A Preliminary Investigation of Overfitting in Evolutionary Driven Model Induction: Implications for Financial Modelling

Clíodhna Tuite, Alexandros Agapitos, Michael O'Neill, Anthony Brabazon

Financial Mathematics and Computation Cluster
Natural Computing Research and Applications Group
Complex and Adaptive Systems Laboratory
University College Dublin, Ireland

Abstract

This paper investigates the effects of early stopping as a method to counteract overfitting in evolutionary data modelling using Genetic Programming. Early stopping has been proposed as a method to avoid model overtraining, which has been shown to lead to a significant degradation of out-of-sample performance. If we assume some sort of performance metric maximisation, the most widely used early training stopping criterion is the moment within the learning process that an unbiased estimate of the performance of the model begins to decrease after a strictly monotonic increase through the earlier learning iterations. We are conducting an initial investigation on the effects of early stopping in the performance of Genetic Programming in symbolic regression and financial modelling. Empirical results suggest that early stopping using the above criterion increases the extrapolation abilities of symbolic regression models, but is by no means the optimal training-stopping criterion in the case of a real-world financial dataset.

1 Introduction

Overfitting is a commonly studied problem which arises in machine learning techniques such as Genetic Programming. A model is described as overfitting the training data if, despite having a high fit on the training examples, there

exists another model which has better fitness on the data as a whole, despite not fitting the training data as well [9]. There are different reasons why overfitting can occur. The existence of noise in training samples can cause a model to be fit to the data which is more complex than the true underlying model [14]. For symbolic regression, an example would be fitting a high order polynomial to noisy data, which happens to pass through all training points, when the true function is in fact a lower order polynomial. Another cause of overfitting is bias in the training data. Overfitting is more likely to occur when the training sample size is small. The more data available to train on, the more likely we are to discover the true underlying model, and the less likely we are to settle on a spurious result. Overfitting is also more likely to occur in the presence of complexity. Complex models (for example symbolic regressions of multiple explanatory variables) are more likely to induce overfitting. Learning algorithms that are run for a long time are also more likely to trigger overfitting, than if they had been run for a shorter time period [1].

This paper aims to begin to explore the issue of overfitting in grammar-based Genetic Programming [8], and provides case studies of overfitting in three symbolic regression problems, and a financial modelling example drawn from an instance of credit risk classification.

2 Model Induction

The underlying data generating process is unknown in many real-world financial applications. Hence, the task is often to deduce or "recover" an underlying model from the data. This usually isn't an easy task since both the model structure and associated parameters must be uncovered. Most theoretical financial asset pricing models make strong assumptions which are often not satisfied in real-world asset markets. They are therefore good candidates for the application of model induction tools, such as Grammatical Evolution [12, 6], which are used to recover the underlying data generating processes [3].

Of course to use a model induction method effectively, that is, to ensure that the evolved models generalise beyond the training dataset, we must pay attention to overfitting. This study aims to highlight this important open issue in the field of Genetic Programming [13] and its implications for financial modelling.

3 Background

3.1 Model Generalisation

A crucial aspect of data-driven modelling is related to model generalisation, and many financial applications of evolutionary methods do not apply techniques to minimise overfitting. Model generalisation concerns the ability of the induced model to correctly represent the underlying structure of the data so as to make accurate predictions when presented with new data from the problem domain. Unfortunately, data-mining methodologies that iteratively refine a model on a set of training instances, as is the case of evolutionary methods, inherently suffer from model overfitting; they produce solutions with poor generalisation abilities. There has been a large amount of statistics and sibling machine learning methodologies to counteract the phenomenon of overfitting and produce models with competent out-of-sample performance.

3.2 Model Overtraining Avoidance through Early Stopping

A well-exercised technique for promoting the generalisation of an induced model is the procedure of *early training stopping* [16, 9, 7]. For most learning algorithms the training error decreases monotonically during training. If an independent validation dataset is used to measure the model's accuracy on unseen data, the validation error tends also to decrease in step with the training error as the model gradually approximates the underlying function. However, it is very often the case that the training data contain spurious and misleading regularities due to sampling. In the later stages of the training process, the model begins to exploit these idiosyncrasies in the training data and the validation error tends to increase again while the training error continues to decrease. This example of overfitting is described in [5]. One approach to avoid overfitting is to use the independent validation dataset as part of a heuristic that dictates the halting of the training process at the first minimum of the validation error. Under such a regime, the learner is trained using the training instances, however, in each learning iteration it is evaluated for both training and validation accuracy. Typically, the error on the validation set decreases along with the training error, but then tends to increase, an indication that the model may be overfitting the training instances, suggesting that the training phase should be stopped. It has been shown that halting the training phase before a minimum of the training error has been reached, represents a way of limiting the complexity of the induced model [2].

3.3 Grammatical Evolution: A Brief Introduction

In Grammatical Evolution [12, 6], the process of evolution first involves the generation of a population of randomly generated binary (or integer) strings, the genotype. In the case of binary genomes, each set of B bits (where traditionally B=8) is converted into its equivalent integer representation. These integer strings are then mapped to a phenotype, or high-level program or solution, using a grammar, which encompasses domain knowledge about the nature of the solution. Therefore, a GE genome effectively contains the instructions of how to build a sentence in the language specified by the input grammar. Grammatical Evolution is a form of what is known as grammar-based Genetic Programming [8], and has been applied to a broad range of problems, including many successful examples in financial modelling [4].

The grammar used in the experiments we performed can be found in Fig. 1. The grammar is composed of non-terminal and terminal symbols. Terminals (for example arithmetic operators) appear in the solution, whereas non-terminals can be further expanded into terminals and non-terminals. Here we can see that knowledge of the solution (that it will be constructed from arithmetic operators, mathematical functions, variables and constants) is encoded in the grammar. The mapping process involves the use of an integer from the genotype to choose a rule from the production rule currently being mapped. This process proceeds as follows. The first integer from the genotype is divided by the number of rules in the start symbol (`<expr>` in our example). The remainder from this division is used to select a rule from the grammar (for example, if the first integer was 8, the result of dividing 8 by the number of choices available for the `<expr>` production rule, which is 5, would result in the choice of the third rule - which is `<pre-op>(<expr>)`. The next integer in the genotype would then be used in the same way to map between `<pre-op>` and one of its constituent rules, and the third integer in the genotype would be used to map between `<expr>` and one of its constituent rules. This process continues until either all integers in the genotype have been used up, or our mapping process has resulted in the production of a phenotype (that is a structure comprised of only terminal symbols) [12].

```
<prog> ::= <expr>
<expr> ::= <expr> <op> <expr>  | ( <expr> <op> <expr> ) |
           <pre-op> ( <expr> ) | <protected-op> | <var>
<op> ::= + | * | -
<protected-op> ::= div( <expr>, <expr>)
<pre-op> ::= sin | cos | exp | inv | log
<var> ::= X | 1.0
```

Figure 1: Grammar used in Symbolic Regressions

4 Experimental Setup

4.1 Symbolic Regression

Grammatical Evolution was used to fit models to 3 symbolic regression problems. Equations 1 through 3 show the target functions. The training dataset was comprised of 10 randomly generated points. The test dataset (which was not used to train the model) was comprised of 20 randomly generated points, 10 of which were drawn from the same range as the training data (to test how well the model interpolates, and to serve as a proxy for a validation dataset, see Section 5.1), and 10 of which were drawn from outside the range from which the training data were drawn (to test how well the model extrapolates).

$$Y = 0.6X^3 + 5X^2 - 10X - 25 \tag{1}$$

Training dataset range: [-5, 5]. Test dataset ranges: [-10, 10].

$$Y = 0.3X \times \sin 2X \tag{2}$$

Training dataset range: [-1, 1]. Test dataset ranges: [-2, 2].

$$Y = \exp X - 2X \tag{3}$$

Training dataset range: [-2, 2]. Test dataset ranges: [-4, 4].

These functions and ranges were chosen so that the target function would be trained using a biased sample. The bias resulted from training in a range in which the target function closely resembled an alternative function. Over a wider range than that from which the training data was drawn, the target function looked dramatically different from this alternative (for example, function 2 looked very like a quadratic in the training range (see Fig. 7), but as can be seen, it is truly a sine function). In this way, we engineered a situation in which overfitting was likely to take place. In each case, Grammatical Evolution was run on a population size of 100 individuals, for 50 generations, using Grammatical Evolution in Java [11]. The grammar used is shown in Fig. 1.

Fitness was evaluated by computing the mean squared error of the training points when evaluated on each individual (therefore the lower the fitness value, the better the evolved function fitted the training data).

$$MSE = \frac{\sum_{i=1}^{n} |targetY - phenotypeY|^2}{n} \tag{4}$$

4.2 The case of a financial dataset

We also test the early stopping approach to model overfitting on a real world financial dataset from the UCI Machine Learning repository [10]. The financial dataset represents a binary classification problem of categorising credit card applications between those which are approved or rejected. The dataset contains 690 number of instances, and each instance has 15 attributes.

We employed a grammar-based GP system to evolve non-linear discriminant functions that use the threshold value of zero to differentiate among the classes. The context-free grammar is represented below.

```
<prog> ::= <expr>
<expr> ::= <expr> <op> <expr> | <var>
<op> ::= + | * | - | /
<var> ::= instance attributes
```

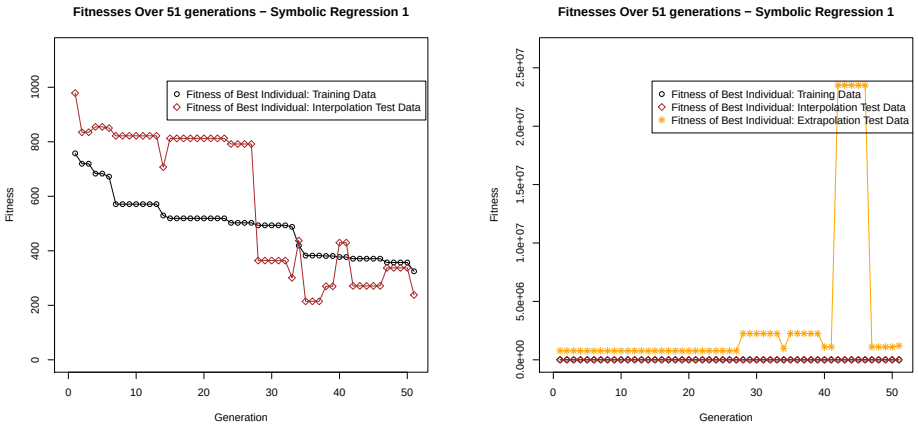
Figure 2: Grammar used in the financial credit classification problem.

The GP algorithm employs a panmictic, generational, elitist genetic algorithm. The algorithm uses tournament selection with a tournament size of 7. The population size is set to 500 individuals, and the number of generations to 100. Ramped-half-and-half tree creation with a maximum depth of 6 is used to perform a random sampling of DTs during run initialisation. Throughout evolution, expression-trees are allowed to grow up to depth of 15. The evolutionary search combines standard subtree crossover with subtree mutation; a probability governing the application of each, set to 0.6 in favour of subtree crossover. We used the classification accuracy (CA) as the fitness function, but in order to convert it to a minimisation problem we assigned fitness using $1.0 - CA$. We split the original dataset into two random equally-sized subsamples with equal distribution of classes, serving as the training and validation (out-of-sample) datasets.

5 Results and Discussion

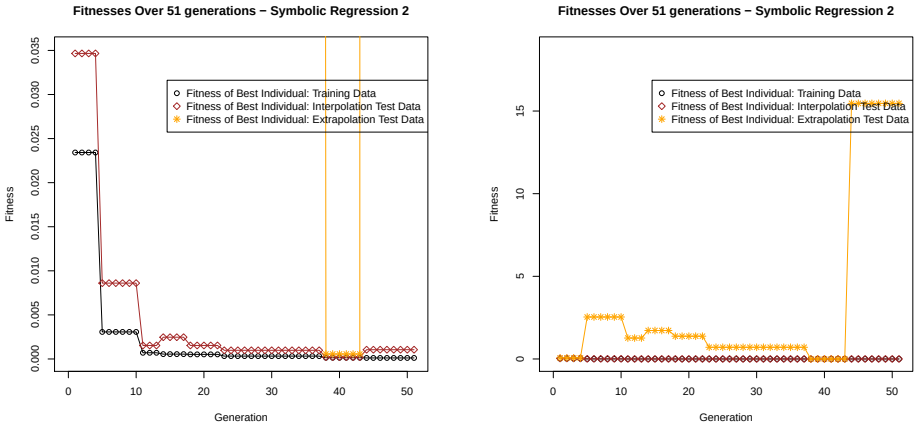
5.1 Symbolic Regression

Figs. 3(a) through 6(b) are plots of the fitness of the best individual at each generation as evaluated on the training data, against the fitness of the best individual at each generation as evaluated on the test datasets, for four illustrative runs - one run each of target functions 1 and 3, and two runs of target function 2. Table 1 contains details on the fitness as evaluated on the test dataset, for 9 runs. It shows that stopping evolution before the specified number of generations had elapsed, would have led to the model extrapolating better beyond the range in which it was trained.



(a) (b)

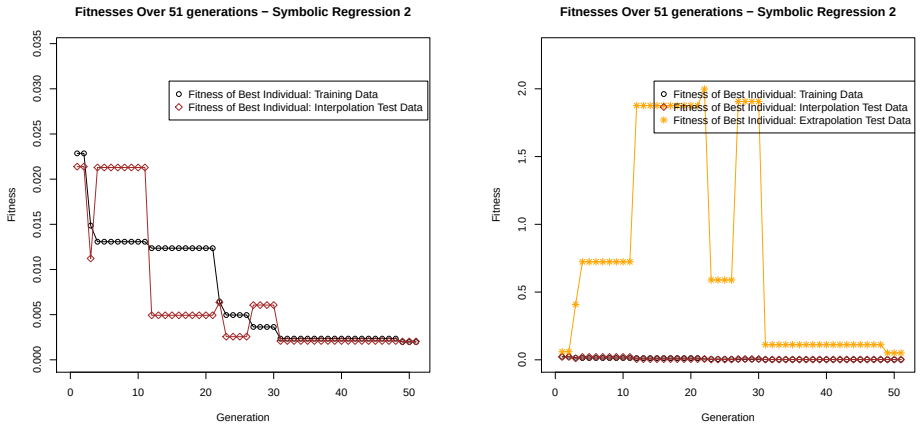
Figure 3: Target Function 1



(a) (b)

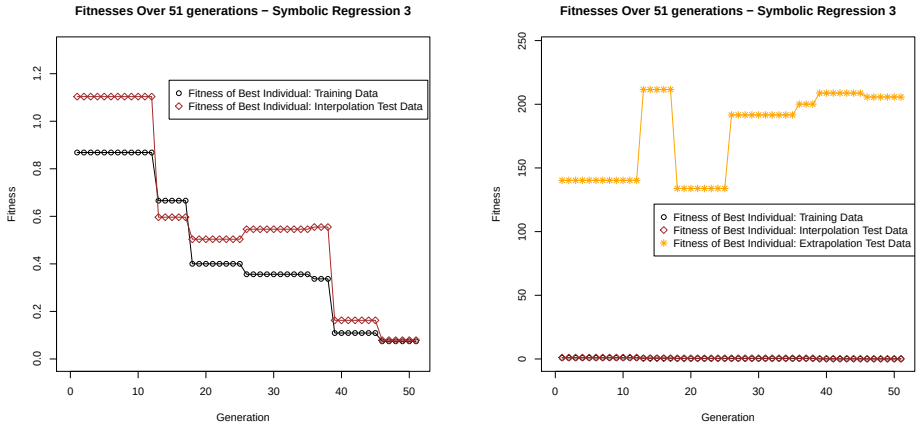
Figure 4: Target Function 2, Example 1

Early stopping has been described in Section 3.2. The validation dataset is not used to train the model, but instead is used to test the fitness of the model every once in a while (for example each generation, or at five generation intervals). If the fitness of the best individual as evaluated on the validation dataset disimproves, this is taken as an indication that the evolved model is overfitting the data, and evolution is stopped. (Test data is used as before to evaluate the fitness of the evolved model on out-of-sample data, after evolution has terminated, either prematurely (if early stopping has been deemed necessary), or after the specified number of generations has elapsed.)



(a) (b)

Figure 5: Target Function 2, Example 2



(a) (b)

Figure 6: Target Function 3

Since we explicitly chose target functions and ranges with an inherent bias, these symbolic regressions triggered overfitting, as expected. Table 1 shows the generation at which the fitness, as evaluated on the part of the test dataset used to measure the ability of the evolved model to interpolate in the training range (henceforth referred to as interpolation fitness), first disimproved. In 8 of the 9 runs described, the fitness as evaluated on the part of the test dataset used to measure the ability of the evolved model to extrapolate beyond the training range (henceforth referred to as extrapolation fitness), was better the generation immediately before the interpolation fitness first disimproved, than at the end of the run. Had we stopped evolu-

Table 1: Interpolation, Extrapolation fitnesses - Generations at which deteriorations took place

Target Function Number	1	1	1
Generation Interpolation Fitness First Disimproved (GIFFD)	4	10	4
Interpolation Fitness Better Generation GIFFD-1, or End of Run?	End of Run	End of Run	GIFFD-1
Extrapolation Fitness Better Generation GIFFD-1, or End of Run?	GIFFD-1	GIFFD-1	GIFFD-1
Best stopping point (Generation(s) of Lowest Extrapolation Fitness)?	14	7 - GIFFD-1	12
Target Function Number	2	2	2
Generation Interpolation Fitness First Disimproved (GIFFD)	14	3	4
Interpolation Fitness Better Generation GIFFD-1, or End of Run?	End of Run	End of Run	End of Run
Extrapolation Fitness Better Generation GIFFD-1, or End of Run?	GIFFD-1	End of Run	GIFFD-1
Best stopping point (Generation(s) of Lowest Extrapolation Fitness)?	38 - 43	49 - End of Run	5 - 8
Target Function Number	3	3	3
Generation Interpolation Fitness First Disimproved (GIFFD)	26	7	19
Interpolation Fitness Better Generation GIFFD-1, or End of Run?	End of Run	End of Run	GIFFD-1
Extrapolation Fitness Better Generation GIFFD-1, or End of Run?	GIFFD-1	GIFFD-1	GIFFD-1
Best stopping point (Generation(s) of Lowest Extrapolation Fitness)?	18 - GIFFD-1	7 - 10	8 - GIFFD-1

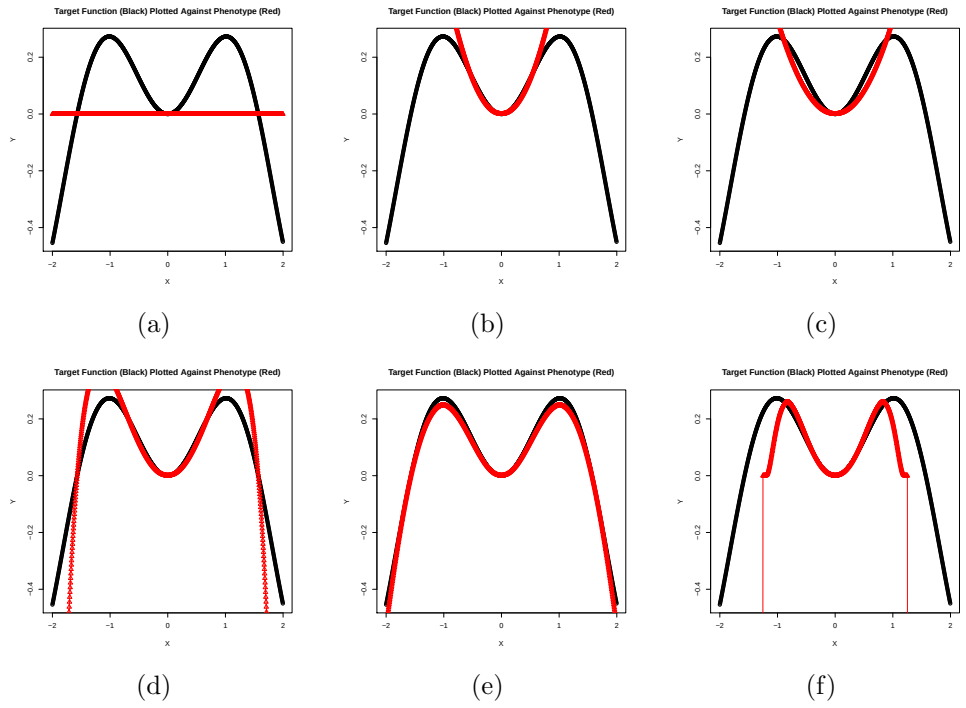


Figure 7: (a) Generation 1 (b) Generation 5 (c) Generation 11 (d) Generation 23 (e) Generation 38 (f) Generation 44

tion at this point, we would have produced a model that extrapolated better beyond the training range, than the model produced at the end of the run.

The data points from the test dataset drawn from the same range as the training dataset (and used to measure how well the evolved model is interpolating within the training range), can also be used as a proxy for a validation dataset. [15] show that when training artificial neural networks,

the first time the error on the validation set increases is not necessarily the best time to stop training, as the error on the validation set may increase and decrease after this first disimprovement. Such a pattern seems to exist in the runs we performed. In 5 of the 8 runs where early stopping would have made sense, the optimal generation at which to stop (the generation with the lowest extrapolation fitness value) came later than the generation at which the interpolation fitness first disimproved.

To give further insight into the evolutionary process that underlie the changes in fitness observed for the training and test data sets, the phenotype was plotted against the target function in the extrapolation range, at each generation. Fig. 7 shows a selection of these generational graphs for the first run of function 2.

Comparing Figs. 7 and 4(b), we can clearly see the correspondences between changes in the graphed phenotype over the generations, and changes in the fitness as evaluated on the extrapolation test data. Between generations 1 and 22, the extrapolation test fitness is either disimproving, or not improving by much. At generation 23 fitness improves significantly, and at generation 38, an extremely fit individual has been evolved, both with respect to the training and test set. The model extrapolates well. However, come generation 44, a much less fit function has been evolved. It's fitness on the training data has improved, but it's fitness on the extrapolation test data has drastically disimproved. If we look back at Fig. 4(b), we can clearly see both an extremely low value in the fitness on the extrapolation test data at generation 38, and an explosion in the value of the fitness on the extrapolation test data at generation 44.

5.2 Financial Dataset

We performed 100 independent evolutionary runs. Table 2 presents average performances of the best-of-generation individuals, on both training and validation sets, throughout the evolutionary process. Results suggest that there is no particular evidence of model overfitting; the validation performance curve monotonically decreases up until generation 80, at which point a slight degree of overtraining becomes apparent. This is evidenced by the average percentage change in the validation performance, which reaches a negative number between generations 80 and 90 (Table 3). The model performance in the case of early stopping at the generation that the validation error becomes a local minimum for the first time, is summarised in Table 4. It is apparent that stopping at approximately generation 6 results in a model with validation performance of 0.31, which is clearly not the optimal point at which to stop, given that the best validation performance is 0.27, attained by generation 80 (Table 2).

Overall, this empirical result suggests that early training stopping at the

Table 2: Training and Test Learning Curves for the classification problem. Averages of 100 evolutionary runs. Standard deviation in parentheses.

	Gen. 10	Gen. 20	Gen. 30	Gen. 40	Gen. 50
Training performance	0.25 (0.01)	0.24 (0.01)	0.23 (0.01)	0.22 (0.01)	0.22 (0.01)
Validation performance	0.29 (0.02)	0.29 (0.02)	0.28 (0.02)	0.28 (0.02)	0.28 (0.02)
	Gen. 60	Gen. 70	Gen. 80	Gen. 90	Gen. 100
Training performance	0.21 (0.01)	0.21 (0.01)	0.21 (0.01)	0.20 (0.01)	0.20 (0.01)
Validation performance	0.28 (0.02)	0.27 (0.02)	0.27 (0.02)	0.28 (0.02)	0.28 (0.02)

Table 3: Percentage change in Training and Testing performance. Averages of 100 evolutionary runs. Standard deviation in parentheses.

	Gen. 10-20	Gen. 20-30	Gen. 30-40	Gen. 40-50	Gen. 50-60
Training performance change (%)	5.2% (0.03)	3.7% (0.03)	2.7% (0.02)	1.7% (0.02)	1.7% (0.02)
Validation performance change (%)	2.0% (0.07)	1.9% (0.07)	0.7% (0.05)	0.6% (0.04)	0.2% (0.03)
	Gen. 60-70	Gen. 70-80	Gen. 80-90	Gen. 90-100	
Training performance change (%)	1.4% (0.01)	1.2% (0.01)	1.3% (0.01)	0.9% (0.01)	
Validation performance change (%)	0.2% (0.03)	0.3% (0.03)	-0.6% (0.03)	0.6% (0.03)	

first point where the validation error reaches a local minimum (assuming fitness minimisation) is by no means a reliable indication of overtraining. Future research needs to address the issue of early stopping with more sophisticated stopping criteria.

6 Conclusions and Future Work

In this study we set out to highlight a significant open issue in the field of Genetic Programming, namely generalisation of evolved solutions to unseen data, which has real world implications for all model induction methods, and can have serious financial implications when considered in the domain of financial modelling. Empirical investigations on four benchmark problems are undertaken. Three of the problems were drawn from the popular Genetic Programming domain of symbolic regression and the fourth problem was an instance of credit classification.

In summary the results illustrate the important role which the detection of overfitting during training can play, in order to improve the generalisation of the evolved models. What is also clear from these results is that further

Table 4: Performance statistics during early stopping. Averages of 100 evolutionary runs. Standard deviation in parentheses.

Early Stopping Generation	5.88 (6.07)
Early stopping training performance	0.27 (0.02)
Early stopping validation performance	0.31 (0.03)

lessons need to be drawn from the machine learning literature on effective early stopping strategies, and the myriad of other strategies which have been adopted to avoid overfitting. The results on both classes of problem domain investigated here demonstrate that early stopping could be an effective strategy to improve generalisation, however, following a naive early stopping heuristic can lead to stopping *too early*.

Acknowledgments

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant No. 08/SRC/FMC1389.

References

[1] L.A. Becker and M. Seshadri. Comprehensibility and overfitting avoidance in genetic programming for technical trading rules. *Worcester Polytechnic Institute, Computer Science Technical Report*, 2003.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1996.

[3] A. Brabazon, J. Dang, I. Dempsey, M. O'Neill, and D. Edelman. Natural computing in finance: a review. 2010.

[4] Anthony Brabazon and Michael O'Neill. *Biologically inspired algorithms for financial modelling*. Springer, 2006.

[5] Y Chauvin. Generalisation performance of overtrained back-propagation networks. In *EUROSIP Workshop*, pages 46–55, 1990.

[6] I. Dempsey, M. O'Neill, and A. Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments*. Springer Verlag, 2009.

[7] Richard Duda, Peter Hart, and David Stork. *Pattern Classification*. John Wiley and Sons, 2nd edition, 2001.

[8] R.I. Mckay, N.X. Hoai, P.A. Whigham, Y. Shan, and M. ONeill. Grammar-based Genetic Programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396, 2010.

[9] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[10] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998.

[11] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, and A. Brabazon. GEVA: grammatical evolution in Java. *ACM SIGEVOlution*, 3(2):17–22, 2008.

[12] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary automatic programming in an arbitrary language*. Springer Netherlands, 2003.

[13] Michael O'Neill, Leonardo Vanneschi, Steven Gustafson, and Wolfgang Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3-4):339–363, 2010.

[14] G. Paris, D. Robilliard, and C. Fonlupt. Exploring overfitting in genetic programming. In *Artificial Evolution*, pages 267–277. Springer, 2004.

[15] L. Prechelt. Early stopping-but when? *Neural Networks: Tricks of the trade*, pages 553–553, 1998.

[16] Warren S. Sarle. Stopped training and other remedies for overfitting. In *Proceedings of the 27th Symposium on the Interface of Computing Science and Statistics*, pages 352–360, 1995.