Genetic Algorithms for Discrete Sequence Prediction

Anikó Ekárt
School of Computer Science
University of Birmigham
B15 2TT Birmingham
United Kingdom
A.Ekart@cs.bham.ac.uk

Abstract

A statistics-based method using genetic algorithms for predicting discrete sequences is presented. The prediction of the next value is based upon a fixed number of previous values and the statistics offered by the training data. According to the statistics, in similar past cases different values occurred next. If these values are considered with the appropriate weights, the forecast is successful. Weights are generated by genetic algorithms.

1 Introduction

Time series prediction is a problem that has many applications in computing, engineering, economical science and other domains. The particular case of discrete time series where only a predefined number of different symbols can appear in the series is already very complex, with applications such as text compression, computer memory management, adaptive human-machine interfaces [6]. Sequences are also important in human skill learning [8].

The importance of the sequence prediction problem has been recognized by its addition to the list of fundamental problems in machine learning [6]. Laird and Saul define sequence prediction as the task of finding statistical regularities in the input data so that the ability to predict the next symbol becomes better than random guessing, when the input consists of an infinite stream of symbols.

More generally, we can see the sequence prediction problem as a kind of sequence learning [7, 8]. The sequence learning problems are closely interrelated:

- sequence prediction is concerned with predicting elements in a sequence based on previous elements;
- sequence generation is concerned with generating elements in a sequence (basically the same as sequence prediction);
- sequence recognition tries to determine whether a sequence is legitimate or not according to some criteria;
- sequential decision making consists of selecting a sequence of actions to fulfill some goal (such as minimizing some cost).

Sun and Giles [8] give an overview of different sequence learning models. Several neural network models have been developed for sequences: for example, recurrent back-propagation networks are widely used in sequence prediction problems. Hidden Markov Models are also suitable for discrete time series prediction problems [2]. Evolutionary computation is another alternative, it can be used in all types of sequence learning problems due to its generality.

Here we present a hybrid method for sequence prediction that combines a kind of Markovian model and a genetic algorithm [5]. We first construct a probability tree based on the known segment of the sequence. Then we apply genetic search for finding appropriate weights for the predictions provided by the probability tree.

The paper is organized as follows: in Section 2 the prediction method is presented, then in Sections 3 and 4 the details of the two components of the method are shown. In Section 5 we describe the experiments and finally, we draw the conclusions in Section 6.

2 The Prediction Method

2.1 The Discrete Sequence Prediction Problem

Discrete sequences can be found in many practical domains. This article is an example of a discrete sequence containing lexicographic symbols: the letters of the English and the Greek alphabet and the decimal numbers. From text compression or intelligent word processing point of view it is important to note that (throughout this document) for instance any word starting with "seq" is actually one of the three possible words *sequence, sequences, sequential*. Thus, when the letter sequence "seq" is met in the document, it is always followed by "uen", and most of the time by "uence". An intelligent word processing software – trained on this document as it is produced – completes the typed "seq" to "sequen" thus saving the effort of typing three characters. This word processor is in fact a word predictor: it can predict the words as they are typed in.

More generally, a discrete sequence predictor is able to predict – with some accuracy – the future symbols in the sequence, given the observations of previous values.

2.2 The Proposed Prediction Method

Our goal is to create a predictor that is able to predict – as accurately as possible – the next value of a sequence based on a number of previous values. If N previous values $X_1, \ldots, X_N$ are recorded, a prediction for X_{N+1} could be made on the basis of the past $1, 2, 3, \ldots, N-1$ or N values. For doing so, we have at our disposal a finite – but sufficiently large – segment of the sequence.

Without restricting the generality of the problem, we can suppose that the symbols that can occur in the sequence are the numbers $0, 1, \ldots, k-1$, if k is the number of different symbols in the sequence.

Let $P_i^j(X, N)$ denote the probability that based on the i most recent previous values $X_{N-i+1}, \ldots, X_N$, symbol j should follow in the sequence (that is, $X_{N+1} = j$).

Since we generally do not know in advance on what number of previous values it is best to base the prediction, we shall use a weighted sum of all predictions based on $1, 2, \ldots, N$ previous values.

Then, the probability that the next value is j can be computed as

$$P^j(X, N) = \sum_{i=1}^{N} W_i \times P_i^j(X, N), \qquad (1)$$

where W_i is the weight of the prediction based on i values (i.e., $X_{N-i+1}, \ldots, X_N$) that symbol j should follow in the series (that is, $X_{N+1} = j$).

The sequence predictor is built in two phases, as shown in Figure 1. Accordingly, we divide the available segment of the sequence into:

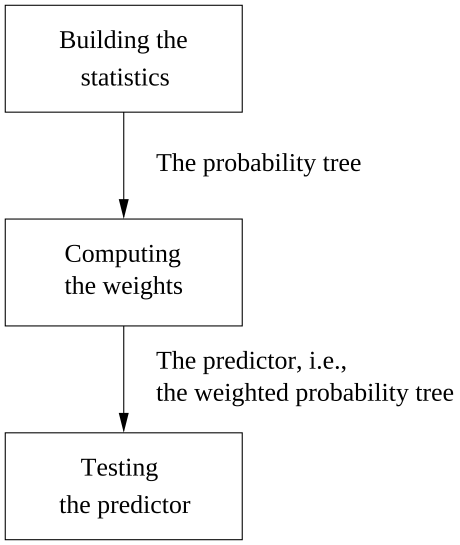

Figure 1: The prediction method.

- training data, further divided into
  - the segment for building statistics; and
  - the segment for computing the weights;
- test data.

The predictions based on i previous values $P_i^j(X, N)$, $i = 1, \ldots, N$, $j = 0, 1, \ldots, k-1$, are computed from the first subset of the training data, for each possible sequence of symbols of length N that appeared in that part of the sequence. The method for computing the probabilities relies on constructing a probability tree for the possible sequences of length up to N as described by Laird and Saul [6] (see Section 3).

Then, a genetic algorithm [5] is applied for finding the appropriate weights. The predicted next value, when using a given weight vector, is

$$X_{N+1} = j, \text{ where } P^j(X, N) = max_{l=0}^{k-1} P^l(X, N). \qquad (2)$$

The weights are evolved for accurately predicting the symbols in the second segment of the training data (see Section 4).

The performance of the evolved predictors is measured on the last segment of the sequence that has been set aside for testing.

3 The Data Structure for Maintaining Statistical Information

We construct a probability tree based on the selected part of the sequence. To each subsequence of length i ($i = 0, \ldots, N$) that oc-

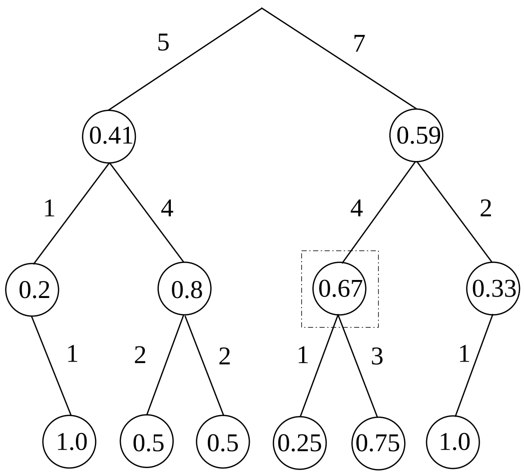

Figure 2: An example probability tree.

curs in this part of the sequence there is associated a path from the root of the tree to a node at depth i in the tree (by convention, the root is at depth $i = 0$). The jth child of a node at depth i corresponds to the situation when the i symbols in the corresponding sub-sequence $X_{N-i+1}, \ldots, X_N$ are followed by $X_{N+1} = j$. This node contains as information the probability value $P_i^j(X, N)$.

For simplicity, we present the tree constructed in the case of a binary sequence ($k = 2$) containing the symbols 0 and 1. Suppose that the part of the sequence for constructing the probability tree is

$$101100101011. \tag{3}$$

We show in Figure 2 the probability tree for $N = 2$ previous values. The left child of each node corresponds to next symbol in the sequence 0 and the right child to next symbol in the sequence 1, respectively. On each arc we mark the number of occurrences of the corresponding sub-sequence. The marked node containing 0.67 corresponds to the sub-sequence 1 followed by next symbol 0. In the given sequence, the sub-sequence 1 is followed by 0 four times and by 1 two times, as marked on the arcs.

The tree is constructed in two steps. First we build the tree structure on the basis of the given sequence. We dynamically create new nodes when needed and update the information associated to the corresponding arcs. When the tree is completed, i.e., all the data in the sequence have been considered, we compute the probability values. The value for *node* is

$$P_{node} = \frac{in(node)}{\sum_{j=0}^{k-1} in(child_j(parent(node)))}, \tag{4}$$

where $in(T)$ denotes the number on the arc from $parent(T)$ to node T and $child_j(T)$ – the jth child of T in the tree.

Now consider the situation when at some later moment a sub-sequence 01 occurs in the sequence. Then, considering the prediction based on two previous values the probabilities of having a 0 or a 1 as next symbol are both equal to 0.5. Based on one previous value, the probability of getting a 0 is 0.67, so the next symbol would be predicted as 0. Based on no previous values, the probability of having a 1 is 0.59, so the next symbol would be predicted as 1. A weighted sum of these probabilities can lead to a more reliable prediction.

4 The Genetic Algorithm

A genetic algorithm [5] was devised for finding the weights of the predictions based on different numbers of previous values.

A genetic search is performed in the space of N-dimensional weight vectors. The component W_i of a weight vector is associated to the prediction based on i previous values. Starting from a random population of weight vectors, by repeatedly applying the so-called genetic operators in a number of iterations (i.e., generations) an improved weight vector is obtained. The fitness evaluation of the individual weight vectors consists of computing their success rate on the selected segment of the sequence.

Evolution is driven by selection of better performing (i.e., fitter) individuals in each generation, the fitter individuals have better chances for survival throughout the generations. The parameters of the genetic algorithm are shown in Table 1. In order to find more accurate predictors, we made experiments with different genetic operators.

As a first approach, we use a simple genetic algorithm with fixed length, real-valued vector representation. So, we adopt a predefined value N for the number of previous values on which the prediction is based.[1] We employ the roulette wheel selection method, and mutation and crossover operators suitable for real-

1. Finding the appropriate N will be the purpose of future experiments.

Table 1: The parameter setting for the genetic algorithm.

Population size	100
Individual	Weight vector W
Gene	Weight W_i
Crossover probability	90 %
Mutation probability	10 %
Selection method	Roulette wheel
Max. nr. of generations	50

valued strings. Mutation consists in the modification of the value of a randomly selected gene. We consider three types of crossover: the simple single point crossover, the median crossover and the so-called blend crossover (see Section 5). We evaluate the performance of the generated weight vectors on the segment of the sequence set aside for this purpose (as explained in Section 2). For a segment of the sequence containing M symbols, we calculate the success rate of a weight vector W (i.e., its fitness) as:

$$Fitness(W) = \frac{1}{M}\sum_{p=1}^{M} S(p), \qquad (5)$$

where

$$S(p) = \begin{cases} 1 & \text{if the predicted value in position} \\ & p \text{ is the same as the actual value,} \\ 0 & \text{otherwise.} \end{cases} \qquad (6)$$

The predicted value is calculated by using (2). For example, consider a predictor based on the probability tree in Figure 2 constructed for sequence (3), with weight vector $W = [0.4\ 0.6]$ and the segment of the sequence for evaluation

$$001 \qquad (7)$$

following the segment in (3). Then the weight vector W is assigned $Fitness(W) = \frac{2}{3}$, where

$$\begin{aligned} S(1) &= 1, \text{ the predicted value is 0, with} \\ P^0 &= 0.4 \times 0.67 + 0.6 \times 1 = 0.868; \\ S(2) &= 0, \text{ the predicted value is 1, with} \\ P^1 &= 0.4 \times 0.8 + 0.6 \times 0.75 = 0.77; \\ S(3) &= 1, \text{ the predicted value is 1, with} \\ P^1 &= 0.4 \times 0.8 + 0.6 \times 1 = 0.92. \end{aligned}$$

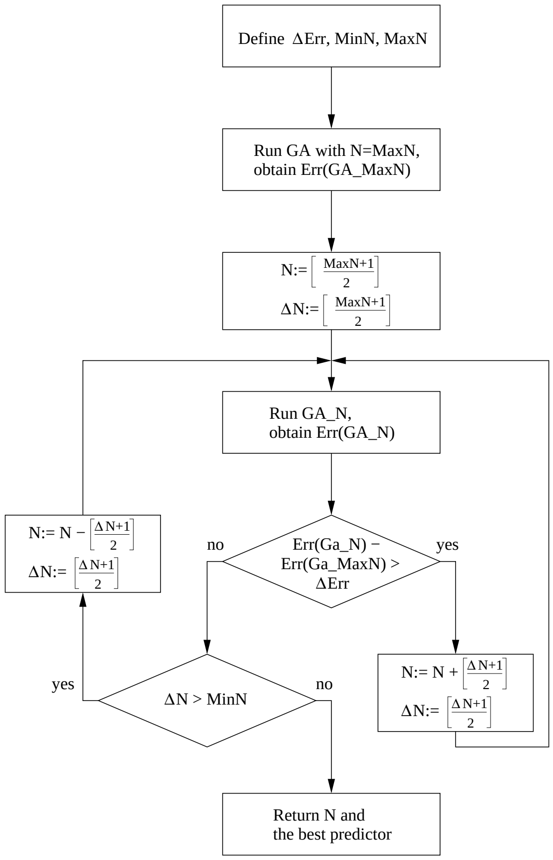

Figure 3: The combined binary-genetic search for the appropriate N.

4.1 Extension of the Genetic Algorithm for Finding the Appropriate N

The problem of finding the appropriate number of previous values N could be approached in two ways:

- by running the fixed length vector representation genetic algorithm with different values of N; or
- by using a variable length vector representation genetic algorithm.

In the first case, the genetic algorithm can be combined with a binary search for the appropriate value of N, as it is shown in Figure 3. If the prediction error is likely to become larger when we decrease N, it is fairly easy to devise a binary search for the least value of N that provides satisfactory prediction accuracy. One has to define the acceptable loss in prediction accuracy ($\Delta\, Err$), the maximum number of previous values ($MaxN$), and the satisfactory precision for N ($MinN$). Then the combined binary-genetic search finds the least value for N (with precision $MinN$), for which the loss of accuracy is less than $\Delta\, Err$ (compared to the prediction accuracy for $MaxN$).

In the second case, the genetic algorithm operates on weight vectors of different lengths. A maximum length $MaxN$ should be defined, so that the probability tree could be constructed in advance. Since W_i encodes the weight for the prediction based on i previous values, the conversion of the fixed-length representation to a variable length representation is straightforward. The crossover operators need a slight modification; namely, the shorter vector should be extended with zeroes up to the length of the longer vector, and then any crossover operator can be applied in its original form. These zeroes are actually incorporated in the shorter vector: the predictions corresponding to the non-existing weights are not taken into consideration, i.e., their weight is zero. If the assumption that accuracy increases with N is true then the variable length representation genetic algorithm is most likely to lead to solution vectors of maximum length.

5 Experimental Results

The experiments were performed on the sequence provided at the *Binary Time Series* competition organized at the *Congress on Evolutionary Computation* in 1999 [1].

The time series contains 10000 data bits. We used the first 9000 bits for training the predictor and tested its accuracy on the remaining 1000 bits. We employed $N = 100$ previous values. We found that the accuracy of the evolved predictor depends on the used genetic search operators and the proportion of training data used for creating the probability tree and for generating the weights, respectively.

We tested three types of crossover operators:

- Simple crossover - for parents P_1 and P_2, and crossover point x the offspring are

  $C_1 = P_1(1)\ldots P_1(x)P_2(x+1)\ldots P_2(N)$ and

  $C_2 = P_2(1)\ldots P_2(x)P_1(x+1)\ldots P_1(N)$;

- Median crossover - the same as simple crossover with the exception of the genes at the crossover points:

  $C_1(x) = \alpha\, P_1(x) + (1-\alpha)\, P_2(x)$ and

  $C_2(x) = (1-\alpha)\, P_1(x) + \alpha\, P_2(x)$

  where $\alpha = 0.7$;

- Blend crossover [4] - the same as simple crossover with the exception of the genes at the crossover points, which are selected as random numbers in the interval

  $[P_1(x) - \alpha\,(P_2(x) - P_1(x)),$

  $P_2(x) + \alpha\,(P_2(x) - P_1(x))]$,

  where $P_1(x) \leq P_2(x)$ and $\alpha = 0.5$.

In Table 2 the number of mistakes per 100 predictions for the evolved predictors averaged over 10 runs of the genetic algorithm is shown. We present the results for two distributions of the training data: 7000 bits for probability tree, 2000 for weights (7_2_1) and 8000 bits for probability tree, 1000 for weights (8_1_1), respectively. We show the error rates of the resulting predictors on the training data and on the test data as well.

The predictors are more accurate when the proportion of data used for creating the probability tree is higher. There are only slight differences for the different crossover types. The evolved predictors are quite robust, they generalize well to unseen test data.

If we look at the evolution of the genetic algorithm over the generations (Figure 4), we notice that the initial random weights were actually quite good already, the genetic algorithm brought only slight improvement. However, the performance of the genetic algorithm is strongly related to the choice of genetic operators. We intend to experiment with more sophisticated crossover operators, such as the simulated binary crossover of Deb and Agrawal [3], in the hope of generating even better predictors.

6 Conclusions

We can conclude that the presented method is able to generate considerably accurate predictors for discrete time series, assuming there is a sufficiently large segment of the series available to provide reliable statistics.

In the future we plan to conduct experiments on discrete sequences containing more than two

Table 2: The average error rates for the solutions obtained in different experiments.

Error rates	7_2_1		8_1_1	
	Train	Test	Train	Test
Simple Xover	8.31	9.04	7.52	8.65
Median Xover	8.24	8.93	7.65	8.61
Blend Xover	8.25	8.98	7.36	8.54

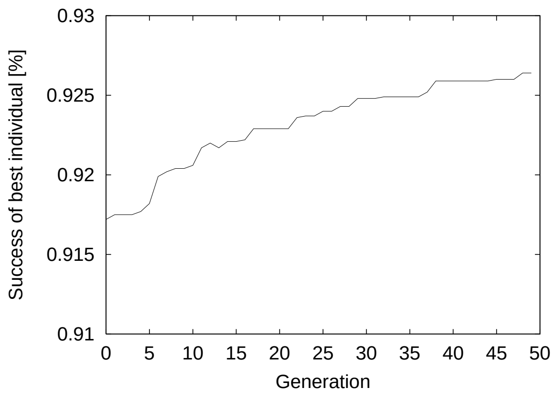

Figure 4: The learning curve for experiment 8_1_1 averaged over 10 runs.

symbols. We also plan to experiment with the variable length representation genetic algorithm for detecting the appropriate number of previous symbols in the sequence on which the prediction of the next symbol should be based.

References

[1] D. Ashlock. Time series competitions, Congress on Evolutionary Computation. http://www.math.iastate.edu/danwell/EP99 /time.html, 1999.

[2] Y. Bengio. Markovian models for sequential data. *Neural Computing Surveys*, 2:129–162, 1999.

[3] K. Deb and R. B. Agrawal. Simulated binary crossover for continuous search space. Technical Report IITK/ME/SMD-94027, Indian Institute of Technology, Kanpur, 1994.

[4] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval schemata. In D. Whitley, editor, *Foundations of Genetic Algorithms, II*, pages 187–202, 1993.

[5] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[6] P. Laird and R. Saul. Discrete sequence prediction and its applications. *Machine Learning*, 15:43–68, 1994.

[7] R. Sun. Introduction to sequence learning. In *Sequence Learning: Paradigms, Algorithms, and Applications*, volume 1828 of *LNAI*, pages 1–10. 2000.

[8] R. Sun and C. L. Giles. Sequence learning:from recognition and prediction to sequential decision making. *IEEE Intelligent Systems*, July/August:67–70, 2001.