

Genetic Programming of Fuzzy Logic Production Rules with Application to Financial Trading.

Andrew N. Edmonds
Science in Finance Ltd
Haydon House, Station Rd , Woburn Sands, Bucks, MK17 8RX, UK.
ane@peoplebank.co.uk

D. Burkhardt & O. Adjei
University Of Luton, Park Sq. Luton Beds. UK

ABSTRACT

John Koza⁷ has demonstrated that a form of machine learning can be constructed by using the techniques of Genetic Programming using LISP statements. We describe here an extension to this principle using Fuzzy Logic sets and operations instead of LISP expressions. We show that Genetic programming can be used to generate trees of fuzzy logic statements, the evaluation of which optimise some external process, in our example financial trading. We also show that these trees can be simply converted to natural language rules, and that these rules are easily comprehended by a lay audience. This clarity of internal function can be compared to “Black Box” non-parametric modelling techniques such as Neural Networks. We then show that even with minimal data preparation the technique produces rules with good out of sample performance on a range of different financial instruments.

Introduction

In 1965 Lotfi Zadeh¹ first published a description and analysis of Fuzzy Logic. This is a true superset of Boolean Logic and permits the description of functions and processes with a degree of vagueness or uncertainty. Since then various workers in the field have extended the concept of rule based expert systems² to embrace Fuzzy Logic^{3,4,5} especially in the field of industrial control.

John Holland⁶ published his book on Genetic Algorithms in 1975. This described a methodology for optimisation of arbitrary functions using techniques gleaned from the processes of Darwinian evolution. This methodology is both robust in optimising noisy and non-linear functions, and powerful in that multiple solutions, where they exist, are automatically generated by the technique. In 1992 Koza⁷ published his extension to Holland’s work that used Genetic techniques to modify and mate a set of LISP statements. Koza used LISP because statements in this language can be easily converted to a parse-tree representation, and evaluation involved only compilation and provision of data. In Koza’s methodology a set of

parse trees are evaluated using some external test set and performance measure, and mated to form a new set with selection of pairs at random, but with a bias towards those with the highest performance. Mating, unlike Holland's crossover technique, is performed by cutting the parent parse trees at randomly selected points and swapping the pruned subtrees to produce offspring that are copied into the next generation. It has been found that the performance of succeeding generations improves, if hesitantly, and that a wide variety of problems can be solved by this technique. A great benefit of this methodology is that the model discovered, being coded in LISP, is immediately accessible to those who can bring themselves to learn that language.

Koza has demonstrated the re-discovery of various natural laws and even economic principles from raw data using this technique.

Implementation

In our implementation the parse trees contain *fuzzy operators* and *fuzzy sets*. We follow Koza's methodology in this area, but limit the combinations permitted under mating so that the trees generated could be converted back to production rules of the form:

if <Conditions> then <Crisp Classification>

where <Conditions> are:

<Conditions> *And* <Conditions>

<Conditions> *Or* <Conditions>

Not <Conditions>

<operand> is <Fuzzy Comparative Set> <operand>

<operand> is <Fuzzy Set>

Fuzzy Operators:

And is defined as the minimum operation on two variables

Or is defined as the maximum operation on two variables

Not is the inversion of a single variable (output = 1-input)

Fuzzy Sets

We have implemented fuzzy sets in two forms. The first is a *threshold* function of the type:

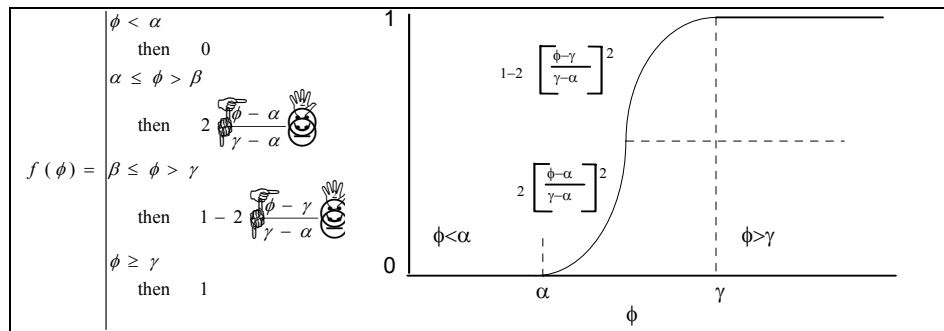


Figure 1, Threshold function

This is used to generate fuzzy sets with linguistic values such as *greater than*, *less than*, *much larger than*, *large*, *small* etc.. The two parameters a and g control the position and steepness of the threshold.

The second set type has a *range* function:

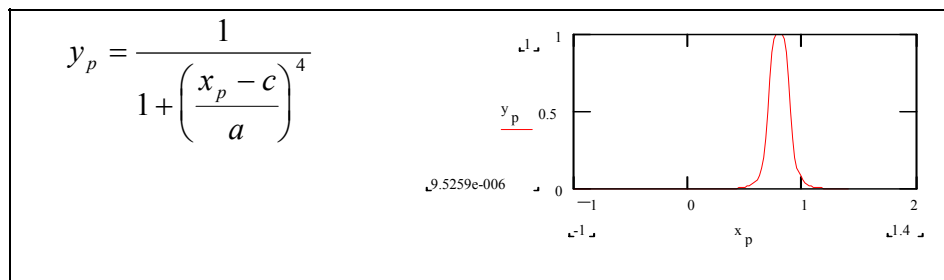


Figure 2, Range function

The two parameters a and c control the width and position of the fuzzy set respectively. The range function is used to generate fuzzy sets with linguistic values such as *close to*, *fairly close to*, *medium* etc..

The fuzzy sets can be used in two ways, either as *absolute sets* where the value of a variable is used as the input, or *comparative sets* where the difference between two variables is input to the set. The former has linguistic values such as *high*, *low*, *medium*, etc. the latter *close to*, *greater than*, *less than*, etc..

Genetic Programming of the Fuzzy Rule Sets

The initial population

The size of the initial population may be varied in our application, but we have found that Koza's suggested size of 500 provides a good selection of genetic material without requiring excessive processing time in the examples given here. Where more categories of input data are provided, considerably larger populations are required.

We generate rule sets using Koza's 'ramped half-and-half' method. In this, equal numbers of rules are generated for each depth between 2 and the maximum depth. Of these, half the trees are 'full', (i.e. all branches are full length) and the rest are 'grown' (i.e. branches may be different lengths). Our default maximum depth is 5. Inputs to a fuzzy link (*And, Or, Not*) may be other links or fuzzy sets. Inputs to fuzzy sets can only be terminals (leaf nodes).

The inputs to comparative fuzzy sets must be different, so 'if A is greater than A' is illegal. However, we have not, at present, disallowed deeper forms of contradiction or tautology, e.g. 'if A is greater than B or A is much greater than B'.

We exclude the expression '*Not Not*'.

Terminal nodes are selected randomly from the general pool of input values.

We normalised the data we supplied to the genetic programming system to the interval 0,1. Where inputs are of the same type and can be grouped, for instance prices, normalisation is performed over the group, thus preserving the interrelationships. This scaling enabled us to limit the fuzzy set parameters to the interval -1,1

The fuzzy set functions above were recast to give two parameters that controlled the centre or transition point of the set, and the rate of transition or crispness of the set.

In the initial generation the centre parameters were selected randomly with a uniform distribution. The crispness parameter was randomly chosen with a gaussian distribution centred on 0, and the result squared to give a single ended result.

Subsequent generations

Subsequent generations are produced by a roulette selection of tree pairs based upon their fitness calculated in the previous generation (see below for details of the fitness algorithm). The rules for legal sets are the same as those described above, but the maximum depth of created trees is greater (default 9).

After selecting a pair of trees, we randomly determine whether they will be passed directly to the next generation (probability 0.1) or mated together (probability 0.9).

Mating is done by randomly selecting crossover points anywhere on each tree, then swapping the cut limbs. This may involve a single node or almost the entire tree. When mating produces an illegal tree, as defined above, an alternate cross point is selected. If a pair of legal trees cannot be generated, then the parent trees are transferred to the next generation unchanged.

Elitism (where the best tree so far is always passed to the next generation unchanged) was employed for all the examples discussed.

Experiments

We are interested in the apparent ability of this technique to generate rules that optimise some external process that are both effective and easily understood. We have chosen the generation of financial trading rules as a test of this technique partly because of personal and corporate interest, but also because the making of money is the most widely accepted measure of effectiveness. Furthermore while financial time series are considered to be some of the hardest series to predict, there is strong evidence that they have the attributes of predictability⁸.

The process in this case to be optimised is the trading of financial instruments. We use as input the price history of the instrument of interest consisting of 500 daily samples. The samples contain the open price, high achieved during the day, low achieved, and closing price.

The other vital constituent is a simulation of trading. Financial trading is not a homogenous activity, in particular the time scale over which traders work varies dramatically. Because only daily data was initially available we chose to simulate trades with a minimum duration of 1 day. In the markets we looked at short trading was possible, i.e. as well as buying an instrument in the expectation of a price rise, one can also borrow an instrument in the expectation of a price drop, and thus make a profit on either direction of market movement.

Some instruments are traded with bid-ask spreads, this means that the selling price is higher than the buying price by some narrow margin which is a source of profit to the dealer, and there can be commission charges.

Wise traders use stops. These are prices at which a losing position is terminated. In our experiments stops were set at 5% down from the peak price during a trade.

When a stop was triggered trading was not resumed until a day had elapsed.

We simulated each of these attributes of trading and had our simulation checked by a suitably qualified external organisation.

The inputs to the rules were constructed by moving a time window over the historical data, the rules were evaluated with the input data and a trading decision for each day was generated. The trading decisions were to go long or short, we did not permit a neutral position unless stops had been triggered. We settled on an output above 0.5 representing a long decision, and below 0.5 a short decision. In our simulation we kept the capital employed at a nominal 1 million units, for spot

markets, and 100 contracts for futures markets. Profits were accumulated but not committed. The data window was swept across the historical data and a trading history generated along with a total final profit or loss figure.

Data Formatting

The pool of values used for input to the rules was created from the sampled time series using a time window. If we were determining the rules trading advice for day n , and had set, for instance, a window size of 3, the pool from which inputs were initially randomly extracted would consist of $Close_{n-1}, Close_{n-2}, Close_{n-3}, Open_{n-1}, Open_{n-2}$, etc.. In previous work^{9,10}, we have made much of the importance of window size in non-linear time series prediction; here we select only the maximum size of the window and let natural selection decide on embedding parameters. In the examples described here a window size of 20 was used

Fitness Algorithm

The profit earned by a trading rule was calculated as described above, and a running total profit/loss series was generated. Various measures derived from this series have been tried out as our fitness algorithms, and clearly the choice made is crucial to the performance of the system. Our current favourite is a mixture of final profit and linearity of the equity curve. It has also proved helpful to pre-test the initial generation for profitability, and discard and replace non-profit making rules until the entire first generation is profitable.

Results

In Sample Performance

Initial training runs on foreign exchange data (DM versus \$) produced results rapidly as can be seen in Figure 3 which shows that the best profit evolved after 12 generations.

The best result found in our initial trials, however, was: *IF Day2 is small OR Day3 is small OR Day4 is small then buy DM*, which yielded a profit of 18%. This is simply a summary of the simple rule 'buy low, sell high'.

(In our system Day 1 represents the most recent data, day2 the day before, and so on.)

Whilst it was encouraging to find that the system spotted the obvious, this is clearly not useful information since the system took advantage of the normalisation to which the data was subjected before use. In taking advantage of the normalisation, the system made use of data not available to the trader. Trading rules which make use of level information will not generalise well since the price of financial instruments can not be expected to remain within historic bounds.. We can obtain rules that generalise better by making use of the relationships between input values. We found however, that given this trivial solution the comparative sets were driven out of the population by absolute sets as training continued. We overcame this by only allowing comparative sets. This yielded the results shown in Figure 4. Note that the profit returned by this algorithm is 62%.

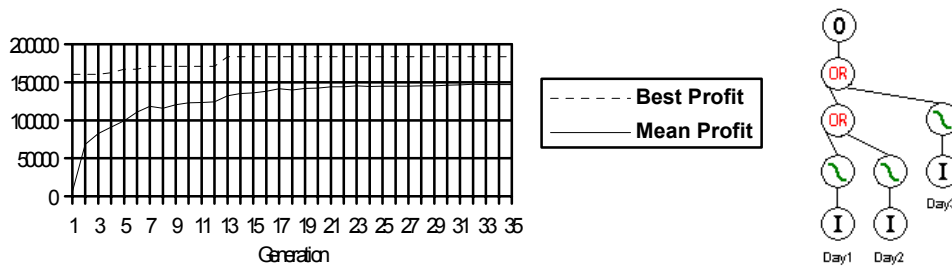


Figure 3, Training run and best rule with mixed sets

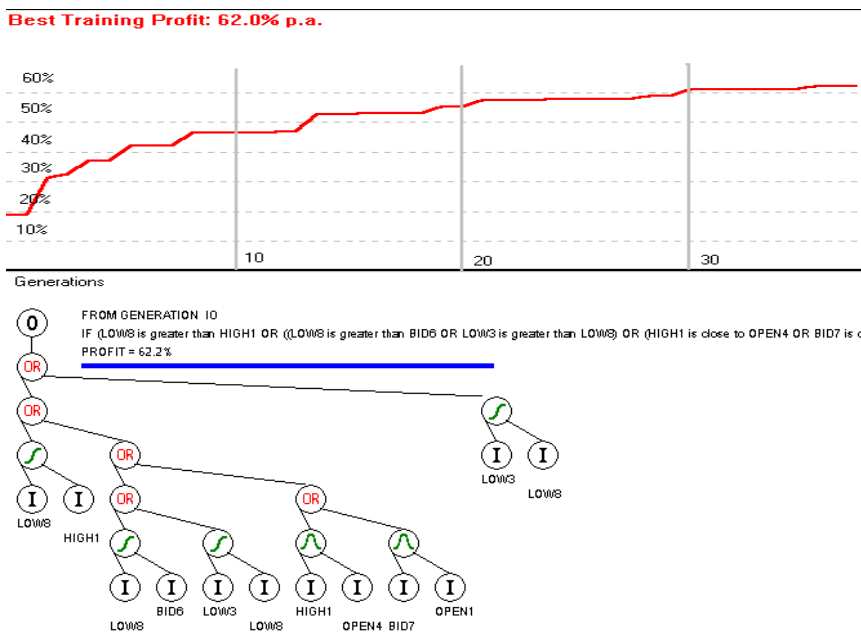


Figure 4, Training run and best rule with comparative sets only

Evolutionary anomalies

We observed an occasional tendency for the population to evolve towards a non-productive special case. For example, a population produced with the same initial conditions as the example above (Figure 4) evolved into a population containing only *Or* links and only *Range* sets.

To prevent this recurring, we added mutation to the parse trees. All nodes were given a probability of 0.005 of mutation. If selected for mutation, the following transformation would be performed on the node:

If an operator:

And \Rightarrow *Or*,

Or \Rightarrow *And*,

Not \Rightarrow *Not*. (*Not* is the only unary operator)

If a set:

either a) *threshold* \Rightarrow *range*, *range* \Rightarrow *threshold*,

or b) a new pair of parameters is randomly chosen,

or c) the input terminals are exchanged.

With equal probability of each.

This made little difference to the general performance of the algorithm, but the anomalous evolution described above has not recurred.

Computational requirements

These are very much subject to the implementation chosen. We have coded our application using C++ in an object oriented fashion. On a 90Mhz Pentium machine running Windows NT and with 20Mbytes of RAM and with the data consisting of 500 days we can expect processing to last around 40 seconds per generation. Thus results are typically available in 30 minutes. The evaluation of the rule once generated requires only milliseconds. It is therefore possible to use this methodology for any trading time frame. Indeed since the rule evaluation requires comparatively simple software, the generation of rules, and their practical use in trading can be performed by separate applications.

The learning algorithm is naturally parallel, and some work has already been performed to produce a distributed version of our software.

Out of Sample Performance

In order to find out if this technique produced rules that generalised well with out-of sample data we tried 5 different daily time series of over 500 days, and left the last

30 trading days out of the training set as an out of sample test. It is our practice to store the best 10 rules, in terms of in-sample performance, of any particular training run as they are generated. The figures below are the average in-sample and out of sample scores of those ten rules annualised so as to make comparison possible. The system was retrained for each series.

	Average in sample annualised profit %	Average out of sample annualised profit %
US T. Bond	61.35	11.58
NIKKEI	61.57	7.67
FTSE	73.75	13.11
S&P	81.93	3.00
DM	39.58	13.73

Note that these figures are unleveraged where futures are concerned and represent percentage profit over the underlying deliverable.

In the above trials the fitness measure was chosen to be a mixture of mostly profit and some equity curve linearity. It is possible to choose various other fitness measures, and thus select whatever in-sample trading characteristics are deemed desirable. There is no reason why for instance maximum draw-down and profit should not be used to generate the fitness measure.

Conclusion

We have shown that the combination of Genetic Programming and a Fuzzy Logic Inference engine produces a powerful methodology for the generation of Fuzzy production rules that are both effective and intelligible. In this work for obvious reasons we have outlined only the bare bones of this technique. In future papers we will cover in more detail various extensions and refinements that we have put in place.

Acknowledgements

Our thanks to David Vanrenen and Reuters Ltd for funding and provision of data and resources, and Walton Asset Management Ltd for the verification of our trading simulation and checking of results.

References

- 1 Zadeh, L..A.. *Fuzzy Sets*. Information and Control **8** 338-353 1965.
- 2 E. Feigenbaum *An Informal Processing Theory of Verbal Learning*, Santa Monica, California, The Rand Corp. 1959.
- 3 Takagi & Sugeno *Derivation of fuzzy control rules from human operator's control actions*. Proceedings. of the IFAC Symposium on Fuzzy Information, Knowledge Representation and Decision Analysis, 55-60 July 1983
- 4 Tsukamoto Y. *An approach to fuzzy reasoning method*. In Mada M. Gupta, Rammohan K. Ragade & Ronald R. Yager, Editors, *Advances in Fuzzy Set Theory and Applications*, 137-149 North-Holland, Amsterdam 1979
- 5 C.-C. Lee *Fuzzy Logic in Control Systems: Fuzzy Logic Controller*, IEEE transactions on Systems, Man & Cybernetics. **20(2)**:419-435,1990
- 6 J.H. Holland *Adaptation in Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press. 1975.
- 7 J.R. Koza *Genetic Programming: on the programming of computers by means of natural selection*, MIT Press, 1992.
- 8 A. Medio, *Chaotic Dynamics - Theory and Applications to Economics*, Cambridge University Press 1992
- 9 A.N. Edmonds, *Multivariate prediction of financial time series using recent developments in chaos theory*. Proceedings of the 1st International Workshop on Neural Networks in the Capital Markets. London Business School, Nov. 1993.
- 10 A.N Edmonds, D. Burkhardt & O. Adjei *Simultaneous Prediction of Multiple Financial Time Series using Supervised Learning and Chaos Theory*. Proceedings of the IEEE ICNN V 3158-3563, Orlando, 1994