Genetic Programming and Evolvable Machines, 6:3, 265–281, 2005

DOI: 10.1007/s10710-005-2988-7

Genetic Programming with a Genetic Algorithm for Feature Construction and Selection

MATTHEW G. SMITH Matt@matt-smith.me.uk
LARRY BULL Larry.Bull@uwe.ac.uk
Faculty of Computing, Engineering & Mathematical Sciences,University of the West of England, Bristol, BS16 1QY, UK

Submitted October 6, 2003; Revised December 22, 2004

Published online: 12 August 2005

Abstract. The use of machine learning techniques to automatically analyse data for information is becoming increasingly widespread. In this paper we primarily examine the use of Genetic Programming and a Genetic Algorithm to pre-process data before it is classified using the C4.5 decision tree learning algorithm. Genetic Programming is used to construct new features from those available in the data, a potentially significant process for data mining since it gives consideration to hidden relationships between features. A Genetic Algorithm is used to determine which such features are the most predictive. Using ten well-known datasets we show that our approach, in comparison to C4.5 alone, provides marked improvement in a number of cases. We then examine its use with other well-known machine learning techniques.

Keywords: genetic programming, genetic algorithm, feature construction, feature selection, classification, machine learning

1. Introduction

Classification is one of the major tasks in machine learning [17], involving the prediction of class value based on information about some other attributes. The process is a form of inductive learning whereby a set of pre-classified training examples are presented to an algorithm which must then generalise from the training set to correctly categorise unseen examples. One of the most commonly used forms of classification technique is the decision tree learning algorithm C4.5 [19]. In this paper we examine the use of Genetic Programming (GP) [4, 14] and a Genetic Algorithm (GA) [10] to improve the performance of, initially, C4.5 through feature *construction* and feature *selection*. Feature construction is a process that aims to discover hidden relationships between features, inferring new composite features. In contrast, feature selection is a process that aims to refine the list of features used thereby removing potential sources of noise and ambiguity. We use GP individuals consisting of a number of separate trees/automatically defined functions (ADFs) [14] to construct features for C4.5. A GA is then used to select over the combined set of original and constructed features for a final hybrid C4.5 classifier system. In an effort to reduce over-fitting the train data is randomly reordered between the two stages. Results show that the system is able to outperform standard C4.5 on a number of datasets held at the UCI repository (http://www.ics.uci.edu/∼mlearn/MLRepository.html). We then show how similar benefits are obtained for the *k*-nearest neighbour algorithm [1] and Naïve Bayes [11].

Raymer et al. [20] have used ADFs for feature *extraction* in conjunction with the *k*-nearest-neighbour algorithm. Feature extraction replaces an original feature with the result from passing it through a functional mapping. In Raymer et al.'s approach, as with the algorithm presented here, for problems with n features individuals consist of n ADFs whose fitness is evaluated using an external classifier. However in Raymer et al.'s approach each ADF is based on a single feature and evolved for that feature only, with the aim of increasing the separation of pattern classes in the feature space—in our approach each ADF may be constructed from multiple features. Ahluwalia and Bull [2] extended Raymer et al.'s approach by coevolving the ADFs for each feature and adding an extra coevolving GA population of feature selectors; extraction and selection occurred simultaneously in $n + 1$ populations (this is in contrast to the single population in the approach presented here, with (initially) creation and selection in separate stages). Amit and Geman [3] have explored the joint induction of binary features and tree classifiers for shape recognition. They grow multiple decision trees where the non-terminal nodes are binary features based on the spatial relationships of image 'tags' and the terminal nodes are labelled with an estimate of the conditional distribution over the shape classes. For other (early) examples of evolutionary computation approaches to data mining see [12, 21] for GA-based feature selection approaches using *k*-nearest-neighbour, and [9] for an overview of a special edition of the journal of Machine Learning Research on Variable and Feature Selection. Vafaie and DeJong [23] have combined GP and a GA for use with C4.5. They used the GA to perform feature selection for a face recognition dataset where feature subsets were evaluated through their use by C4.5. GP individuals were then evolved which contained a variable number of ADFs to construct new features from the selected subset, again using C4.5. Our approach is very similar to Vafaie and DeJong's but initially the feature operations are reversed such that feature construction occurs before selection (and later the two stages are combined). We find that our approach performs as well or better than Vafaie and DeJong's.

More recent work using GP to construct features for use by C4.5 includes that of Otero et al. [18]. They use a population of GP trees to evolve a single new feature using information gain as the fitness measure (this is the criteria used by C4.5 to select attributes to test at each node of the decision tree). This produces a single feature that attempts to cover as many instances as possible—a feature that aims to be generally useful and which is appended to the set of original features for use by C4.5. Ekárt and Márkus [8] use GP to evolve new features that are useful at specific points in the decision tree by working interactively with C4.5. They do this by invoking a GP algorithm when constructing a new node in the decision tree—e.g., when a leaf node incorrectly classifies some instances. Information gain is again used as the fitness criterion but the GP is trained only on those instances relevant at that node of the tree.

Krawiec [15] also uses GP to construct new features for use by C4.5 but instead of using information gain as the fitness criterion uses, like the technique presented here, the "so-called wrapper approach [13] where the evaluation consists of multiple train-and-test experiments carried out [with] the same inducer that is used to create the final classifier" [15] (the train and test sets used in such cross-validation are sometimes referred to as train-train and train-test as they are drawn only from the train set). Krawiec justifies the additional computational expense involved by reporting Kohavi and Johns [13] findings that "although computationally demanding . . . [the wrapper approach] seems to out-perform other methods on most tasks." Krawiec's algorithm produces a fixed number of new features (4 in the

experiments shown) which *replace* the original set of features without any subsequent selection. Krawiec also extends the algorithm with the concept of features that are hidden from the evolutionary process to preserve them from destruction. These features (2 in the experiments shown) are selected according to the number of times they appear in the decision trees constructed during fitness evaluation. While Krawiec's approach bears some similarity with the algorithm presented here, there are a number of differences: the fixed number of new features introduces a parameter that must be altered for each new problem (whilst the algorithm presented here specifies a *minimum* number of features the actual number rises and falls with the number of original features in the dataset); it does not involve any subset selection (other than the implicit selection of original features by their presence in the ADFs); nor does it appear to allow for the inclusion of any original features found to be useful.

Song et al. [22] have used Subset Selection with Genetic Programming to avoid overfitting by individuals to specific subsets, and also to make training on a very large dataset more efficient. Here, we present the entire train set at all times (there is no attempt to make the training more efficient in the same manner as Song et al.) but present the set in a different order at different times (so as to create different train-train and train-test sets during cross-validation) to reduce overfitting by individuals to the train set.

This paper is arranged as follows: the next section describes the initial approach; Section 3 presents results from its use on a number of well-known datasets and discusses the results. This is followed by some amendments to the algorithm and further results; finally Section 4 presents some conclusions and future directions.

2. The GAP algorithm

In this work we have used the WEKA [24] implementation of C4.5, known as J48, to examine the performance of our Genetic Algorithm and Programming (GAP) approach. This is a wrapper approach, in which the fitness of individuals is evaluated by performing 10-fold cross validation using the same inducer as used to create the final classifier: C4.5(J48). The approach consists of two phases:

2.1. Feature creation

An initial population of 101 genotypes is created at random. Each genotype consists of n trees, where n is the number of numeric valued attributes in the dataset, subject to a minimum of 7. This minimum is chosen to ensure that, for datasets with a small number of numeric features, the initial population contains a sufficient number of compound features. A tree can be either an original feature or an ADF. That is, a genotype consists of n GP trees (where n is the number of attributes in the original dataset, but at least 7), each of which may contain 1 or more nodes. The chance of a node being a leaf node (a primitive attribute) is determined by:

$$P_{\text{leaf}} = 1 - \frac{1}{(\text{depth} + 1)}$$

where *depth* is the depth of the tree at the current node. Hence a root node will have a depth of 1, and therefore a probability of 0.5 of being a leaf node. Nodes at depth 2 will

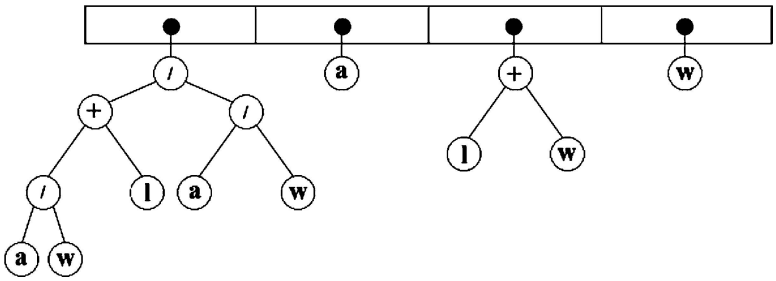

Figure 1. Sample genotype.[1].

have a 0.67 probability of being a leaf node, and so on. If a node is a leaf node, it takes the value of one of the original features chosen at random. Otherwise, a function is randomly chosen from the set $\{*, /, +, -, \%\}$ and two child nodes are generated. In this manner there is no absolute limit placed on the depth any one tree may reach but the average depth is limited. This initialisation method was compared against the well known ramped half and half method—this is discussed further in Section 3.3.

During the initial creation no two trees in a single genotype are allowed to be alike, though this restriction is not enforced in later stages. Additionally, nodes with '$-$', '$\%$' or '$/$' for functions cannot have child nodes that are equal to each other. In order to enforce this child nodes within a function '$*$' or '$+$' are ordered alphabetically to enable comparison (e.g. [width + length] will become [length + width]). For the sake of simplicity, no random constants are used.

An individual is evaluated by constructing a new dataset with one feature for each tree in the genotype. During calculation of instance values the result of any invalid calculation (e.g. division by 0, or a calculation involving a missing value) is replaced by zero. The resulting dataset is then passed to a C4.5(J48) classifier (using default parameters), whose performance on the dataset is evaluated using 10-fold cross validation.[2] The percentage correct is then assigned to the individual and used as the fitness score.

Once the initial population has been evaluated, several generations of selection, crossover, mutation and evaluation are performed. After each evaluation, if the fittest individual in the current generation is fitter than the fittest so far, a copy of it is set aside and the generation noted. The fittest genotype is always copied unchanged into the next generation. We use tournament selection to select the parents of the next generation, with two-point crossover possibly occurring between the ADFs of the two selected parents (whole trees are exchanged between genotypes).[3] There is an additional chance that crossover will occur between two ADFs at a randomly chosen locus (sub-trees are exchanged between trees at the same position in each parent). Mutation may occur at any node, whereby a randomly created subtree replaces the selected node. We also use a form of inversion where the order of the trees between two randomly chosen loci is reversed.

The evolutionary process continues until a minimum number of generations have passed and the fittest genotype has reached a specific age. There is no maximum generation, but in practice very rarely have more than 50 generations been necessary, and often fewer than 30 are required. This is still a lengthy process, as performing 10-fold cross validation for each member of the population is very processor intensive. The extra time required can justified

by the improvement in the results over using, e.g., a single train and test set (results not shown). Information Gain, the fitness criterion employed by both Otero and Ekárt, is much faster but is only applicable to a single feature—it cannot provide the fitness criterion for a set of features.

Once the termination criteria have been met the fittest individual is used to seed the feature selection stage.

2.2. Feature selection

The fittest individual from the feature creation stage (ties broken randomly) is analysed to see if any of the original features do not appear. If some initial attributes do not appear as elementary (single node) trees, they are added to the genotype as elementary trees. This genotype (with the same size as the fittest individual, plus at most the initial number of attributes) replaces the initial individual and is used as the basis of the second stage.

For the GA a population of 101 bit strings is randomly created. The strings have the same number of bits as the genotype has trees—there is one bit for every attribute in the extended genotype. The last member of the population, the 101st bit string, is not randomly created but is initialised to all ones. This ensures that there are no missing alleles at the start of the selection process.

A new dataset is constructed with one attribute for every tree in the extended genotype. In an attempt to reduce over-fitting of the data, the order of the dataset is randomly reordered at this point. This has the effect of providing a different split of the data for 10-fold cross validation during the selection stage, giving the algorithm a chance of recognising trees that performed well only on the particular data partition in the creation stage. As a result of the reordering, it is usually the case that the fitness score of individuals drops at the start of the selection stage before improving again. Often individuals in the selection stage fail to reach the same fitness levels as seen in the construction stage, but solutions should be more robust.

Each bit string is evaluated by taking a copy of the parent dataset and removing every attribute that has a '0' in the corresponding position in the bit string. As in the feature creation stage, this dataset is then passed to a C4.5(J48) classifier whose performance on the dataset is evaluated using 10-fold cross validation. The percentage correct is then assigned to the bit string and used as the fitness score.

If the fittest individual in the current generation has a higher fitness score than the fittest so far (from the selection stage), or it has the same fitness score and fewer '1's, a copy of it is set aside and the generation noted. As in the feature creation stage the cycles of selection, crossover, mutation, and evaluation continue until a minimum number of generations have passed and the fittest genotype has reached a specific age.

2.3. Experimental settings

The parameter settings used in the experiments were as follows (except where otherwise noted in the text):

- Population size: 101 (this is the same for both stages).

- Tournament selection: Group size of 8, with a probability of 0.3 of selecting the fittest of the group (otherwise a 'winner' is selected at random from the tournament group). Tournament selection is the same for both stages.
- Crossover: The probability of crossover is 0.6, (This probability applies separately to two-point crossover exchanging complete trees between individuals and to crossover exchanging sub-trees between trees.), Two-point crossover is the same for both stages.
- Mutation: There is a probability of 0.008 per node that the node will be replaced by a new sub-tree during the construction stage, and a probability of 0.005 per bit that the bit will be flipped during the selection stage.
- Inversion: Inversion occurs with a probability of 0.2 per individual. Inversion is the same for both stages.
- Termination criteria: At least 10 generations have passed, and the fittest individual is at least 6 generations old. The termination criteria are the same for both stages.

Experimentation on varying these parameters (not shown) has found the algorithm to be fairly robust to their setting.

3. Experimentation

The experiments outlined in this section had a number of goals: (1) to assess the utility of the GAP algorithm as a feature construction method for C4.5; (2) to compare the effects of the initialisation method outlined above with ramped half and half; (3) to compare different orders for the create and select stages; (4) to see if creation and selection could be successfully combined in a single stage; (5) to see if the GAP algorithm could be successfully applied to classifiers other than C4.5.

We have used ten well-known data sets from the UCI repository to examine the performance of the GAP algorithm. The UCI datasets were chosen because they consisted entirely of numeric attributes (though the algorithm can handle some nominal attributes, as long as there are two or more numeric attributes—it ignores the presence of the nominal attributes, but does pass them to C4.5(J48)). Table 1 shows the details of the ten datasets used here.

For performance comparisons the tests were performed twenty times on each dataset (in which 90% of the data was used for training and 10% withheld for testing in each run).

3.1. Initial results

The highest classification score for each dataset is shown in Table 2 in bold. The first column shows the performance of the GAP algorithm on unseen test data, the third column the performance of C4.5(J48) on test data, and the last column shows the results of the paired *t*-test. *T*-test results that are significant at the 95% confidence level are shown in bold.

The GAP algorithm out-performs C4.5(J48) on eight out of ten datasets, and provides a significant improvement on three (Glass Identification, New Thyroid, and Wisconsin Breast Cancer Original)—two of which are significant at the 99% confidence level. There are no datasets on which the GAP algorithm performs significantly worse than C4.5(J48) alone.

The standard deviation of the GAP algorithm's results do not seem to differ greatly from that of C4.5(J48); there are five datasets where the GAP algorithms' standard deviation is greater and five where it is smaller. This is perhaps the most surprising aspect of the results, given that the GAP algorithm is unlikely to produce the same result twice when presented with exactly the same data, whereas C4.5(J48) will always give the same result if presented with the same data.

Table 1. UCI dataset information.

Dataset	No. of numeric features	No. of nominal features	No. of classes	No. of Instances	No of instances with missing attributes
BUPA Liver Disorder (Liver)	6	0	2	345	0
Glass Identification (Glass)	9	0	6	214	0
Ionosphere (Iono.)	34	0	2	351	0
New Thyroid (NT)	5	0	3	215	0
Pima Indians Diabetes (Diab.)	8	0	2	768	0
Sonar	60	0	2	208	0
Vehicle	18	0	4	846	0
Wine Recognition (Wine)	13	0	3	178	0
Wisconsin Breast Cancer – New (WBC New)	30	0	2	569	0
Wisconsin Breast Cancer – Original (WBC Orig.)	9	0	2	699	16

Table 2. Comparative performance of GAP algorithm and C4.5 (J48).

Dataset	GAP	S.D.	C4.5 (J48)	S.D.	Paired *t*-test
Liver	65.97	11.27	**66.37**	8.86	−0.22
Glass	**73.74**	9.86	68.28	8.86	**3.39**
Iono	89.38	4.76	**89.82**	4.79	−0.34
NT	**96.27**	4.17	92.31	4.14	**3.02**
Diab	**73.50**	4.23	73.32	5.25	0.19
Sonar	**73.98**	11.29	73.86	10.92	0.05
Vehicle	**72.46**	4.72	72.22	3.33	0.20
Wine	**94.68**	5.66	93.27	5.70	0.85
(WBC New)	**95.62**	2.89	93.88	4.22	1.87
(WBC Orig.)	**95.63**	1.58	94.42	3.05	**2.11**
Overall average	83.12		81.77		2.91

Table 3. Analysis of the constructed features for each data set.

	Results			
Dataset name	No. of features in Dataset	Average no. of features	Average no. of ADFs	Minimum no. of ADFs
Liver	6	4.9	2.6	1
Glass	9	7.1	3.1	1
Iono	34	19.7	7.6	4
NT	5	3.2	2.3	1
Diab	8	6.4	2.7	1
Sonar	60	38.0	13.6	5
Vehicle	18	16.3	5.5	2
Wine	13	5.2	2.1	0
(WBC New)	30	15.7	7.0	4
(WBC Orig.)	9	5.0	2.9	1

3.2. Analysis

We were interested in whether the improvement over C4.5(J48) is simply the result of the selection stage choosing an improved subset of features and discarding the new constructed features. An analysis of the attributes output by the GAP classifier algorithm, and the use made of them in C4.5's decision trees, shows this is not the case.

As noted above, the results in Table 2 were obtained from twenty runs on each of ten UCI datasets, i.e. a total of two hundred individual solutions. In those two hundred individuals there are a total of 2,425 trees: 982 ADFs and 1,443 original features—a ratio of roughly two constructed features to three original features. *All but two of the two hundred individuals contained at least one constructed feature*. Table 3 gives details of the average number of ADFs per individual for each dataset (the number of original features used is not shown).

Knowing that the feature selection stage continues as long as there is a reduction in the number of attributes without reducing the fitness score, we can assume that C4.5(J48) is making good use of all the attributes in most if not all of the winning individuals. This can be demonstrated by looking in detail at the attributes in a single solution, and the decision tree created by C4.5(J48).

One of the best performers on the New Thyroid dataset had three trees, two of them ADFs and hence constructed features. The original dataset contains five features and the class:

(a) T3-resin uptake test (A percentage).
(b) Total Serum thyroxin as measured by the isotopic displacement method.
(c) Total serum triiodothyronine as measured by radioimmuno assay.
(c) Basal thyroid-stimulating hormone (TSH) as measured by radioimmuno assay.
(d) Maximal absolute difference of TSH value after injection of 200 micro grams of thyrotropin-releasing hormone as compared to the basal value.

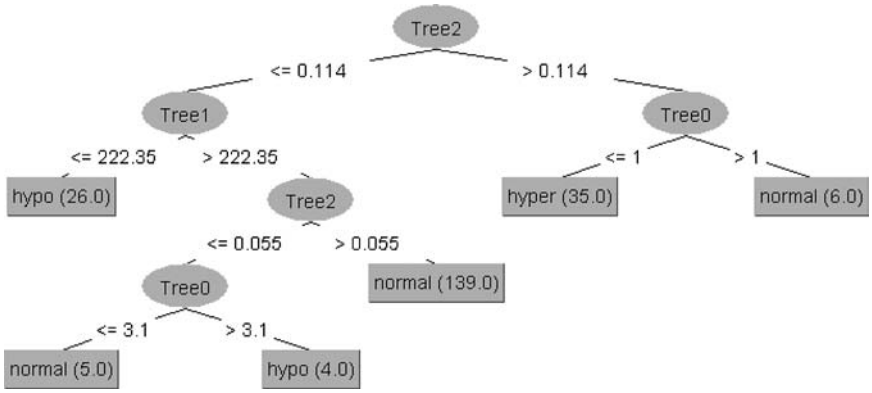

Figure 2. New Thyroid decision tree using constructed features.

(e) Class attribute. (1 = normal, 2 = hyper, 3 = hypo)

In the chosen example the newly constructed features were:

- "e" becomes `Tree0`
- "((a/d)∗b)" becomes `Tree1`
- "(b/a)" becomes `Tree2`

The decision tree created by C4.5 using these constructed features is shown in figure 2 (the numbers after the class prediction indicate the count of instances correctly/incorrectly classified by that node).

It is apparent that C4.5(J48) is using the constructed features to classify a large majority of the instances, and only referring to an original feature (`Tree0`, or the original feature "e") to classify 50 of 215 instances. The decision tree is also simpler than that created using the five original features (figure 3)—11 nodes compared with 17, while using only four of the five original features (the original feature "c" is not used in the construction of the new feature set).

3.3. Comparison of the initialisation method with ramped half and half

As mentioned previously, we compared the performance of the initialisation method outlined in Section 2.1 (hereafter referred to as P(leaf)) with the Ramped Half and Half method. As can be seen in Table 4, ramped half and half provides significantly improved fitness scores (with a better fitness in 8 of 10 datasets tested), but this results in a *lower* average test score (test scores are lower in 7 of ten datasets).

Comparing the size and number of trees in the solutions generated using the different methods provides a possible explanation—the solutions generated with ramped half and half have on average 0.88 more trees and 3.6 more nodes per tree than the solutions generated with P(leaf). We speculate that the extra nodes have the same effect as too many nodes in an artificial neural network—causing the solutions to overfit the data.

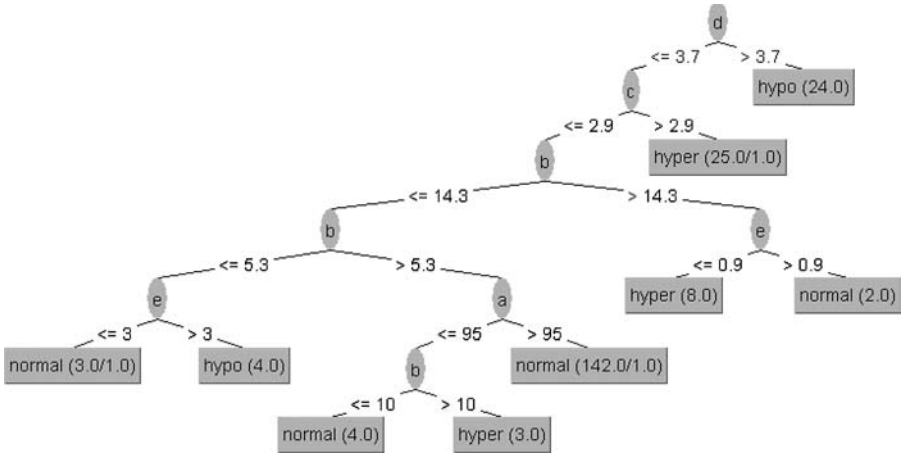

Figure 3. New Thyroid decision tree using original attributes.

3.4. Computational effort

As this algorithm employs a wrapper approach it is computationally expensive when compared to C4.5 alone (though for an embarked application only the best feature set would be used, and the computational effort would differ from C4.5 only in the time taken to transform the dataset). For instance on the New Thyroid dataset the algorithm runs for an average of 23.8 generations (this figure includes both create and select stages). With a population size of 101 (and assuming an unlikely worst case where every genotype is un-

Table 4. Comparing the initialisation method with ramped half and half.

	Train				Test			
Data set	P(leaf)	S.D.	RHH	S.D.	P(leaf)	S.D.	RHH	S.D.
Liver	75.04	2.37	78.10	1.59	65.97	11.27	66.52	9.26
Glass	78.17	1.95	80.12	2.30	73.74	9.86	68.83	11.03
Iono.	95.99	0.87	95.89	1.04	89.38	4.76	89.62	4.99
NT	98.22	0.68	98.94	0.46	96.27	4.17	94.47	4.81
Diab.	78.15	0.96	79.65	0.92	73.50	4.23	73.82	4.43
Sonar	92.33	1.27	92.23	2.68	73.98	11.29	73.16	11.94
Vehicle	78.82	1.21	79.33	1.76	72.46	4.72	72.28	5.00
Wine	98.47	0.80	99.13	0.68	94.68	5.66	93.86	7.49
(WBC New)	97.86	0.47	98.55	0.42	95.62	2.89	94.65	3.07
(WBC Orig.)	97.65	0.38	98.05	0.28	95.63	1.58	95.63	2.28
Overall average	89.07		90.00		83.12		82.28	

familiar and requires fitness evaluation) this means ∼2404 genotypes requiring evaluation. As fitness evaluation involves 10-fold cross-validation on the training data, this equates to roughly 24038 times the amount of computational effort of C4.5 alone[4]—something in the region of 3 minutes per run on a 1.4 GHz PC.

The results achieved with the GAP algorithm compare favourably with those achieved by Bagging C4.5 [5] with the same computational effort—Bagging C4.5(J48) (using the Weka implementation of Bagging) with 24038 iterations provides an average performance of 94.40% (and standard deviation of 3.60%) on the New Thyroid dataset (using the same 20 train/test splits as the results shown above)—compared with 96.27% for the GAP algorithm.[5] A paired *t*-test comparing these two methods gives a score of 7.57—significant at the 99% confidence level.

3.5. The order of the create and select stages

As noted in the introduction, Vafaie and DeJong [23] have used a very similar approach to improve the performance of C4.5. They use feature selection (GA) followed by feature construction (GP). We have examined the performance of our algorithm as described above with the two processes occurring in the opposite order. Results indicate that GAP gives either equivalent (e.g. Wisconsin Breast Cancer) or better performance (e.g. New Thyroid) (Table 5). We suggest this is due to GAP's potential to construct new features in a less restricted way, i.e. its ability to use all the original features during the create stage. For instance, on the New Thyroid dataset the select stage will always remove either feature a or feature b thus preventing the create stage from being able to construct the apparently useful feature "b/a" (see Section 3.2). That is, on a number of datasets there is a significant difference (at the 95% confidence level) in the results brought about by changing the order of the stages.

Table 5. Comparison of ordering of create and select stages.

	Create then select		Select then create	
Dataset	Test	S.D.	Test	S.D.
Liver	65.97	11.27	**67.42**	8.23
Glass	**73.74**	9.86	68.75	6.36
Iono	**89.38**	4.76	89.02	4.62
NT	**96.27**	4.17	93.67	5.82
Diab	73.50	4.23	**74.09**	4.46
Sonar	**73.98**	11.29	73.16	9.05
Vehicle	72.46	4.72	72.46	3.01
Wine	94.68	5.66	**94.69**	3.80
(WBC New)	95.62	2.89	**95.88**	3.15
(WBC Orig.)	**95.63**	1.58	95.13	2.16
Overall Average	83.12		82.43	

3.6. *The importance of reordering the dataset*

In section two it was mentioned that the dataset was randomly reordered before the second stage commenced, providing a different view of the data for 10-fold cross validation during fitness evaluation and, it was hoped, this would reduce over fitting and improve the performance on the test data. Is this what actually happens? In order to test this hypothesis we turned off the randomisation and retested the algorithm. The first impression is that there is no important difference between the two sets of results—there are 5 datasets where not reordering gives a better result and 5 where it is worse. However, there are now only two (rather than three) datasets on which the algorithm provides a significant improvement over C4.5(J48) (New Thyroid and Wisconsin Breast Cancer); and most importantly the *t*-test performed over the 200 runs from all datasets no longer shows a significant improvement. The results are shown in Table 6 (the column for paired *t*-test shows the results for testing the algorithm without reordering against C4.5(J48)).

3.7. *Combining creation and selection in a single stage*

Having successfully tested the algorithm with two separate stages, we redesigned it to move feature selection into the construction stage. Feature construction occurs as before but each tree now has a bit flag associated with it, to determine whether the tree is passed to C4.5(J48) for evaluation. During crossover each tree retains its associated bit flag, which is subject to the same chance of mutation as during the second stage (0.005).

Testing the amended algorithm with the same parameter values as before gives a much shorter run time (not surprisingly, roughly half the time of the two stage algorithm) but with poorer results—an overall average of 82.20%[6] (though this is still an improvement over unaided C4.5(J48)).

Table 6. Comparative performance of GAP algorithm (with and without reordering) and C4.5 (J48).

Dataset	GAP reorder	S.D.	GAP no reorder	S.D.	C4.5 (J48)	S.D.	Paired *t*-test
Liver	65.97	11.27	**66.65**	7.84	66.37	8.86	0.17
Glass	**73.74**	9.86	69.74	9.79	68.28	8.86	0.61
Iono	89.38	4.76	89.77	4.24	**89.82**	4.79	−0.04
NT	96.27	4.17	**97.22**	3.78	92.31	4.14	**3.99**
Diab	**73.50**	4.23	71.74	4.34	73.32	5.25	−1.37
Sonar	73.98	11.29	**75.22**	8.32	73.86	10.92	0.50
Vehicle	**72.46**	4.72	71.94	4.43	72.22	3.33	−0.28
Wine	**94.68**	5.66	94.08	5.09	93.27	5.70	0.55
(WBC New)	**95.62**	2.89	94.56	2.58	93.88	4.22	0.71
(WBC Orig.)	95.63	1.58	**95.71**	1.59	94.42	3.05	**2.03**
Overall average	83.12		82.66		81.77		1.82

There are three primary differences between the two versions of the algorithm that may account for this drop in performance:

1. With a single stage we are asking the algorithm to do the same amount of work in half the time.
2. There is no longer an opportunity to randomly reorder the dataset between stages.
3. There is no longer an opportunity to reintroduce any original attributes that have been dropped during the first stage.

There seems no reasonable way to address the third of these differences with a single stage approach, but the other two can be compensated for. Firstly we can change the termination criteria—by doubling both the minimum number of generations to 20 and the age of the fittest individual to 12 generations. Doing this does improve the result (to an overall average result of 82.88%) but not sufficiently to bring it into line with a two stage process.

Additionally we can randomly reorder the dataset. We considered two approaches to this. The first was to have two versions of the dataset from the start, with the same data but in a different order, and simply alternate between datasets when evaluating each generation (i.e. the first dataset was used to evaluate even numbered generations, the second to evaluate odd numbered)—this approach did not seem to improve the results (slightly worse than having no reordering at 82.24%). The second, more successful, approach was to reorder the dataset once the termination criteria had been reached. That is, run as before but when the fittest individual reaches 12 generations old reorder the dataset, re-evaluate the current generation and reset the fittest individual, then continue until the termination criteria are met again. The results obtained with a longer run time and randomly reordering the dataset part-way through are shown in Table 7 (the column for paired *t*-test shows the results for testing the single stage algorithm against C4.5(J48)):

Table 7. Comparative performance of a single stage and C4.5 (J48).

Dataset	2 stage	S.D.	1 stage	S.D.	C4.5 (J48)	S.D.	Paired *t*-test
Liver	65.97	11.27	**66.55**	8.10	66.37	8.86	0.11
Glass	**73.74**	9.86	71.84	10.26	68.28	8.86	1.78
Iono	89.38	4.76	**90.69**	4.66	89.82	4.79	0.96
NT	96.27	4.17	**96.49**	3.98	92.31	4.14	**3.69**
Diab	73.50	4.23	**73.64**	5.11	73.32	5.25	0.24
Sonar	73.98	11.29	**75.89**	9.00	73.86	10.92	0.80
Vehicle	**72.46**	4.72	72.11	4.60	72.22	3.33	−0.09
Wine	94.68	5.66	**96.10**	4.08	93.27	5.70	1.76
(WBC New)	95.62	2.89	**95.71**	4.39	93.88	4.22	1.71
(WBC Orig.)	**95.63**	1.58	95.56	2.52	94.42	3.05	1.62
Overall average	83.12		83.46		81.77		3.62

Although the single stage algorithm out-performs C4.5(J48)) on only one dataset at the 95% confidence level (a *t*-test of 1.96 or higher), as compared to three datasets for the two stage algorithm, it outperforms C4.5(J48) on everything but the Vehicle dataset (and then loses only by a very small margin). It also improves on the performance of the two stage version on seven out of ten datasets, resulting in an increase of the (already high) overall confidence of improvement over C4.5(J48).

3.8. Applying GAP to other classification algorithms

As the version of C4.5 used is part of the Weka package, it is a simple matter to replace C4.5(J48) with different classifiers and thus test the GAP algorithm with a number of different classification techniques. We replaced C4.5(J48) with IBk (a *k*-nearest neighbour classifier [1] with $k = 1$) and Naïve Bayes (a probability based classifier [11]). Tables 8 and 9 present the results of using GAP with these algorithms.

Table 8 shows that IBk on its own offers marginally better performance over the ten datasets than C4.5 (on average only—there are several individual datasets on which C4.5 performs better) and perhaps offers the GAP algorithm less scope for improvement. There is no significant overall improvement over IBk at the 96% level and no improvement at all on half the datasets, but there are two datasets on which GAP does offer a significant improvement (Glass and Ionosphere) and none where there is a significant drop in performance. Overall the result is competitive with GAP using C4.5.

By looking at the overall average of Table 9 it can be quickly seen that, on its own, Naïve Bayes cannot compete with C4.5 or IBk over the ten datasets—it is approx. 6% worse on average (though again there are individual datasets where Naïve Bayes performs better than the other two—e.g., New Thyroid, Wine). This relatively poor performance gives the GAP algorithm much greater scope for improvement. In fact, the GAP algorithm brings the results very closely into line with those achieved using C4.5 and IBk. Further GAP with Naïve Bayes outperforms both IBk and C4.5 on their own when averaged over the ten datasets.

Table 8. Results with IBk.

Dataset	GAP	S.D.	IBk	S.D.	*t*-test
Liver	60.89	7.65	62.62	8.68	−0.86
Glass	73.92	9.10	68.79	9.75	**2.19**
Iono	91.38	4.00	86.95	4.53	**3.62**
NT	95.36	3.28	96.95	3.73	−1.91
Diab	68.96	6.27	69.90	4.27	−0.64
Sonar	83.72	7.88	86.65	6.39	−1.39
Vehicle	72.46	5.42	70.03	4.10	1.87
Wine	96.00	4.88	95.44	5.81	0.52
(WBC New)	94.82	3.25	95.44	2.92	−1.26
(WBC Orig.)	95.64	2.51	95.42	2.16	0.41
Overall average	83.31	5.42	82.82	5.23	1.01

Table 9. Results with Naïve Bayes.

Dataset	GAP	SD	N.B	SD	*t*-tsest
Liver	71.35	8.51	54.19	9.61	**5.82**
Glass	61.45	7.73	48.50	14.03	**4.23**
Iono	90.60	4.38	82.37	7.31	**5.09**
NT	97.18	3.48	97.20	3.83	−0.02
Diab	75.77	4.83	75.13	5.00	0.59
Sonar	77.64	8.77	67.16	7.53	**5.80**
Vehicle	69.03	4.96	43.98	4.61	**20.69**
Wine	96.40	4.80	97.99	3.85	−1.47
(WBC New)	96.75	2.20	93.26	4.79	**4.04**
(WBC Orig.)	95.92	2.16	96.06	1.74	−0.32
Overall average	83.21	5.18	75.58	6.23	**9.68**

While neither of the two new classifiers provide an improvement on GAP's overall result using C4.5 both results are competitive—regardless of the performance of the classifier on its own. It seems as if there is a ceiling on the overall results achievable with any one classifier. While using any particular classifier GAP may perform well on some datasets and worse on others, the average seems to settle out at somewhere near 83% no matter which classifier is employed. That is, GAP appears to provide a robustness to the classifier techniques used.

Table 10. Performance of GAP and other algorithms on the UCI datasets.

Dataset	GAP (J48)	GAP (IBK)	GAP (NB)	C4.5 (J48)	IBk	N.B.	HIDER	XCS	O. F. A.	LSVM	Krawiec
Liver	66.55	60.89	**71.35**	66.37	62.62	54.19	64.29	67.85	57.01	68.68	
Glass	71.84	**73.92**	61.45	68.28	68.79	48.50	70.59	72.53	69.56		66.39
Iono.	90.69	**91.38**	90.60	89.82	86.95	82.37				87.75	
NT	96.49	95.36	97.18	92.31	96.95	**97.20**					
Diab.	73.64	68.96	75.77	73.32	69.90	75.13	74.1	68.62	69.8	**78.12**	76.41
Sonar	75.89	83.72	77.64	73.86	**86.65**	67.16	56.93	53.41	79.96		
Vehicle	72.11	**72.46**	69.03	72.22	70.03	43.98					
Wine	96.10	96.00	96.40	93.27	95.44	97.99	96.05	92.74	**98.27**		
WBC new	95.71	94.82	**96.75**	93.88	95.44	93.26					
WBC orig	95.56	95.64	95.92	94.42	95.42	96.06	95.71	**96.27**	94.39		

3.9. A rough comparison to other algorithms

Table 10 presents a number of published results we have found regarding the same ten UCI datasets using other machine learning algorithms. The first 3 columns present results for the single stage version of the algorithm using C4.5(J48), IBk and Naïve Bayes—the next 3 columns present the performance of those algorithms on their own. Cells in the table are left blank where algorithms were not tested on the dataset in question. The highest classification score for each dataset is shown in bold underline.

The results for HIDER and XCS were obtained from [7], those for O.F.A. ('Ordered Fuzzy ARTMAP', a neural network algorithm) from [6], LSVM (Lagrangian Support Vector Machines) from [16] and Krawiec from [15].

The results are by no means an exhaustive list of current machine learning algorithms, nor are they guaranteed to be the best performing algorithms available, but they give some indication of the relative performance of our approach—which appears to be very good.

4. Conclusion

In this paper we have presented an approach to improve the classification performance of the well-known induction algorithm C4.5. We have shown that GP individuals consisting of multiple trees/ADFs can be used for effective feature creation and that solutions, combined with feature selection via a GA in either a separate or the same stage, can give significant improvements to the classification accuracy of C4.5. We have also indicated that randomly reordering the dataset part-way through the process may help to reduce the problem of over fitting. We have shown that the same algorithm can be used successfully with more than one type of classifier. Given that Table 10 shows that using a classifier more appropriate to the dataset improves the results of the GAP algorithm, future work will look at the possibility of using evolution to select the most appropriate classifier for a particular problem.

Notes

1. Genotypes have a minimum of 7 trees, but only 4 are shown here due to space constraints. The sample genotype has been constructed using a very simple dataset with 3 attributes—Area, Length and Width.
2. This cross-validation is performed using the supplied train set only —it does not involve any data later used to test the effectiveness of the algorithm.
3. Tests using this algorithm have shown no significant difference between two-point crossover with inversion and uniform crossover. Of ten datasets tested, 5 showed a higher fitness with two-point crossover and inversion and 5 a lower fitness, with no significant difference overall. Similar results were obtained comparing two-point crossover with inversion and uniform crossover with inversion.
4. The computational effort required to transform the dataset is minimal compared to that required to perform 10-fold cross-validation, so it has been ignored for this calculation.
5. It should perhaps be noted that Bagging C4.5(J48) with only 10 iterations provides an average performance of 94.27%—the additional iterations add very little value on the New Thyroid dataset.
6. It should be noted that the results for some of the datasets have a fairly high standard deviation, and so can show some variation in the results from run to run. For this reason we have taken to using the average result over all 10 datasets as a useful (and briefer!) indicator of the performance of the algorithm.

References

1. D. Aha and D. Kibler, "Instance-based learning algorithms," Machine Learning vol. 6, pp. 37–66, 1991.
2. M. Ahluwalia and L. Bull, "Co-evolving functions in genetic programming: Classification using *k*-nearest neighbour," in GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, W. Banzhaf, J. Daida, G. Eiben, M-H. Garzon, J. Honavar, K. Jakeila, and R. Smith (Eds.), Morgan Kaufmann: San Mateo, 1999, pp. 947–952.
3. Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," Neural Computation, vol. 9, no. 7, pp. 1545–1588, 1996.
4. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone, Genetic programming—An Introduction on the Automatic Evolution of Computer Programs and its Applications, Morgan Kaufmann: San Mateo, 1998.
5. L. Breiman, "Bagging predictors," Machine Learning, vol. 24, no. 2, pp. 123–140, 1996.
6. I. Dagher, M. Georgiopoulos, G. L. Heileman, and G. Bebis, "An ordering algorithm for pattern presentation in fuzzy ARTMAP that tends to improve generalization performance," IEEE Transactions on Neural Networks vol. 10, no. 4, pp. 768–778, 1999.
7. P. Dixon, D. Corne, and M. Oates, "A preliminary investigation of modified XCS as a generic data mining Tool," in Advances in Learning Classifier Systems, P-L. Lanzi, W. Stolzmann, and S. Wilson (Eds.), Springer, 2001, pp. 133–151.
8. A. Ekárt and A. Márkus, "Using genetic programming and decision trees for generating structural descriptions of four bar mechanisms," Artificial Intelligence for Engineering Design, Analysis and Manufacturing, vol. 17, no. 3. 2003, to appear.
9. I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," Journal of Machine Learning Research, vol. 3, pp. 1157–1182, 2003.
10. J. Holland, Adaptation in Natural and Artificial Systems. Univ. Michigan, 1975.
11. G. John and P. Langley, "Estimating continuous distributions in bayesian classifiers," in Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann: San Mateo, 1995, pp. 338–345.
12. J. Kelly and L. Davis, "Hybridizing the genetic algorithm and the K nearest neighbors classification algorithm", in Proceedings of the Fourth International Conference on Genetic Algorithms, R. Belew and L. Booker (Eds.), Morgan Kaufmann: San Mateo, 1991, pp. 377–383.
13. R. Kohavi and G. John, "Wrappers for feature subset selection," Artificial Intelligence Journal, vols. 1–2, pp. 273–324, 1997.
14. J. Koza, Genetic Programming, MIT Press, 1992.
15. K. Krawiec, "Genetic programming-based construction of features for machine learning and knowledge discovery tasks," Genetic Programming and Evolvable Machines, vol. 3, no. 4, pp. 329–343, 2002.
16. O. Mangasarian and D. Musicant, "Lagrangian support vector machines," Journal of Machine Learning Research vol. 1, pp. 161–177, 2001.
17. T. M. Mitchell, Machine Learning. McGraw-Hill: New York, 1997.
18. F. Otero, M. Silva, A. Freitas, and J. Nievola, "Genetic programming for attribute construction in data mining," in Proceedings of Genetic Programming: 6th European Conference, EuroGP 2003, Essex, UK, Springer, 2003, pp. 384–393.
19. J. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann: San Mateo, 1993.
20. M. Raymer, W. Punch, E. Goodman, and L. Kuhn, "Genetic programming for improved data mining—Application to the biochemistry of protein interactions," in Proceedings of the Second Annual Conference on Genetic Programming, J. Koza, K. Deb, M. Dorigo, D. Fogel, M.Garzon, H. Iba, and R. Riolo (Eds.), Morgan Kaufmann: San Mateo, 1996, pp. 375–380.
21. W. Siedlecki and J. Sklansky, "On automatic feature selection," International Journal of Pattern Recognition and Artificial Intelligence vol. 2, pp. 197–220, 1988.
22. D. Song, M. I. Heywood, and A. Nur Zincir-Heywood, "A linear genetic programming approach to intrusion detection," Genetic and Evolutionary Computation—GECCO-2003, E. Cantú-Paz et al. (Eds.), 2003, pp. 2325–2336.
23. H. Vafaie and K. De Jong, "Genetic algorithms as a tool for restructuring feature space representations," in Proceedings of the International Conference on Tools with A.I., IEEE Computer Society Press, 1995.
24. I. Witten and E. Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann: San Mateo, 2000.