

A Genetic Algorithm Based Approach for Discovering Temporal Trends Using Bayesian Networks

Andy Novobilski, Ph.D.

Computer Science, The University of Tennessee at Chattanooga
Chattanooga, TN 37403

Farhad Kamangar, Ph.D.

Computer Science and Engineering, University of Texas at Arlington
Arlington, TX 76019

ABSTRACT

This paper presents a proof of concept system based on Commercial Off The Shelf (COTS) technology that utilizes a genetic algorithm approach to synthesis a Bayesian belief net from an arbitrary stream of temporal data. The discovered Belief Network is capable of predicting the future trend of currency data based on historical performance. The purpose of this system is to evaluate the potential effectiveness of generating a belief net as part of a probabilistic planning activity that would provide an agent with a performance framework that would be resilient to change during the time the plan was being executed.

For this experiment, the Genetic Algorithm portion of the proof of concept was implemented by defining a critter to be used inside of the Goldberg's Simple Genetic Algorithm (SGA). This critter contained both the organization (acyclic inference relationships) of the belief network's nodes and the actual definition of the moving average filters that each nodes represented. The Bayesian belief network was created, trained, and evaluated through the use of a second application, named Netico, that was attached to the SGA extension via a DLL.

Preliminary results from the proof of concept involving currency prices are positive, with a mean accuracy of 83% (including six 100% cases) across 36 test executions representing controlled changes to each of 4 parameters. In addition to presenting these results, the authors describes several areas for future refinement that could be used to increase the utility of the GA-Bayesian Belief Net hybrid.

Keywords: Time-series Forecasting, Bayesian Network, Genetic Algorithm, Netica

INTRODUCTION

Over the past several years, efforts to identify trends in temporal data via future predictions based on some combination of its prior historical performance have taken on a renewed sense of urgency. Gershenfeld and Weigend [1] provide a strong example of this interest and applicability in the application of learning techniques to the future of time series analysis. With there encouragement, a serious competition was sponsored by the Santa Fe Institute

for the prediction of 6 data sets chosen from varying fields of interest, including currency exchange rates.

This paper concentrates on a hybrid approach for predicting the trend of currency data relying upon the techniques of inference (estimation, consistency, uncertainty, assumptions, robustness, and model averaging) brought forth by Glymour, Madigan, Pregibon, and Smyth [2]. The method of inference chosen was to represent the data model in the form of a Bayesian Belief Network. A Bayesian Network is a good choice in a proof of concept situation in that it provides a graphical representation of uncertain knowledge that most people find easy to construct and interpret. This representation has formal probabilistic semantics that makes it suitable for statistical manipulation [3].

In addition to its inference orientation, both organization and probability tables of a Bayesian Network can also be discovered from a time series data set [4]. Mitchell [5] provides an overview of techniques used to train Bayesian Networks from available data, such as the gradient ascent based method. Although alternative methods for finding significant events in time series data such Ciesielski and Palstra's [6] hybrid neural/expert system exist, it is the relative ease of constructing and interpreting attributes of Bayesian Nets that led to their focus in this paper.

Given the selection of a Bayesian Network as the method of inferencing, a second concern relative to the type of data being analyzed needed to be addressed. In this paper, the initial time series was taken from a market/currency price data set assumed to be representable by some combination of moving averages that were related to each other. Unfortunately, the search space for a valid representation is large. A method such as Hekanaho [7] used in his use of GAs to enhance supervised concept learning of rule sets was needed to help guide the learning. Specifically, the use of GAs to overcome the local minima trap of fast greedy learners.

By combining a Genetic Algorithm Approach with a Belief Network, it should be possible to discover robust and consistent networks that provide estimates (with a predicted amount of uncertainty) of the trend of the underlying data set. The only necessary assumption on the data being that it is a time series value for the same item sampled in a periodic fashion. By using moving average filters, the network is able to quantify new assumptions by specifying

inference relationships between small pieces of the time series as a means of predicting future trends.

The following sections describe the approach taken by the paper's proof of concept system from a high level perspective. Next, the actual implementation is described, including the supporting components used as the foundation of this new tool. After describing the approach and implementation, the report continues on to provide results of running the simulation in several different configurations using a dataset composed of actual prices of the Deutch Mark . Finally, conclusions are presented in addition to several areas of interest for future study.

APPROACH

There are two key elements of the approach taken by the proof of concept system described in this paper. The first is the type and limitations of the observable attributes available for use in building the belief networks. The second element is the organization of the network its self, in terms of inference structure and potential for evaluating future information

Operator Space

One of the most common models applied to time series data is the moving average (MA), or Finite Input Response (FIR) filter. For the purpose of our approach, we are going to use a series of moving average based operators to represent attributes within the Belief Network. Each operator C is described as having a positive, stationary, or negative trend (+1, 0, -1) based on the n sized moving average value of Y at time t and Y at time (t-l) according to:

$$C_{n,l}(t) = \begin{cases} +1 & \text{if } Y_n(t) > Y_n(t-l) \\ 0 & \text{if } Y_n(t) = Y_n(t-l) \\ -1 & \text{if } Y_n(t) < Y_n(t-l) \end{cases} \quad [\text{Eq. 1}]$$

Where:

$$Y_n(t) = \frac{1}{n} \sum_{i=0}^{n-1} \text{value}(t-i) \quad [\text{Eq. 2}]$$

This produces attributes that reflect change over a certain period of time, after both values have been passed through a low-pass filter to remove some of the noise in the data series.

Network Organization

The network is organized as a set of attribute nodes and one classification node that is used to predict the trend of the data at time t+1 based on data from time t, t-1, t-2, etc. Each attribute node contains a matrix of conditional probabilities that relate the predicted value of the node to the observed values of the node's predecessors. The final prediction of trend is taken by providing values for all nodes other than the predictor node, and then taking the highest probability result from the predictor node. In order to honor the Bayesian network requirement for an acyclic graph and the GA requirement for a normalized measure of

fitness for the sample without compromising the ability to generate graphs of many different orderings, the following constraints are enforced:

- Each attribute node has the potential to be connected directly to the prediction node.
- A network must have at least one connection from an attribute node to the prediction node to be given a fitness higher than 0.
- Cyclical influence is not allowed.
- The potential exists for dead nodes --- nodes that have no path from them selves either directly or via another node, to the prediction node.

For example, Figure 1 shows a network created by the proof of concept software that uses seven attributes of varying filter and lag characteristics to predict the classification of a case that's handed to it for evaluation. The first letter in the node represents a label for use by the proof of concept software, while the next two numbers represent the value of n and l respectively.

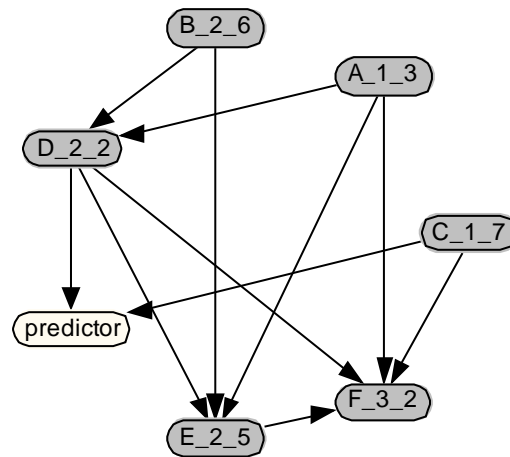


Figure 1 - A Typical SGA Generated Bayesian Belief Network.

There are several observations worth noting before proceeding further. First, nodes E_2_5 (C_{2,5}(t)) and F_3_2 (C_{3,2}(t)) are examples of dead nodes since they have no influence the outcome of the prediction classification node. It would be possible to prune these nodes from the belief net without impacting the accuracy of the outcome, while increasing the performance. Second, the Genetic Algorithm has found a recursive inference between nodes. For example, there is a causal link between A_1_3 and D_2_2 which implies that data at time t-2 is influenced by the value of the data at time t-1. Although not surprising that the relationship exists (time series can often be represented by autoregressive models) it is interesting that the model being explored did not explicitly call the autoregressive relationship out -- it simply did not disallow it from happening.

THE PROOF OF CONCEPT IMPLEMENTATION

Our proof of concept application is implemented by writing glue code between off the shelf versions of a Genetic Algorithm program and a Belief Net program. Specifically, Goldberg's Simple Genetic Algorithm [8], or SGA, is the underlying foundation of Smith and Earickson's [9] C based implementation of the software used in the proof of concept described in this paper. This code makes API style calls into Norsys Software's Netica Application [10] which is the underlying foundation of the Belief Net creation, training, and evaluation software used within our proof of concept application.

Although a full discussion of the source code used to evaluate the concept is beyond the scope of this paper, it is important to spend a few moments and address the software's mode of operation. Specifically, how the critter (genetic chromosome used in the SGA software) is laid out and how the case generation and network evaluations take place.

The Critter Definition

As part of the initial setup of the SGA program, the user specifies constraints for the types of filters allowed in addition to the number of nodes (features) to be used in creating the network. For our application, the critter is defined as the number of moving average filter attributes that can cover the max moving average window value with up to the max lag time increments. Each moving average filter also has a single bit to specify whether it is directly connected to the decision node. Finally, for each node, starting with the n th node, there are $(n-1)$ bits ($1=true/0=false$) that indicate a link exists from node i , where $0 < i < n-1$, to node n . The layout was organized this way to avoid cyclical definitions in the decision tree layout. For example, a critter that represents a decision tree with five nodes, a maximum moving average window of 8, and a maximum comparison lag time of 4 would have bit length of 40 bits:

- 5 nodes at 3 bits (max moving average window) + 2 bits (max comparison time lag) + 1 bit (connect to predictor node)
- plus 4 bits (connections from node 5 to nodes 1-4),
- plus 3 bits (connections from node 4 to nodes 1-3),
- plus 2 bits (connections from node 3 to nodes 1-2),
- plus 1 bits (connections from node 2 to node 1),

In addition to the bit space allocation, the GA software adds one (1) to the values of the filter size and lag distance numbers. This provides for 1 based indexing within the Belief Network.

Case Generation

Training cases are generated dynamically based on the available domain data (in this case, the price of Deutchmarks on a daily basis). This data is used in conjunction with the filters defined by the critter to create a

training set and a single test case. The network is then trained and used to predict the trend of the time series. This step is repeated for each of the number of test cases specified by the user by incrementing where in the data to start and repeating the process.

Network Evaluation

The fitness of the network generated and trained according to the bits in the critter is determined in one of two ways. First, if there are now feature nodes connected to the predictor node, the critter is assigned a fitness of zero. Otherwise, the network is constructed according to the critter, and then trained/evaluated according to the synthesized dataset for the number of test cases specified by the user. The critter's fitness is defined as the percentage of accurately classified data movements, i.e. the actual data moved in the same direction as predicted by the belief network.

ANALYSIS

This section presents the results of running the Application tool for 36 different instances of the various configuration patterns. Results were made available in two forms. The first was a result file that presented a textual description of the fittest network in addition to a sample set of case test data. The textual network description for the network displayed in Figure 1 looks like:

```
A -C[ 1, 3 ] ( t ) - -
B -C[ 2, 6 ] ( t ) - . -
C -C[ 1, 7 ] ( t ) - * -
D -C[ 2, 2 ] ( t ) - * - A->D, B->D,
E -C[ 2, 5 ] ( t ) - - A->E, B->E, D->E,
F -C[ 3, 2 ] ( t ) - - A->F, C->F, D->F, E->F
```

In the above output snippet, the first column is the node name (a letter). The second column displays the node's moving average size and lag for comparison. The third column displays an asterisk to indicate that the node has a direct link to the predictor node. The remainder of the line indicates links that exist from various nodes defined prior to the node in question, to the node in question.

Problem Definition

Each test item of interest actual generated three test runs of the proof of concept system, each for a different size of training cases. The standard parameter set for the application was varied by entering parameter values at the start of each program run. The following 10 items were controllable:

1. The population size of the gene pool. (Default to 100)
2. The maximum number of generations the software could produce before stopping the search procedure. (For example, 5)
3. The probability of crossover occurring. (Default to .8)
4. The probability of a mutation occurring. (Default to .01)

5. The number of past cases in the time series to be used for training. (Default to 10)
6. The number of cases to be used in testing the network during critter evaluation. (Default to 10)
7. The number of attribute nodes (possible filters) available in the network. (Default to 3)
8. The maximum size of the moving average window n. (Default to 4)
9. The maximum size of the time lag used in computing C. (Default to 8)
10. A random number seed.

Before going further, please notice that in the examples, both the population and the number of generations are both very low. This reflects actual values used during the trial runs and are a direct result of the performance of the proof of concept tool. These numbers were chosen to allow the problems to complete in a reasonable period of time. Prudence dictates that much more intensive problem runs be executed before drawing any sweeping conclusions about the technique.

Test/Train Cases	Max Mv Avg	Max Time Lag	Node Cnt	Max Fit	Gen
5 / 5	4	8	3	60%	5
10 / 5	4	8	3	60%	5
20 / 5	4	8	3	80%	5
5 / 10	4	8	3	80%	5
10 / 10	4	8	3	80%	5
20 / 10	4	8	3	90%	5
5 / 20	4	8	3	100%	2
10 / 20	4	8	3	100%	2
20 / 20	4	8	3	85%	5

Table 1 - Varying the number of test cases used.

Test/Train Cases	Max Mv Avg	Max Time Lag	Node Cnt	Max Fit	Gen
5 / 10	2	8	3	80%	5
10 / 10	2	8	3	90%	5
20 / 10	2	8	3	85%	5
5 / 10	4	8	3	80%	5
10 / 10	4	8	3	80%	5
20 / 10	4	8	3	90%	5
5 / 10	8	8	3	100%	2
10 / 10	8	8	3	100%	5
20 / 10	8	8	3	90%	5

Table 2 - Varying the max size of the moving avg filter.

Another point mentioning concerns the reasoning behind such a small number of test cases. At present, the technique described in this paper is being considered as a front end to a probabilistic planner for a short term resource allocation task -- currency trading for example. As a result, I am assuming the ability to replan nightly, with data that was recomputed prior to generating the probabilistic information. Also, the length of time that the prediction needs to hold (keep a higher utility) is relatively small. So, by reducing the size of the test window, the application is able to take advantage of local windows of predictive opportunity and thereby avoid some of the issues associated with over training.

Compiled Results

The test results for the proof of concept system have been compiled into 4 tables based on the parameter (test cases, max avg window, max lag, or max nodes) they are exploring.

Test/Train Cases	Max Mv Avg	Max Time Lag	Node Cnt	Max Fit	Gen
5 / 10	4	4	3	80%	5
10 / 10	4	4	3	70%	5
20 / 10	4	4	3	80%	5
5 / 10	4	8	3	80%	5
10 / 10	4	8	3	80%	5
20 / 10	4	8	3	90%	5
5 / 10	4	16	3	100%	2
10 / 10	4	16	3	100%	2
20 / 10	4	16	3	90%	5

Table 3 - Varying the max len of the comparison lag time.

Test/Train Cases	Max Mv Avg	Max Time Lag	Node Cnt	Max Fit	Gen
5 / 10	4	8	3	80%	5
10 / 10	4	8	3	80%	5
20 / 10	4	8	3	90%	5
5 / 10	4	8	6	80%	5
10 / 10	4	8	6	90%	5
20 / 10	4	8	6	90%	5
5 / 10	4	8	9	80%	5
10 / 10	4	8	9	90%	5
20 / 10	4	8	9	80%	5

Table 4 - Varying the number of nodes in the network.

DISCUSSION

Its interesting to note that there are several perfect classifications systems in each of the three sets of experimental results (this doesn't hold for adding extra nodes). In each case, this 100% classification performance occurs when the maximum freedom is given to the experimental parameter while retaining 10 or less test case. This appears to be a results of not overfitting the data as mentioned in an earlier section.

A second observation is that more nodes are not necessarily better. Indeed, it appears that fewer nodes build a better network. A worthwhile clarification activity would be to look at other data sets with the Proof of concept tool, as well as run an auto-correlation analysis on the information as well to see if there are underlying stochastic trends leading to this behavior.

A third observation is that in the trial runs that achieved success, the lag times were spread out as far as possible. This would seem to indicate that the data contained intrinsic cycles that were related to longer time trends than just one day.

Conclusions

Based on the above observations, it is reasonable to say that the proof of concept system deserves further exploration as a potential adaptive prediction tool. To that end, several immediate tasks need to be accomplished to provide a software platform robust enough for datasets that are closer in magnitude to those available outside of a contrived environment. They are:

- Replace the Netica Library with a version that is optimized for the type of build/train/evaluate over many numbers of critters. This would reduce the time and allow for greater populations sizes to be used.
- Given the large amount of work necessary and the granularity of the type of processing to be done, this approach would benefit greatly from being parallelized.
- Allow for finer granularity of trend prediction.
- The experiments should be extended to multi-variable populations of data for exploration.

Once these tasks are completed, the following set of questions could be explored in greater detail:

1. Would the affect be of allowing different base attribute models (Dynamic Moving Average, Kalman Filter, ARMA, etc.) have on the make up and effectiveness of the generated belief networks.
2. If the genetic algorithm was replaced with a genetic programming mechanism consisting of the base operators (+, -, *, /, and iterate), what type of functions would be learned?
3. Would this system be able to predict the time series described in Gershenfeld and Weigend (1993)?

1 Gershenfeld, N. A.; Weigend, A. S. 1993. "The Future of Time Series." *Time Series Prediction: Forecasting the Future and Understanding the Past*. 1-70. Reading, MA: Addison-Wesley.

2 Glymour, C; Madigan, D.; Pregibon, D.; and Smyth, P. 1996. "Statistical Inference and Data Mining." *Communications of the ACM*. Nov. '96/Vol 39, 35-41.

3 Heckerman, D. 1996. "Bayesian Networks for Knowledge Discovery" *Advances in Knowledge Discovery and Data Mining*, 273-305. Menlo Park, CA: AAAI Press. ISBN#0-262-56097-6.

4 Novobilski, Andrew, F. Kamangar. "A Two-Tiered Cognitive Model for Forecasting Time Series Data", *Second International ICSC Symposium on Neural Computation*, 2000.

5 Mitchell, T. 1997. *Machine Learning*, 188-196. New York, NY: McGraw-Hill. ISBN#0-07-042807-7.

6 Ciesielski, V.; Palstra, G. 1996. "Using a Hybrid Neural/Expert System for Database Mining in market Survey Data." *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* 38-43. Menlo Park, CA: AAAI Press. ISBN#1-57735-004-9.

7 Hekanaho, J. 1997. "GA-based Rule Enhancement in Concept Learning." *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining* 183-186. Menlo Park, CA: AAAI Press. ISBN#1-57735-027-8.

8 Goldberg, D. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, 59-88. Reading, MA: Addison-Wesley. ISBN#0-201-15767-5.

9 Goldberg, D.; Smith, R.; Earickson, J. SGA-C (v1.1) - A C version of the SGA algorithm described by Goldberg as implemented by Smith and later modified by Earickson.

10 The Netica Application API, DLL, and Users Guide. 1997. Norsys Software Corporation, Vancouver, BC, Canada.