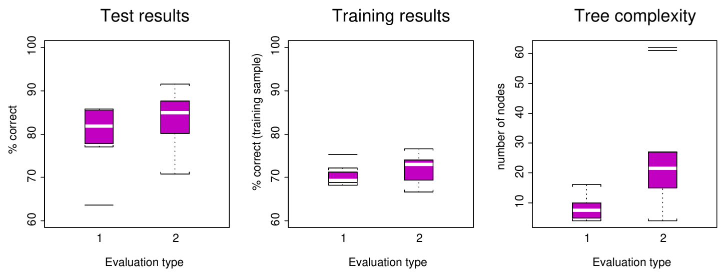

Fig. 5. Learning λ_3 with stochastically generated data (see Fig. 4b). (a) Performance on the test sample. (b) (Final) performance on the training sample. (c) Size of the best trees.

3.5 Measuring the overall fit

In the real world it is unlikely that we find perfectly separated classes as in Fig. 4a. The situation in Fig. 4b should be far more common. Recall that, under stochastic generation, we first compute the probabilities $\hat{\pi}_j$ and then obtain each y_j as a Bernoulli realization with probability $\hat{\pi}_j$. There are at least two issues of interest related to this type of training data. First, and most important, how well do the GP algorithm and its competitors perform here? Some results are reported in the next Section.

Second, we can naturally consider yet another evaluation measure, namely that hinted at in Table 3 above. Specifically, define the overall fit of a given

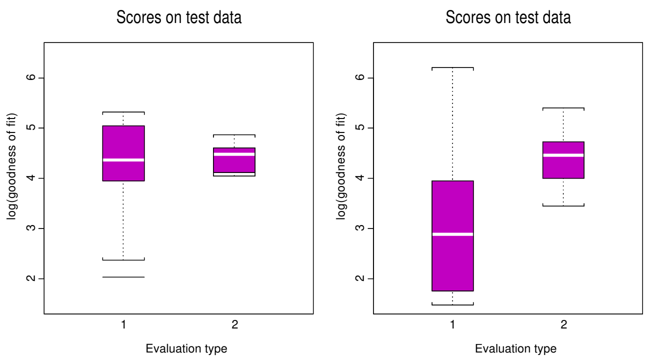

Fig. 6. Θ_f log-scores of individuals evolved earlier for logit functions λ_1 and λ_3 respectively. These numbers are based on a second stochastic test sample. In contrast to previous plots, the lower these values, the better.

individual ϕ as [3]

$$\Theta_f(\phi) = \sum_{i=1}^{10} \frac{[O_i(\phi) - E_i(\phi)]^2}{E_i(\phi)}.$$

Note that this measure is formally reminiscent of the standard χ^2 test for goodness of fit. Nevertheless, the idea is clearly different here since not only O_i but also E_i depend on ϕ. Note also that Θ_f does not make sense when predictions are deterministic. On the other hand, it is quite interesting for real data, particularly skewed data (see Section 2). However, unlike Θ_s, Θ_f does introduce an important computational overhead that needs careful assessment. Hence no GP runs based on it are reported here. To get some intuition for Θ_f, we have evaluated some functions evolved under Θ_s and Θ_d; these will be discussed in the next Section.

3.6 Learning in the presence of noise

We now return to $\lambda_3(\mathbf{x})$ and consider learning based on a *stochastic* sample, see Fig. 5. Θ_s provides once again slightly better results, although we now show that it requires twice as many nodes on average (note also that relatively complex functions emerge occasionally). To put these results in perspective, consider performance of the recursive splitting method on the same training set. We find a much bigger tree with 67 terminal nodes achieving a success rate of 86.4% on the training set but only 65.6% on the test sample. Thus, the GP algorithm does better in this case as well, particularly under Θ_s. Note also that, unlike the recursive splitting method, the GP algorithm generalizes correctly.

Finally, Fig. 6 shows (log) scores achieved under Θ_f by the 40 functions depicted in Figs. 1 and 5. Looking first at Fig. 6a, it appears that Θ_s and Θ_d are comparable as far as the median Θ_f is concerned. As usual, under Θ_s we get a tighter distribution avoiding very poor fits. On the other hand, the best fit is achieved by Θ_d and corresponds to the very simple $rlog(x_2) - .665$.

If we switch to Fig. 6b, the situation is markedly different: Θ_s can not compete now with Θ_d as far as the overall fit is concerned. Curiously enough, the absolute best fit is achieved by the nearly identical

```
(+ (rlog (cos (cos 0.59310)))
   (rlog x2)).
```

The fit, shown in Table 6, is indeed uniformly good. Overall, however, it would seem unlikely that the contribution of x_1 in either λ_1 or λ_3 could be found using Θ_f. On the other hand, Θ_f's apparent built-in bias for parsimony is worth investigating. We also note that the correlation between these Θ_f scores and the original performance scores (shown in Figs. 1 and 5) is low. Thus, the underlying search is quite different in each case.

Interval	[.0,.1)	[.1,.2)	[.2,.3)	[.3,.4)	[.4,.5)	[.5,.6)	[.6,.7)	[.7,.8)
E_i	0	2	7	16	18	39	84	121
O_i	1	3	9	14	18	30	90	127
Number of cases	9	16	29	46	39	71	129	161

Table 6. Groupwise expected (E_i) and true (O_i) number of 1's (analogous to Table 3) for the best fit under Θ_f in problem λ_3.

4 Some related research

The present approach concentrates on the quantitative nature of predictors and outputs real-valued functions. A completely different idea is to dismiss part of this detail and conduct the search over the space of boolean functions. The aim, of course, is to simplify interpretation. For example, Eggermont, Eiben and van Hemert [9] study a scheme based on "booleanization" of numeric predictors and inclusion of logical and comparison operators only. Booleanization is achieved by special functions $A_<(x_k, r)$ and $A_>(x_k, r)$, where r is some threshold, with the obvious interpretation $A_<(x_3, 32.5) = TRUE \iff x_3 < 32.5$. They use specialized mutation operators to modify the arguments of these functions. The overall results (based on several real data sets taken from [12]) are not quite satisfactory though: "Giving up the flexibility of numeric operators for the sake of transparency ... comes at costs of performance". A similar idea is advocated (although not tested) in [10].

Complexity in the resulting functions is indeed a recurrent area of research in GP. Cavaretta and Chellapilla [11] retain the set of mathematical functions but modify the usual fitness measure (Θ_d in our notation, *no-complexity-bias* in their terminology) by adding a term that penalizes functions with a large number of nodes. This is certainly not the first time that this idea is put to work in the GP context. Their results (based on a single test problem from the STATLOG suite [12]) indicate that "the models generated from the no-complexity-bias algorithm were very accurate, beating the best of the STATLOG algorithms, and were statistically significantly more accurate than the low-complexity-bias algorithm". These results are in line with the evidence provided above: under Θ_s, the returned functions are slightly more complex but perform more accurately on (deterministic) test samples.

Interestingly, these authors do not apply Θ_d over the *entire* training sample. As a basis for evaluation, they rather consider several subsamples from the training set, and they let these subsamples change with each generation. Although their main motivation is to reduce the chance of overfitting, this stochastic fitness measure can also be seen as an attempt to capture the overall fit of individual functions. Unfortunately, however, it is not entirely clear how this idea was implemented (see also below).

A different evaluation measure was tested out in [1]. If we label the entries of the confusion table as follows (see Table 2)

	zero	one
zero	*tn*	*fp*
one	*fn*	*tp*

Table 7. Generic confusion matrix. Columns are predicted values, rows are true values.

then their proposal is $\frac{tp}{tp+fn}\frac{tn}{tn+fp}$ (instead of the traditional $\frac{tn+tp}{tn+tp+fn+fp}$). This involves familiar quantities from the medical domain. The authors report that some comprehensible rules of interest were discovered by this measure.

Resampling underlies also the work of Iba [13]. This author believes that the new strategies called *boosting* and *bagging* may help to "control the bloating effect by means of the resampling techniques". In these variants, the population is divided into subpopulations and a different training sample is used in each subpopulation. Boosting places the emphasis on adaptive subsampling of *hard* cases, see also the cited [9]. This approach is closely related to various *co-evolutionary* schemes inspired by the seminal work of Hillis [14]. Bagging proceeds by simply counting votes from the independently evolved solutions. These strategies have proved useful in several machine learning tasks and show great potential in GP (at the expense of some added computational effort).

5 Summary and concluding remarks

We have presented a number of ideas and preliminary empirical results concerning the introduction of simple probabilistic machinery in the GP approach to boolean classification. The main focus has been placed on the computation of the predictive probabilities $\hat{\pi}$ and the associated stochastic evaluation of individuals Θ_s. The main ideas are central to the well-known method of logistic regression (LR). The added computational cost with respect to the more usual Θ_d is negligible and there is some indication that better results are obtained.

Endowed with this new evaluation method, the GP approach has been shown to provide competitive results with respect to standard methods. Further, the GP approach sometimes exhibits pleasant features that the alternative methods do not (eg., it finds the correct function in the LR problem), and is much better in the noisy recursive splitting problem.

We have also provided some insights into an alternative evaluation measure (Θ_f) based on the idea of overall fit. We believe that this is a promising alternative to tackle highly skewed data sets and we intend to investigate its performance in this direction. We have discussed evaluations of some of the functions found under Θ_d and Θ_s. As it turns out, the best-fitting functions under Θ_f are crude but useful versions of the target functions. This is positive in that a tendency towards parsimony may be in effect. On the other hand, it remains to be seen that fine detail is within the reach of the GP algorithm trained with Θ_f. In any case, further research is needed to ascertain its merit, to quantify the effect of the number of subintervals, etc.

Extensive experimentation with real data, particularly from the STATLOG project, should also be conducted to validate our findings. It would be equally interesting to compare the previous results to other established methods such as neural nets.

Finally, we have reviewed a number of recent developments in the GP literature. Our approach is based on maintaining quantitative detail throughout; exclusive consideration of boolean functions just precludes computation of any $\hat{\pi}$'s whatsoever. With this exception, we find it likely that portions of the present framework may cross-fertilize with some previously advocated ideas. Overall, the GP paradigm seems a worthwhile alternative for the problem of boolean classification. Extensions to deal with the general ($k-$leg) classification problem deserve attention as well.

Acknowledgement. We are grateful to Bill Punch for assistance with `lil-gp`. The first author is supported by grants HID98-0379-C02-01 and TIC98-0272-C02-01 from the spanish CICYT agency.

References

1. C. C. Bojarczuk, H. S. Lopes and A. A. Freitas (1999). Discovering Comprehensible Classification Rules Using Genetic Programming: A Case Study in a Medical Domain. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO–99), Vol. 2.
2. J. R. Koza (1992). *Genetic Programming.* MIT Press.
3. R. Christensen (1997). *Log-Linear Models and Logistic Regression* (2^{nd} Ed.). Springer.
4. McCullagh, P. and Nelder, J. A. (1983). *Generalized Linear Models.* Chapman & Hall.
5. C. M. Bishop (1995). *Neural Networks for Pattern Recognition.* Oxford University Press.

6. M. I. Jordan (1995). Why the Logistic Function? A Tutorial Discussion on Probabilities and Neural Networks. Computational Cognitive Science Technical Report 9503. M. I. T.
7. W. N. Venables and B. D. Ripley (1997). *Modern Applied Statistics with S-PLUS* (2^{nd} Ed.). Springer.
8. See `http://GARAGe.cps.msu.edu/software/lil-gp/lilgp-index.html`.
9. J. Eggermont, A. E. Eiben and J. I. van Hemert (1999). A Comparison of Genetic Programming Variants for Data Classification. Proceedings of the Third Symposium on Intelligent Data Analysis (IDA–99).
10. A. A. Freitas (1997). A Genetic Programming Framework for Two Data Mining Tasks: Classification and Generalized Rule Induction. Proceedings of the Second Genetic Programming Conference (GP–97).
11. M. J. Cavaretta and K. Chellapilla (1999). Data Mining Using Genetic Programming: the Implications of Parsimony on Generalization Error. Proceedings of the 1999 Conference on Evolutionary Computation (CEC–99).
12. D. Michie, D. J. Spiegelhalter and C. C. Taylor (1994). *Machine Learning, Neural and Statistical Classification.* Ellis Horwood.
13. H. Iba (1999). Bagging, Boosting and Bloating in Genetic Programming. Proceedings of GECCO–99, Vol. 2.
14. W. D. Hillis (1991). Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure. In *Artificial Life II, SFI Studies in the Science of Complexity*, C. G. Langton, C. Taylor, J. D. Farmer and S. Rasmussen, Eds., Addison-Wesley.

This article was processed using the LaTeX macro package with LLNCS style