# A Bootstrapping Approach to Reduce Over-fitting in Genetic Programming

Jeannie Fitzgerald
BDS Group
CSIS Department
University of Limerick, Ireland
jeannie.fitzgerald@ul.ie

R. Muhammad Atif Azad
BDS Group
CSIS Department
University of Limerick, Ireland
atif.azad@ul.ie

Conor Ryan
BDS Group
CSIS Department
University of Limerick, Ireland
conor.ryan@ul.ie

## ABSTRACT

Historically, the quality of a solution in Genetic Programming (GP) was often assessed based on its performance on a given training sample. However, in Machine Learning, we are more interested in achieving reliable estimates of the quality of the evolving individuals on *unseen data*. In this paper, we propose to simulate the effect of unseen data during training without actually using any additional data. We do this by employing a technique called *bootstrapping* that repeatedly re-samples *with replacement* from the training data and helps estimate sensitivity of the individual in question to small variations across these re-sampled data sets. We minimise this sensitivity, as measured by the *Bootstrap Standard Error*, together with the training error, in an effort to evolve models that generalise better to the unseen data.

We evaluate the proposed technique on four binary classification problems and compare with a standard GP approach. The results show that for the problems undertaken, the proposed method not only generalises significantly better than standard GP while the training performance improves, but also demonstrates a *strong* side effect of containing the tree sizes.

## Categories and Subject Descriptors

1.2.2 [**Artificial Intelligence**]: Automatic Programming - Program Modification

## Keywords

Genetic Programming, Binary Classification, Generalisation

## 1. INTRODUCTION

In recent years several researchers [7, 15, 24] have expressed the view that in the past, GP [23] research effort may have placed too much importance on consistency at the expense of generalisation ability. Here, *consistency* means getting predictably repeatable results that have low variance across different runs, whereas *generalisation* means the ability of a system to reliably model a phenomenon over unseen data, despite learning only from a training sample.


*GECCO'13 Companion,* July 6–10, 2013, Amsterdam, The Netherlands.


Despite the well established importance of generalisation in the wider Machine Learning paradigm, the focus of a considerable amount of GP research has been on achieving consistency in training results, possibly in the belief that a high level of consistency would automatically translate to good generalisation. However, over-emphasis on training performance may lead to over-training and brittle solutions [24] which do not perform well on unseen data.

Kushchu [24], in a review of generalisation in GP, expressed the opinion that until such time as GP researchers adopt the notion of performance evaluation based on generalisation ability, genetic based learners and GP in particular would be limited in their scope.

More recently, GP researchers have shown a strong interest in improving the generalisation ability of evolved solutions [3, 5, 10, 28], and to this end many have adopted a strategy of training the learner on a labelled randomly generated subset of data and then evaluating its performance on a test set of unlabelled instances.

In this paper, we use a similar experimental model on four binary classification tasks and propose to evolve solutions that are not only good on training data but are also not very sensitive to small variations in the training data. First, we compute the classification error rate of a solution on the given training set. Next, we ascertain the statistical confidence in this error rate by using a technique called *bootstrapping*, whereby we re-sample *with replacement* from the training set multiple times and compute the error rate on each re-sampled data set. The standard deviation of these re-sampled error rates, also called the *BootStrap Standard Error (BSE)*, indicates how sensitive the solution in question is to small changes in the training set. We hypothesise that the higher this sensitivity, the lower is the likelihood of the corresponding solution generalising to completely unseen (test) data. Therefore, in this paper, we minimise both the error rate on the original training set as well as the sensitivity to the training data.

We find that, for the problems studied, applying bootstrapping improves results on test data, and that in three out of four problems, the performance on training data reliably indicates the performance on the test data. The results with bootstrap also compare favourably with those from several benchmark machine learning algorithms.

Another interesting benefit of the proposed approach is that it produces smaller individuals than standard GP: the results demonstrate a stronger negative size fitness correlation with bootstrapping than that with standard GP. This is surprising because we do not explicitly penalise individuals for growing in size. We hypothesise that this side-effect is due to bootstrapping promoting small yet accurate programs. To support this view we present the size distributions of best of run individuals with bootstrapping and point out a clear skew towards smaller sized individuals.

Section 2 introduces and describes bootstrapping methods and related work, section 3 provides a very brief overview of the cur-

rent state of the art in *bloat* research as it relates to generalisation, in section 4 we provide details of our proposed method, section 5 outlines the various experiments undertaken, and in section 6 we present and discuss the results and outline our conclusions and aspirations for future work on the topic.

## 2. BOOTSTRAPPING

In statistics, bootstrapping [9] or *the bootstrap* is a non parametric technique which can provide a confidence interval for some statistic. In general, its purpose is to assist with non-normal or unknown distributions where the available sample can be used to indirectly assess some properties of the population from which the sample is taken.

Bootstrapping also estimates the sensitivity of a statistic of interest (such as mean) to the given data. This measure of sensitivity is called the *Bootstrap Standard Error* (BSE). For example, to estimate BSE of the mean of a data set of size $m$, we randomly resample from the data set with replacement to create $n$ samples, each sample being of size $m$. Each of the $n$ re-sampled data sets is called a *bootstrap replicate* or a *bootstrap sample*. Then, for each bootstrap replicate we compute its mean. The BSE, then, is the standard deviation of the means obtained thus from these $n$ bootstrap replicates. Much like standard deviation depicts the variability in any statistic of interest, the lower the BSE, the lower is the variability in the mean of the *original* data set and hence the higher is the confidence in that mean.

In machine learning *bootstrap aggregating* (bagging) proposed by Leo Breiman in 1996 [4] is an ensemble method which may be used to improve the performance of machine learning algorithms on regression or statistical classification tasks. Given an initial training set, bagging generates a number of bootstrap replicates (also known as *bags*) by sampling from the initial set uniformly with replacement. However, then, instead of generating a BSE of some statistic of interest on these bags, typically, the Machine learning algorithms produce multiple solutions each trained on a different bag. These multiple solutions are then pooled: in the case of regression averaging is used, whereas for classification some committee voting mechanism is usually employed.

Inspired by the bootstrap, Oakley [26] generated bootstrap replicates of GP programs, creating multiple populations from an initial base population, and applied these to chaotic time series data. In 1999 Iba [20] developed *BagGP* which combined bagging with a co-evolutionary GP approach. Although the performance improvements reported on test data were modest, the results indicated that the method delivered smaller, more robust solutions than those obtained with standard GP when applied to a real world financial problem.

Several GP researchers including [22, 30] successfully applied bagging to a range of problems. In general, the bootstrapping component of this work involved applying resampling techniques to *training instances* to create ensembles from functions trained on different bootstrap replicates. Results show that the method was suitable for both small and very large datasets: it allows effective use of limited training data as well as facilitating the use of very large datasets by dividing them into manageable subsets. Similar work by Gonclaves et al. [17] demonstrated that random sampling of training data could reduce over-fitting. Doucette and Heywood [8] demonstrated that bagging could be used to handle imbalanced data sets for classification tasks by eliminating the class imbalance in the bootstrap sampling phase.

This study uses bootstrapping in a very different way. Rather than working with ensembles, in this paper we propose to produce reliable *individual* learners that generalise to unseen data. However, there is no reason that individuals thus evolved can not then be combined in ensembles. We defer such work for future. Before we discuss the proposed method in detail in section 4, we give a brief account of how generalisation in GP is linked to the phenomenon of code bloat.

## 3. GENERALISATION AND BLOAT

*Program growth without (significant) return in terms of fitness* [27] is a widely accepted definition of the term *bloat* as used in the GP community. The phenomenon of bloat is inconvenient for several reasons, including computational expense, increased run times, deterioration in program comprehensibility and a likely dilution of program semantics. Naturally, the topic has been an area of intense and prolific research since the behaviour came to the notice of the GP research community, see [25], [29] and [1] for reviews of current bloat theories and methods of controlling or preventing it.

For some time, the accepted wisdom in the GP community was that smaller programs may generalise better. This *appeared* consistent with the general principle of Occam's razor, and its formalisation in the minimum description length (MDL) principle proposed by Grünwald [18] in 1997. However, in the last few years, the link between bloat and over-fitting has become controversial. Recent work by Vanneschi et al. [31] which set out to define the concepts of bloat, over fitting and complexity, and to investigate relationships that may exist between them, suggests that somewhat contrary to popular belief, these phenomena may be independent to some degree. Azad and Ryan [2] explained that a small GP tree is not necessarily simple: for example, $sin(x)$ is more complex than a larger GP tree encoding $x+x+x+x$. Related work [6] tentatively suggested that functional complexity, or the lack thereof, may play a more important role in generalisation than bloat does.

While this background suggests that the relationship between controlling size growth and improving generalisation is not straightforward, they are both desirable for GP. This paper reports on a use of bootstrapping with GP which, given the problems undertaken, delivers on both these objectives.

Table 1: GP Parameters

| Parameter | Value |
|---|---|
| Strategy | Steady State |
| Initialisation | Ramped half-and-half |
| Selection | Tournament |
| Tournament Size | 5 |
| Crossover | 90 |
| Mutation | 10 |
| Initial Min Depth | 1 |
| Initial Max Depth | 8 |
| Max Depth | 20 |
| Function Set | + - * / |
| ERC | -5 to +5 |
| Population | 2000 |
| Max Gen | 200 |
| Bootstrap Replicates | 50 |

## 4. DETAILS OF PROPOSED TECHNIQUE

In this paper, we propose to use bootstrapping to estimate the sensitivity of the evolved models to the training data set. We use this sensitivity as a measure of predictive ability of the evolving solutions on the unseen test data. The proposed approach is different from previous approaches discussed in section 2 that formed ensembles from the GP individuals trained over *different* bootstrap replicates

derived out of a given training set. Thus, in those approaches different GP runs were treated to different subsets of the training set. In contrast, in this paper, we use the *entire* training set for every individual in each run. However, after we score an individual on the training set, we also score the individual under consideration on *n* bootstrap replicates, each derived from the original training set by random resampling *with replacement*. Note, that the size of each bootstrap replicate is the same as that of the original training set; hence, some data points may repeat in a bootstrap replicate while others may not appear at all. The Bootstrap Standard Error (BSE) then is the standard deviation of these *n* scores which gives an estimate of the confidence in the original score on the entire training data set. The lower the BSE, the less sensitive is the evolved model to slight variations in the training set; such a model is more likely to generalise to unseen data.

We propose to minimise BSE along with the percentage error rate on the training data for each individual in the population. To achieve this we compute the fitness of an individual as a product of the corresponding percentage error rate and BSE. The percentage error rate is the percentage classification error of the individual on the overall training set, whereas the BSE is the standard deviation of percentage classification errors on *n* bootstrap replicates. For this preliminary study we have chosen to use 50 bootstrap replicates to get a reliable estimate of BSE [9].

# 5. EXPERIMENTS

GP and dataset parameters used for the experiments are detailed in tables 1 and 2 respectively. We have used four well known datasets obtained from the UCI Machine Learning database [12] and we undertook two hundred runs for each configuration and used identical random seeds for each set of experiments. For each task, at every generation, we record percentage error rate for both training and test data for reporting and comparison purposes. In addition, during each run, details of the performance of the best-so-far individual were captured. We used the Open Beagle Evolutionary Framework [14] for this study.

Table 2: Data Sets [12]

| Data Set | Features | Instances |
|---|---|---|
| Blood Transfusion (BT) | 5 | 768 |
| Bupa Liver Disorders (BUPA) | 7 | 345 |
| Habermans Survival (HS) | 4 | 306 |
| Wisconsin Brest Cancer (WBC) | 10 | 699 |

## 5.1 Bootstrap Configurations

For this preliminary investigation, we have experimented with three different bootstrap configurations in addition to a standard GP set-up. Details of the fitness function for each are outlined in table 3. For the standard GP configuration, the fitness measure used to drive the evolutionary process was simply the percentage of misclassified instances, called error rate from now on, whereas for the proposed Bootstrap method the fitness of each individual was calculated by multiplying the error rate by the bootstrap standard error (BSE) of the bootstrap estimates for that individual. After noting the results from standard GP and for GP with bootstrap (BS), we conducted two additional experiments to further investigate the efficacy of incorporating standard error inside the fitness function. The first set-up, termed *BootRandom (BSR)*, generates a random value from the *entire* range of BSE values observed with original bootstrap based runs. Thus, we introduce a noise within the range of the previously observed BSE, but apply it randomly, and investigate whether this noise also has an effect similar to BSE.

Table 3: Fitness Configurations

| Acronym | Description | Fitness Function |
|---|---|---|
| GP | Standard GP | Error % |
| BS | BootStrap | Error % * BSE |
| BSR | BootRandom | Error % * random value: range 0.01-0.061 |
| BSRT | BootTight | Error % * random value: range 0.01-BSE |

Next, in the final set-up called *BootTight(BSRT)*, we generate the BSE for each individual in the same way as for the BS configuration, but then we select a random value between 0.01 and the generated BSE to use in the fitness function, instead of the BSE itself, thus generating a tighter BSE value which will on average be smaller than the actual BSE. Using these two methods we verify if the original, *true* Bootstrap approach offers benefit over and above just introducing carefully crafted noise.

It may be informative to also compare with methods such as an ensemble based GP or multi-objective GP. However, in this preliminary, proof-of-concept, investigation we restrict ourselves to the methods described above.

# 6. RESULTS AND DISCUSSION

Figures 1 to 4 illustrate average percentage error rate on training and test data for each of the problems studied. Looking at these we can see that although overall on training data, standard GP performed better than the various BS methods, on the all important test data the BS method performs as least as well. Observing the graphs for average training and test fitness side by side, we can see that in the case of standard GP, the gap between training and test fitness is larger than with the BS configuration, and this gap is tending to widen as evolution progresses. This suggests that there is some over-fitting occurring with the standard approach and a stronger correlation in training and test set performances with BS methods.

In addition to the population average test scores, we also examined the test performance of the best-of-run *trained individuals* as shown in table 4. Here, we see that the BS method consistently outperformed GP on each of the four datasets. Tests for statistical significance revealed that the results were significant in all cases using a ninety-five percent confidence interval using a paired Student t-test and a non-parametric paired Wilcoxon signed rank test. Differences between the three different bootstrap methods were not statistically significant. In all cases, the size of the programs generated with standard GP was larger than that of those generated with other methods. Also, note that the best trained individual was on average discovered earlier in the evolutionary process using one of the bootstrap configurations.

Perhaps the most interesting (and unexpected) aspect of the results was a dramatic difference in average tree size, as illustrated in Fig. 5. Even though the experiments were constructed without any explicit growth constraining mechanism, the average tree size obtained when various bootstrap configurations were used was significantly smaller than that with standard GP. In particular, the BSR and BSRT configurations generated very small trees. *Without losing accuracy*, the generation of smaller program trees is a very desirable outcome, offering substantial savings in run times and significant improvements in comprehensibility, particularly when a simple function set is used, as is the case here.

In many standard GP implementations, without explicit bloat curtailment measures, it is often the case that when the system stops learning before evolution ceases, the tree structures continue

Figure 1: BT Average Training and Test Error Rates

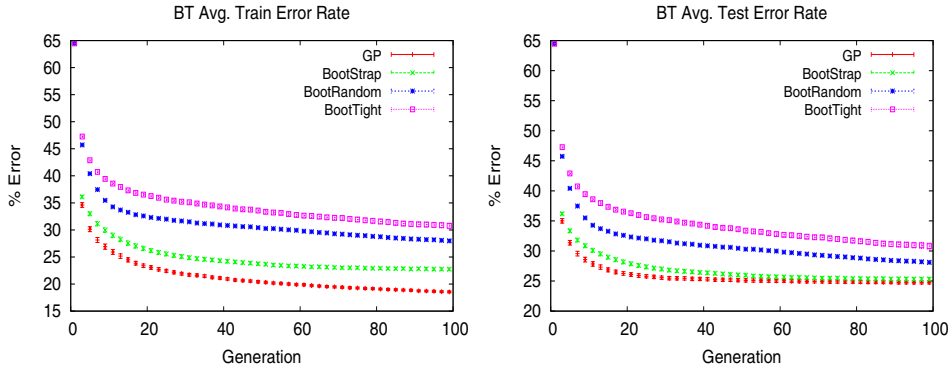



Figure 2: Bupa Average Training and Test Error Rates

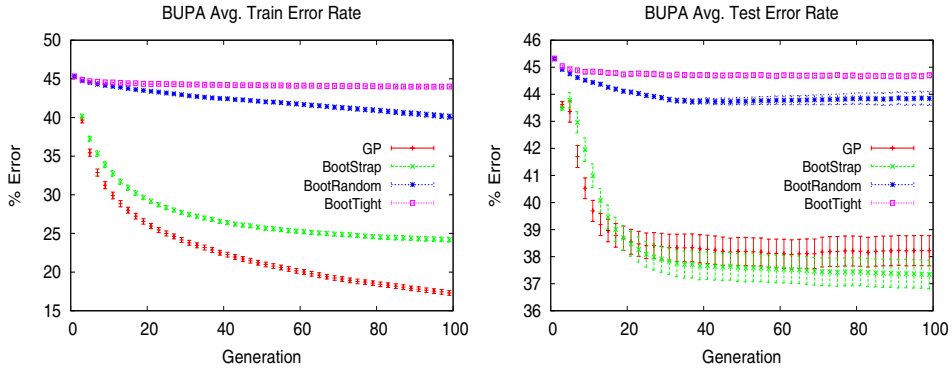



Figure 3: HS Average Training and Test Error Rates

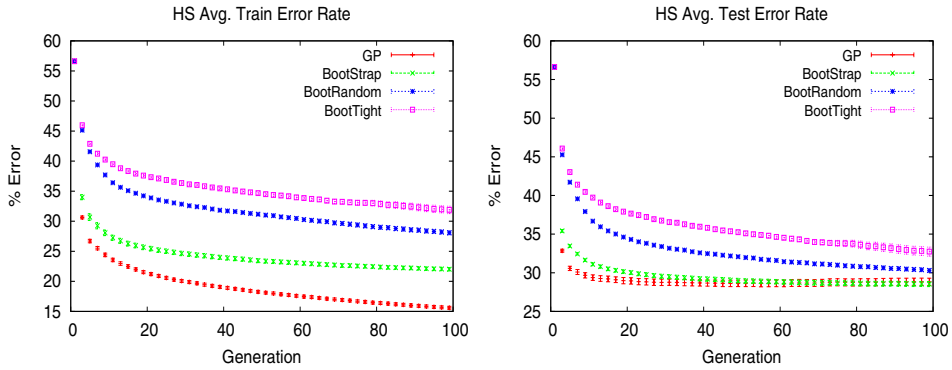



Figure 4: WBC Average Training and Test Error Rates

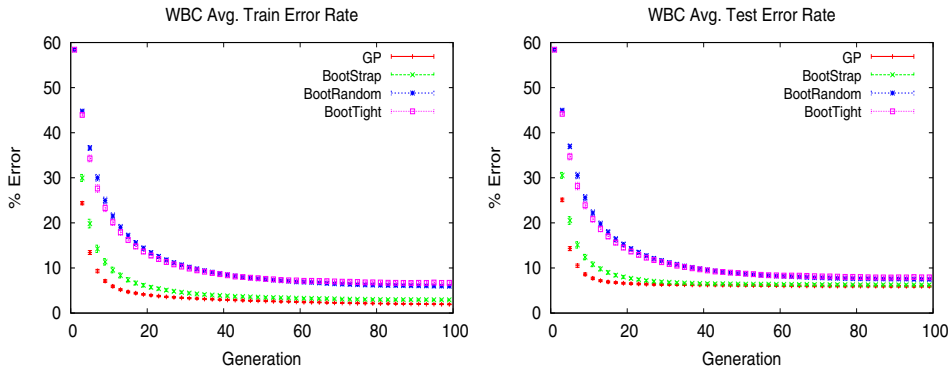


Table 4: Best of Run Individuals: All values are averaged over 200 Best-of-run trained Individuals for each task, for each configuration.

| Data | Method | Size | Train Err% | Test Err% | Best Test Err.% | Gen. |
|---|---|---|---|---|---|---|
| **BT** | GP | 941.82 | 14.38 | 23.21 | 19.50 | 173.18 |
| | BS | 342.05 | 18.42 | 22.18 | 19.21 | 98.20 |
| | BSRT | 136.32 | 18.63 | 22.00 | 19.31 | 73.47 |
| | BSR | 132.46 | 19.72 | 22.02 | 19.31 | 65.46 |
| **BUPA** | GP | 917.42 | 9.97 | 37.63 | 27.33 | 176.76 |
| | BS | 435.38 | 19.35 | 33.95 | 24.42 | 119.90 |
| | BSRT | 49.56 | 25.60 | 33.86 | 25.00 | 101.94 |
| | BSR | 99.83 | 24.91 | 35.49 | 26.75 | 129.84 |
| **HS** | GP | 1083.53 | 11.02 | 28.48 | 21.06 | 156.78 |
| | BS | 462.91 | 17.47 | 25.16 | 20.39 | 115.42 |
| | BSRT | 164.45 | 20.32 | 25.88 | 19.74 | 81.53 |
| | BSR | 205.40 | 20.23 | 25.35 | 19.74 | 97.99 |
| **WBC** | GP | 699.18 | 0.49 | 5.28 | 2.65 | 103.9 |
| | BS | 384.66 | 0.78 | 3.72 | 1.47 | 102.69 |
| | BSRT | 269.95 | 2.14 | 4.03 | 2.06 | 83.86 |
| | BSR | 237.46 | 2.14 | 4.26 | 2.06 | 120.73 |

to grow. Using bootstrap, we see that for the BS configuration, learning tails off somewhere between generations 40 and 60 and that this is matched by a corresponding dramatic slowdown in program growth.

In the case of standard GP, the significant extra program growth does not translate into better test accuracy. Without a detailed knowledge of the reasons for the greatly curtailed program size associated with the BS approach, we cannot be sure if there is a causal relationship between program size and learning outcomes or vice versa. A deeper understanding of the mechanisms involved is necessary, but one possible explanation of the reduced tree size is that the bootstrap standard error is likely to be small when an individual has either very high or very low classification accuracy, as it is consistently classifying or misclassifying. Note, that when the classification threshold is 0, the individual with a very low accuracy is quite good at separating the classes: simply inverting the polarity of outputs makes it an excellent solution [11].

if it were to be the case that both highly fit and highly unfit individuals would tend to have smaller programs, this may have the effect of disproportionately rewarding small programs at the expense of *middle of the road* performers that have larger programs, so that larger programs may be eliminated from the population over time.

This explanation is also possible for the very small trees produced by the BootTight configuration, as the noise value applied instead of the BSE will tend to be smaller than the corresponding BSE, and will thus lead to an even greater disparity in relative fitnesses.

Figure 7 illustrates the frequency distribution of program sizes for the 200 best-of-run individuals on each of the four problems, for the standard GP and the BS respectively. Here, we see that for the standard GP configuration, the distributions have roughly "normal" shape, whereas for the BS configuration, the general trend is skewed towards the left, with a higher frequency of programs in the lower range, with the exception of the WBC data. Therefore, not only the BS programs are smaller than GP programs on average but also a greater proportion of BS programs are clustered around the lower end of the distribution, showing a strong bias of BS towards small solutions.

If the suggested explanation for the smaller trees produced with bootstrap were accurate, we would expect to see a reduction in diversity over time as programs with average fitness are squeezed out of the population. Accordingly we captured various information on population diversity during evolution. We recorded measures of *genotypic* , *phenotypic* and *functional* diversity, roughly corresponding to the notions of *structural*, *behavioural* and *fitness* diversity outlined by Jackson in [21].

In measuring genotypic diversity, we compared the individual tree structures and recorded the number of unique trees in each generation. For phenotypic diversity, as each individual was evaluated on training instances, program output for each instance was concatenated in a string. At the end of each generation, the individual strings were compared for uniqueness and the number of unique strings was recorded. The measure of functional diversity captured is simply the number of unique fitness values in each generation. While each measure taken in isolation provides a coarse and not very insightful indication of population diversity, taken together they provide a useful guideline.

For both genotypic and functional diversity, similar values were achieved across the various configurations for all of the problems undertaken: genotypic diversity of between 90 and 100% was maintained throughout evolution, whereas values between 40 and 60% were typical for functional diversity. There was a greater variety in phenotypic diversity when the different configurations were employed. The percentage of unique phenotypes in the population during evolution is captured in Fig. 6.

It is usually recommended that a high level of diversity is maintained in the population, or at least, that potentially catastrophic losses in diversity are avoided [27]. For our experiments, standard GP maintained a high level of phenotypic diversity throughout, whereas the diversity of the BS populations fell off steadily but to differing degrees depending on the problem undertaken. However, this loss in diversity is clearly not a disadvantage for the BS method as it seems that this method quickly converges to solutions that generalise better.

Table 5: Average training, test error correlated with average tree size, GP and BootStrap configurations.

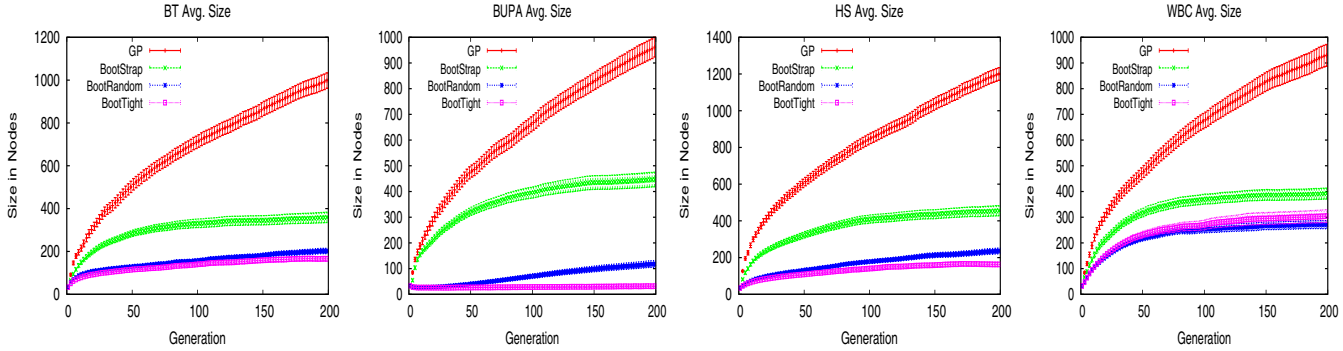
| Data | GPTrain | BSTrain | GPTest | BSTest |
|---|---|---|---|---|
| **BT** | -0.75 | -0.83 | -0.51 | -0.74 |
| **BUPA** | -0.94 | -0.99 | -0.56 | -0.83 |
| **HS** | -0.84 | -0.84 | -0.28 | -0.74 |
| **WBC** | -0.53 | -0.75 | -0.42 | -0.68 |

## 6.1 Size Fitness Correlation

Looking at the plots for program growth and fitness in figures 1 to 5 we see that both of these level off at approximately the same point in evolution with the BS set-up. Accordingly, we carried out some statistical tests with the GP and BS run data to investigate if there was a difference in correlation between program growth and fitness. We measured Pearson's product moment correlation between the average fitness (train and test) and average tree size at each generation across all 200 runs. The results shown in Table 5 indicate that for training fitness, in three out of four cases there was a stronger correlation when the BS method was used, and for two of these it was appreciably stronger. The correlation for test fitness was much stronger for all of the datasets.

As program growth is seen to taper off at approximately the same time as accuracy on test data when using BS, there may be a possi-

Figure 5: Average Tree Size in Nodes




bility that a consistent reduction in the rate of growth could be used as an effective early stopping measure, leading to significant time savings, potentially reducing over-fitting, and mitigating the need to guess an appropriate terminating generation.

In the context of our earlier stated goal in Section 3 of developing classifiers whose training performance could provide a more accurate indication of behaviour on unseen data: it seems that (for the problems undertaken), training scores achieved using the BS method provide a better indication of the generalisation ability of the programs, as the difference between these outcomes is smaller than when standard GP is used. For the latter experiments, there is clear over-training on both the BUPA and HS datasets, each of which produce test scores significantly worse than the average training scores, whereas for the BS experiments the average best test scores are at least as good as the average training scores for all of the datasets except BUPA.

If we accept the earlier definition of bloat in [27] as *program growth without (significant) return in terms of fitness*, we would argue the harmonious tapering off of growth and fitness, strong negative size/fitness correlation and dramatically smaller trees, is evidence that the BS method produces less bloated solutions than standard GP, without any compromise on test accuracy, and that these solutions may also be less likely to over-fit to the same degree.

## 6.2 Comparison with other Methods

In this section we detail results obtained using a variety of machine learning algorithms to classify the four datasets with the Weka Machine learning tool [19]. The algorithms that we have tested are detailed in Table 6.

Table 6: Machine Learning Algorithms

| Method | Acronym | Description |
|---|---|---|
| J48 | J48 | Implementation of C4.5 decision tree algorithm |
| Random Forest | RF | Ensemble decision tree classifier |
| libSVM | SV | Support vector machine |
| Multi Layer Perceptron | ML | Feed-forward ANN |
| Naive Bayes | NB | Probabilistic classifier using Bayes' theorem |
| Logistic | LG | Multinomial Logistic Regression |
| AdaBoostM1 | AB | AdaBoost with base classifier REPTree |
| Bagging | BG | Bagging with base classifier fast REPTree |

The performance of these algorithms on test data, is compared with that of the BS and GP test scores of best-of-run individuals averaged over two hundred runs. For compatibility the results are displayed as % Classification Accuracy, that is, 100 - classification error%. Where different results may be achieved for independent runs of the Weka algorithms, we averaged results over 200 runs. The comparative results can be seen in Table 7 and demonstrate that results obtained using BS are very competitive on all of the problems. To gain a clearer insight as to which method performed best overall we carried out the non parametric Friedman test [13] which is regarded as a suitable test for the empirical comparison of the performance of different algorithms [16]. Results indicated that the best performing algorithm in terms of test accuracy was Bagging closely followed by LG and BS. The relatively good results of the bagging approaches are likely to have been strongly influenced by the underlying base classifier, which was the Weka REPTree algorithm. This was the best performing of the available tree based methods when combined with either the AdaBoostM1 or bagging algorithms.

Table 7: Performance comparison of BS and GP with Machine Learning algorithms in Table 6

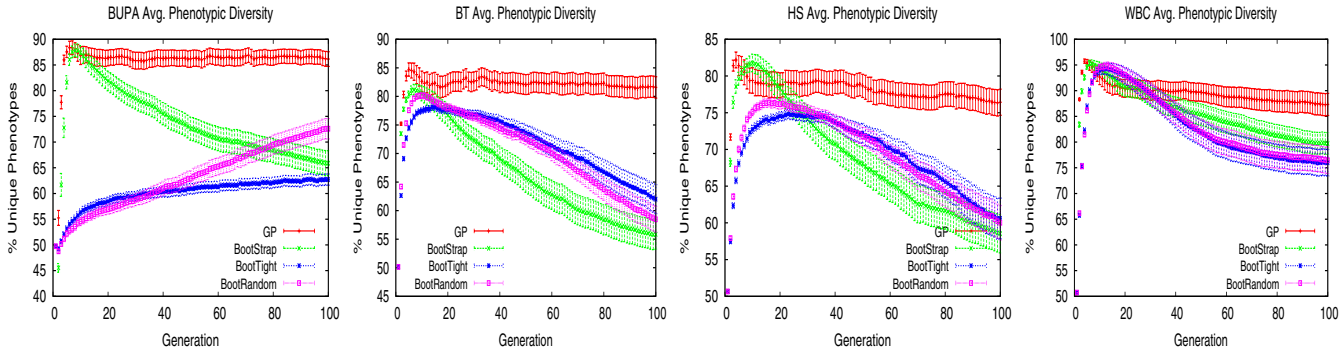
| Method | BT | Bupa | HS | WBC |
|---|---|---|---|---|
| BS | 77.82 | 66.05 | 74.84 | 96.28 |
| GP | 76.79 | 62.96 | 71.52 | 94.72 |
| J48 | 75.70 | 63.95 | 74.36 | 95.15 |
| RF | 73.89 | 65.33 | 65.40 | 96.40 |
| SVM | 74.09 | 64.01 | 71.50 | **96.86** |
| MLP | 77.66 | 64.19 | 74.26 | 95.81 |
| NB | 71.50 | 59.30 | **77.89** | 91.02 |
| AB | 57.41 | 66.48 | 72.68 | 95.54 |
| BG | **78.46** | **67.86** | 76.23 | 96.15 |
| LG | 77.29 | 66.88 | 77.41 | 96.03 |

## 6.3 Conclusions and Future Work

In this paper we use Bootstrap standard error as a measure of sensitivity of evolving classifiers to the training data. We minimise both the standard error and training classification error and find that this approach improves generalisation to unseen data. Rather surprisingly, with this approach we get trees that are much smaller in size than those with standard GP. We find that when using bootstrap, there is a strong negative correlation between fitness and sizes of the individual. This surprising discovery shows that using bootstrap can consistently produce small and reliable programs which is a desirable quality in any machine learning algorithm.

The current work is a preliminary investigation dealing solely with Binary Classification problems and comparison with a basic GP configuration. As such it is not possible to draw general conclusions regarding the success of the technique in terms of generalization and bloat in GP beyond this scope. In the future we would like to evaluate the BS approach on some new and more difficult problems: of particular interest would be tasks which can readily be shown to over-fit using a standard approach.

Figure 6: Phenotypic Diversity




Further experiments and analysis are required in order to gain a deeper understanding on the reasons for the observed constrained program growth and to learn more about the structure and evolution of the evolved programs. Finally, we propose to investigate the possibility of using the deceleration of program growth associated with our application of bootstrapping as an early stopping mechanism.

### *Acknowledgements*

This work has been supported by the Science Foundation of Ireland. Grant No. 10/IN.1/I3031.

## 7. REFERENCES

[1] Eva Alfaro-Cid, JJ Merelo, F Fernández de Vega, Anna I Esparcia-Alcazar, and Ken Sharman. Bloat control operators and diversity in genetic programming: A comparative study. *Evolutionary Computation*, 18(2):305–332, 2010.

[2] R. Muhammad Atif Azad and Conor Ryan. Abstract functions and lifetime learning in genetic programming for symbolic regression. In Juergen Branke et al. editors, *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 893–900, Portland, Oregon, USA, 7-11 July 2010. ACM.

[3] R. Muhammad Atif Azad and Conor Ryan. Variance based selection to improve test set performance in genetic programming. In Natalio Krasnogor et al. editors, *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1315–1322, Dublin, Ireland, 12-16 July 2011. ACM.

[4] Leo Breiman and Leo Breiman. Bagging predictors. In *Machine Learning*, pages 123–140, 1996.

[5] Mauro Castelli, Luca Manzoni, Sara Silva, and Leonardo Vanneschi. A comparison of the generalization ability of different genetic programming frameworks. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, Barcelona, Spain, 18-23 July 2010. IEEE Press.

[6] Mauro Castelli, Luca Manzoni, Sara Silva, and Leonardo Vanneschi. A quantitative study of learning and generalization in genetic programming. In Sara Silva et al. editors, *Proceedings of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 25–36, Turin, Italy, 27-29 April 2011. Springer Verlag.

[7] Dan Costelloe and Conor Ryan. On improving generalisation in genetic programming. In Leonardo Vanneschi et al. editors, *Proceedings of the 12th European Conference on Genetic Programming, EuroGP 2009*, volume 5481 of *LNCS*, pages 61–72, Tuebingen, April 15-17 2009. Springer.

[8] John Doucette and MalcolmI. Heywood. GP classification under imbalanced data sets: Active sub-sampling and auc approximation. In Michael O'Neill et al. editors, *Genetic Programming*, volume 4971 of *LNCS*, pages 266–277. Springer Berlin Heidelberg, 2008.

[9] B. Efron and R.J. Tibshirani. *An Introduction to the Bootstrap*. Monographs on Statistics and Applied Probability. Taylor & Francis, 1994.

[10] Jeannie Fitzgerald and Conor Ryan. Validation sets for evolutionary curtailment with improved generalisation. In *ICHIT (1)*, pages 282–289, 2011.

[11] Jeannie Fitzgerald and Conor Ryan. Exploring boundaries: optimising individual class boundaries for binary classification problem. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 743–750, New York, NY, USA, 2012. ACM.

[12] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[13] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of American Statistical Assoc.*, 32(200):675–701, 1937.

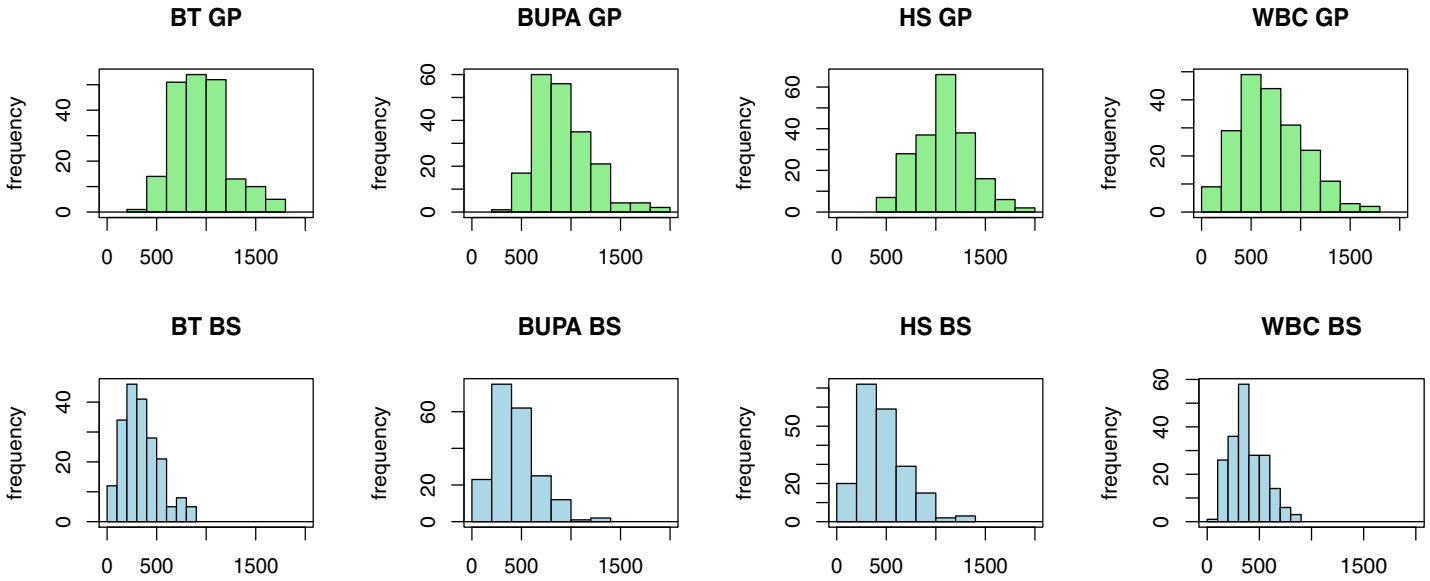
[14] Christian Gagné and Marc Parizeau. Open beagle: A new c++ evolutionary computation framework. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '02, pages 888–, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

[15] Christian Gagné, Marc Schoenauer, Marc Parizeau, and Marco Tomassini. Genetic programming, validation sets, and parsimony pressure. In Pierre Collet et al. editors, *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *LNCS*, pages 109–120, Budapest, Hungary, 10 - 12 April 2006. Springer.

[16] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. An experimental study based on Friedman's test of some local search techniques for planning. pages 59–68.

[17] Ivo Goncalves, Sara Silva, Joana B. Melo, and Joao M. B. Carreiras. Random sampling technique for overfitting control in genetic programming. In Alberto Moraglio et al. editors, *Proceedings of the 15th European Conference on Genetic Programming, EuroGP 2012*, volume 7244 of *LNCS*, pages 218–229, Malaga, Spain, 11-13 April 2012. Springer Verlag.

Figure 7: Size Frequency Distribution of Best of Run Individuals GP and BS Configurations





[18] Peter Grünwald. The minimum description length principle and non-deductive inference. In *Proceedings of the IJCAI Workshop on Abduction and Induction in*, page http://www.mpeg.org/, 1997.

[19] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[20] Hitoshi Iba. Bagging, boosting, and bloating in genetic programming. In Wolfgang Banzhaf et al. editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1053–1060, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[21] David Jackson. Promoting phenotypic diversity in genetic programming. In Robert Schaefer et al. editors, *PPSN 2010 11th International Conference on Parallel Problem Solving From Nature*, volume 6239 of *LNCS*, pages 472–481, Krakow, Poland, 11-15 September 2010. Springer.

[22] Maarten Keijzer and Vladan Babovic. Genetic programming, ensemble methods and the bias/variance tradeoff - introductory investigations. In Riccardo Poli et al. editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 76–90, Edinburgh, 15-16 April 2000. Springer-Verlag.

[23] J. R. Koza. Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical report, 1990.

[24] Ibrahim Kushchu. Genetic programming and evolutionary generalization. *IEEE Transactions on Evolutionary Computation*, 6(5):431–442, October 2002.

[25] Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, 2006.

[26] E. Howard N. Oakley. Genetic programming, the reflection of chaos, and the bootstrap: Towards a useful test for chaos. In John R. Koza et al. editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 175–181, Stanford University, CA, USA, 28–31 July 1996. MIT Press.

[27] Riccardo Poli, W B Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Lulu.com, March 2008.

[28] Uy Nguyen Quang, Thi Hien Nguyen, Xuan Hoai Nguyen, and Michael O'Neill. Improving the generalisation ability of genetic programming with semantic similarity based crossover. In Anna Isabel Esparcia-Alcazar et al. editors, *Proceedings of the 13th European Conference on Genetic Programming, EuroGP 2010*, volume 6021 of *LNCS*, pages 184–195, Istanbul, 7-9 April 2010. Springer.

[29] Sara Silva and Ernesto Costa. Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genetic Programming and Evolvable Machines*, 10(2):141–179, 2009.

[30] Dong Song, Malcolm I. Heywood, and A. Nur Zincir-Heywood. Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Trans. Evolutionary Computation*, 9(3):225–239, 2005.

[31] Leonardo Vanneschi, Mauro Castelli, and Sara Silva. Measuring bloat, overfitting and functional complexity in genetic programming. In Juergen Branke et al. editors, *GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 877–884, Portland, Oregon, USA, 7-11 July 2010. ACM.