Searching Cellular Automata Rules for Solving Two-Dimensional Binary Classification Problem

Anna Piwonska[1], Franciszek Seredynski[2,3], and Miroslaw Szaban[4]

[1] Bialystok University of Technology, Computer Science Faculty, Poland
[2] Polish-Japanese Institute of Information Technology, Poland
[3] Cardinal Stefan Wyszynski University in Warsaw, Poland
[4] Siedlce University of Natural Sciences and Humanities, Poland
a.piwonska@pb.edu.pl, sered@pjwstk.edu.pl, mszaban@uph.edu.pl

Abstract. This paper proposes a cellular automata-based solution of a two-dimensional binary classification problem. The proposed method is based on a two-dimensional, three-state cellular automaton (CA) with the von Neumann neighborhood. Since the number of possible CA rules (potential CA-based classifiers) is huge, searching efficient rules is conducted with use of a genetic algorithm (GA). Experiments show an very good performance of discovered rules in solving the classification problem. The best found rules perform better than the heuristic CA rule designed by a human and also better than one of the most widely used statistical method: the k-nearest neighbors algorithm (*k-NN*). Experiments show that CAs rules can be successfully reused in the process of searching new rules.

Keywords: Cellular Automata, Binary Classification Problem, Genetic Algorithm.

1 Introduction

CA is a discrete, dynamical system composed of many identical cells arranged in a regular grid, in one or more dimensions [9]. Each cell can take one of a finite number of states and has an identical arrangement of local connections with other cells called a neighborhood, which also includes the cell itself. After determining initial states of all cells (an initial configuration of a CA), states of cells are updated synchronously at discrete time steps, according to a local rule defined on a neighborhood, with the assumption of particular boundary conditions.

CAs were intensively studied in many science disciplines, including computability theory, mathematics, physics, theoretical biology, etc. The reason is that in spite of their simple construction and principle of operation, these relatively simple systems can exhibit complex global behavior.

In spite the fact that CAs have the potential to efficiently perform complex computations, the main problem is a difficulty of designing CAs producing a desired behavior. One must not only select a neighborhood type and size, and above all the appropriate rule (or rules). Since the number of possible rules is usually huge, this is the extremely hard task. In some applications of CAs one can design an appropriate rule by hand based on partial differential equations describing a given phenomenon, see for example [7]. However, it is not always possible. In the 90-ties of the last century Mitchell and collaborators proposed to use GAs to find CAs rules able to perform one-dimensional density classification task [6] and the synchronization task [2].

Statistical classification is the problem of determining to which class new observations belong, based on the training set of data containing observations whose class is known. The binary classification deals with only two classes, whereas in a multiclass classification observations belong to one of several classes. The well-known classifiers are neural networks, support vector machines, $k-NN$ algorithm, decision trees and others.

The idea of using CAs in the classification problem was described by Maji et al. [5], Povalej et al. [8] and recently by Fawcett [3]. Fawcett designed the heuristic rule based on the von Neumann neighborhood (so-called voting rule) and tested its performance on different data sets. This paper proposes a different approach: finding appropriate CAs rules by a GA. The effectiveness of rules discovered by a GA will be compared with the effectiveness of the hand-designed voting rule. Both CA-based approaches will be compared with the $k - NN$ algorithm.

This paper is organized as follows. Section 2 defines the classification problem and describes the proposed CAs-based algorithm. Experimental results are presented in Section 3. The last section contains conclusions.

2 Binary Classification Problem and Cellular Automata

2.1 Two-Dimensional Binary Classification Problem

In this paper we deal with the classification problem described in [4] in the context of fuzzy rule-based classification system. Let us assume that the data space is the unit square $[0, 1] \times [0, 1]$. Suppose that m data-points $\mathbf{x}_p = (x_{p1}, x_{p2})$, $p = 1, 2, ..., m$ are given as a training set from two classes: class 1 and class 2. That is, the classification of each $\mathbf{x}_p = (x_{p1}, x_{p2})$, $p = 1, 2, ..., m$ is known as one of two classes. The classification problem can be stated as follows. Given m training data find a rule (or "classifier") which divides the data space into two disjoint decision areas (class 1 or 2) such that the class number can be assigned to any new observation.

2.2 Proposed CA-Based Classifier

The idea of using CAs to solve the binary classification problem is based on the construction of a CA and finding an appropriate rule which can perform the classification task. Since the problem is defined in the two-dimensional space, our CA will also be the two-dimensional. The CA works on a grid of cells, so we must partition our data space $[0, 1] \times [0, 1]$ into a grid of cells. Let us assume that the considered CA has an equal number of cells in each dimension ($N = M$).

Our CA is a three-state automaton ($k = 3$). The initial state of each cell is set by training points belonging to this cell and is determined in the following way:

- if there is no point in a cell (an empty cell), then a cell is in state 0 (a cell is marked in grey color),
- if there are points only from class 1 in a cell, then a cell is in state 1 (a cell is marked in white color),
- if there are points only from class 2 in a cell, then a cell is in state 2 (a cell is marked in black color),
- if there are points from both classes (class 1 and 2) in a cell, then a cell is in state 0 (a cell is marked in grey color).

The interpretation of the above mentioned rules is simply and intuitive. The class of training points determines the state of a cell. If the state of a cell cannot be assigned, a cell is in state 0 (unknown class). This can happen in two situations: either there are no training points in a cell or there are points from both classes in it. The performance of the CA-based classifier depends on the size of a partition. If a partition is too coarse, the performance of the system may be low (many observations may be misclassified). On the other hand, if a partition is too fine, one can observe the lack of training points in corresponding cells.

The next step is to determine the neighborhood type, along with boundary conditions. We assumed the von Neumann neighborhood and null boundary conditions: border cells have dummy neighbors always in state 0.

After determining initial states of all cells (i.e. the initial configuration of the CA), cells change their states synchronously according to a certain rule which must be found. An appropriate rule transforms, during T time steps, the initial configuration of the CA into the final configuration in which there are no empty cells and for which the correct class number can be assigned to any new observation. Finding an appropriate rule is a key factor for performance of CA-based classifier.

Let us first consider the heuristic rule for the classification problem designed by Fawcett [3]. The rule, called n4V1nonstable, is a non-stable update rule defined on the von Neumann neighborhood with $k = 3$, in which a cell may change its state if the majority changes. According to this rule, the state of a cell at the next time step is determined in the following way:

- if $neigh1 + neigh2 = 0$, then a cell state will be 0,
- if $neigh1 > neigh2$, then a cell state will be 1,
- if $neigh1 < neigh2$, then a cell state will be 2,
- if $neigh1 = neigh2$, then a cell state will be $\text{rand}\{1, 2\}$,

where $neigh1$ and $neigh2$ denote the number of a cell's neighbors, respectively in state 1 and 2, and $\text{rand}\{1, 2\}$ selects randomly 1 or 2 with equal probability. After determining initial states of all cells, the CA runs for a maximum number of T time steps (if two subsequent CA's configurations are identical, the run is stopped). The intention is that cells will organize themselves into regions of similar class assignment (class 1 or 2). Details of this approach along with the extension to multi-dimensional data mining are described in [3].

In our approach there is the same goal but we want to discover such rules by the GA and compare them with the hand-designed n4V1nonstable rule and with the $k-NN$ algorithm. The quality of a given CA rule (n4V1nonstable rule and rules discovered by the GA) is determined on the base of a final configuration of the CA. We generate l new observations of the classification problem and test if new points fall into cells with right states. If a cell is in state 1 then "the answer" of the CA is: "the class of all points falling into this cell is 1" (and similarly with state 2). In rare cases, when a final configuration contains cells in state 0, new points falling into these cells cannot be classified. The score of a CA rule is the sum of the correctly classified points.

2.3 A Genetic Algorithm for Searching Efficient Rules

Assuming the von Neumann neighborhood, with three possible cell states and the neighborhood size equal to 5 we have $3^5 = 243$ possible neighborhood states. Thus, the length of a CA rule is equal to 243 and the number of possible rules is equal to 3^{243}. We use the neighborhood coding described below.

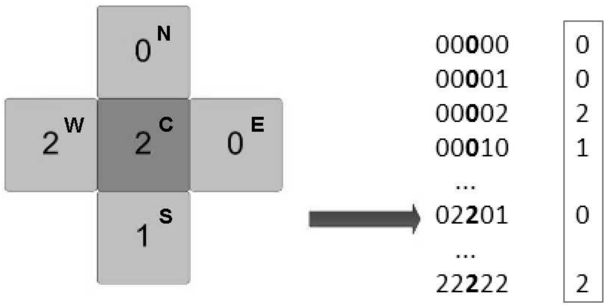

Fig. 1. The neighborhood coding (on the left) and the fragment of the rule - the chromosome of the GA (on the right, in the rectangle)

Five cells of the von Neumann neighborhood are usually described by directions on the compass: North (N), West (W), Central (C), East (E), South (S). Fig. 1 (left) presents the example of such a neighborhood: 02201. It also lists possible neighborhood states and presents the example of CA rule (on the right, in the rectangle). The value at position 0 in the rule (the value at the top in the rectangle) denotes a state of the central cell of the neighborhood 00000 at the next time step, the value at position 1 in the rule denotes a state of the central cell of the neighborhood 00001 at the next time step and so on, in lexicographic order of neighborhoods. One can see that the central cell of the neighborhood 02201 will be in state 0 at the next time step.

The next step is to evaluate individuals in the initial population for the ability to perform the classification task. For this purpose each rule in the population is run on the initial configuration of a CA for T time steps. The initial configuration corresponds to the given problem instance and is determined as described in Section 2.2. The final configuration of a CA is used to compute the following fitness function components:

– $n0$ - the number of cells in state 0,
– nc - the number of cells in correct state (1 or 2),
– ni - the number of cells in incorrect state (1 or 2),
– nb - the number of cells with a "suspicious neighbor".

Cells in correct states are these cells in state 1 or 2 whose states in the initial configuration remained unchanged in the final configuration. Cells in incorrect states are these cells in state 1 whose state in the initial configuration was 2 and vice versa: these cells in state 2 whose state in the initial configuration was 1. A cell with a "suspicious neighbor" is a cell which has at least one neighbor in different state than cell's own state. These values are used to compute the fitness f of a rule i, denoted as f_i:

$$f_i = nc - ni - n0 - w \cdot nb \ , \tag{1}$$

where $w \in \langle 0, 1 \rangle$ is a coefficient used to adjust the influence of the number of cells with a "suspicious neighbor" on the fitness. Omitting nb factor causes that the GA tends to evolve CA rules which change states of empty cells into the state 1 or 2 randomly: in the final configuration cells in states 1 and 2 do not form consistent regions, as one would expect.

Once we have the genetic representation and the fitness function defined, the GA starts to improve the initial population of rules through repetitive application of selection, crossover and mutation operators. In our experiments we used the selection scheme described by Mitchell [6] in which E best individuals (the elite) are copied without modifications to the next generation. The remaining $P - E$ rules are formed by crossover and mutation from the elite rules. Crossover between two rules involves randomly selecting a single crossover point in the rules and exchanging parts of the rules before and after this point. Mutation is performed for each individual in the population (with the exception of the elite rules) with the probability p_m. When a given gene is to be mutated, we replace the current value of this gene by the value 1 or 2, with equal probability. Omitting the value 0 prevents from evolving rules with many 0s. Such rules are more likely to produce configurations containing cells with state 0. It would be unfavorable situation.

These steps are repeated through G generations. Then, the quality of the final population of rules is tested on $l = 1000$ randomly generated new points of the classification problem. A new point is classified correctly if it falls into a cell whose state is the same as the class of a point. The quality of a rule is measured by the number of correct classifications. The higher score a rule obtains, the better classifier it represents. The result of the best rule is considered as the result of the proposed method.

3 Experimental Results

As test problems, we took two classification problems. In each problem the data space $[0,1]\times[0,1]$ is divided into two classes according to the value of the function $g(\mathbf{x})$, i.e. if $g(\mathbf{x}) \geq 0$ then $\mathbf{x}$ belongs to class 1, else $\mathbf{x}$ belongs to class 2. The functions used in experiments are:

– Problem 1: $g(\mathbf{x}) = -sin(2\pi x_1)/4 + x_2 - 0.5$ [4]
– Problem 2: $g(\mathbf{x}) = -x_1^3 + x_2 - 0.3$

For both problems we randomly generated 10 problem instances, where each of them had 50 points in class 1 and 50 points in class 2 ($m = 100$). Each problem instance was used to determine the initial configuration of the CA, with the number of cells in each dimension equal to 10 (grid 10×10).

The parameters of the CA and the GA were the following: $T = 50$, $P = 200$, $E = 50$, $p_m = 0.05$, $w = 0.1$, $G = 500$. The parameters were tuned during many experiments and these values were chosen to final runs.

Since the GA and the n4V1nonstable rule are probabilistic, five runs of each of them were performed. The best of five runs of the GA (CA-GA (10×10)) were compared with the best five runs of the n4V1nonstable rule (n4V1 (10×10), $T = 50$) and with the best results obtained by the distance weighted $k - NN$ method ($k - NN$) [1]. The $k - NN$ was allowed to use up to five neighbors and the best k was determined experimentally. Results of these experiments are presented in Table 1 and Table 2.

Table 1. Problem 1: the number of correct classifications, *l=1000*

instance	CA-GA 10×10	CA-GA 20×20	$k - NN$	n4V1 10×10	n4V1 20×20
1	938	974	926	908	928
2	965	974	981	951	977
3	956	967	951	914	951
4	936	943	944	917	944
5	966	975	962	942	953
6	943	965	952	950	964
7	951	961	961	955	947
8	951	964	956	917	953
9	967	971	967	934	936
10	953	923	927	939	925
average	952.6	**961.7**	952.7	932.7	947.8

The structure of these tables is as follows. The first column lists the number of a problem instance, the second and the third present results obtained by the GA on grids 10×10 and 20×20, the fourth lists results obtained by the $k-NN$ and the last two columns present results obtained by n4V1nonstable rule on grids 10×10 and 20×20.

Table 2. Problem 2: the number of correct classifications, *l=1000*

instance	CA-GA 10×10	CA-GA 20×20	$k - NN$	n4V1 10×10	n4V1 20×20
1	964	975	959	952	956
2	954	976	973	966	973
3	969	970	962	964	963
4	967	964	962	943	965
5	970	972	972	957	963
6	975	977	966	963	959
7	961	965	945	956	939
8	941	978	939	970	946
9	965	972	947	937	941
10	972	990	967	939	962
average	963.8	**973.9**	959.2	954.7	956.7

As it was mentioned, in the first experiment we compared the results obtained by both CA-based methods on grid 10×10 with the results obtained by the $k - NN$.

In the case of Problem 1, the CA-GA approach received better results than the $k - NN$ and n4V1nonstable rule in four instances (instances: 1, 3, 5, 10). In the case of instance 9, the CA-GA obtained the same result as the $k - NN$. The $k - NN$ received better results than both CA-based methods in other five instances. The n4V1nonstable was never the best. Looking at the average results one can see that the best values were obtained by the $k - NN$. However, the CA-GA algorithm performed significantly better than the n4V1nonstable rule.

In the case of Problem 2, the CA-GA method gained higher score than the $k - NN$ and the n4V1nonstable rule in seven cases (instances: 1, 3, 4, 6, 7, 9, 10). In two cases (instances 2 and 5) the $k - NN$ received better results than both CA-based methods. N4V1nonstable was the best in instance 8. The best average result was obtained by the CA-GA which also performed better than n4V1nonstable.

However, it is worth to notice that the CA-based methods perform on a grid of cells (in contrast to the $k - NN$) and the performance of these methods strongly depends on the size of the partition (the number of cells in each dimension). In the second series of experiments we tested the quality of n4V1nonstable rule on the same problem instances, but on the grid size 20×20. Results of these experiments are presented in the sixth column in Table 1 and Table 2 (n4V1 (20×20)).

The CA-GA method was also tested on the grid size 20×20. We performed two series of experiments. In the first case the initial population of rules was created fully randomly. In the second case, we took five best rules from the final populations obtained for the grid size 10×10 for each problem instance and then inserted them into initial populations of the GA (grid size 20×20). Experiments showed that the GA (grid size 20×20) with the initial population containing

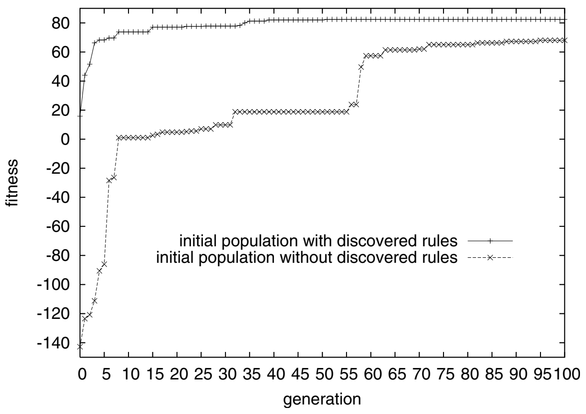

Fig. 2. Problem 2, instance 10: fitness of the best individual

previously discovered rules evolved better solutions than the GA with randomly generated initial population. Moreover, inserting the best rules into the initial population caused that the efficient rules were discovered very quickly.

As the example, let us look at the instance 10 of Problem 2. Fig. 2 presents two runs of the GA (with the grid size 20×20, first 100 generations marked) in the case of randomly generated initial population and in the case of inserting previously discovered rules into it. In the first case, the GA needs 500 generations (not shown in the figure) to evolve the best rule. In the second case, one of the best rules found for the grid size 10×10 receives high fitness value on the double grid size (15.80 in the generation 0). The GA quickly improves the best rule and as early as in the sixtieth generation the final, best rule is discovered. This rule obtained the high score equal to 990 (see, Table 2, instance 10).

Fig. 3 and 4 present final configurations of the best rules during selected generations of the GA in the case of randomly generated initial population (Fig. 3) and inserting previously discovered rules into the initial population (Fig. 4). One can see the best rule in the random initial population performs chaotically (the first picture in the first row, Fig. 3). On the contrary, the best rules found for the grid size 10×10 performs quite good on the double grid size (the first picture in the first row, Fig. 4). This is the evidence of the scalability of CAs rules: rules discovered for a given problem instance can be used to solve the same problems on denser grids. Results of the experiment in which previously discovered rules are inserted into the initial population of the GA are presented in the third column of both tables (CA-GA (20×20)). One can see that this approach usually improves results for both problems in comparison with the CA-GA performing on the grid 10×10. Only in two cases (Problem 1, instance 10 and Problem 2, instance 4) obtained results were worse. In the case of Problem 1 the average result has improved from 952.6 to 961.7 and in the case of Problem 2 from 963.8 to 973.9.

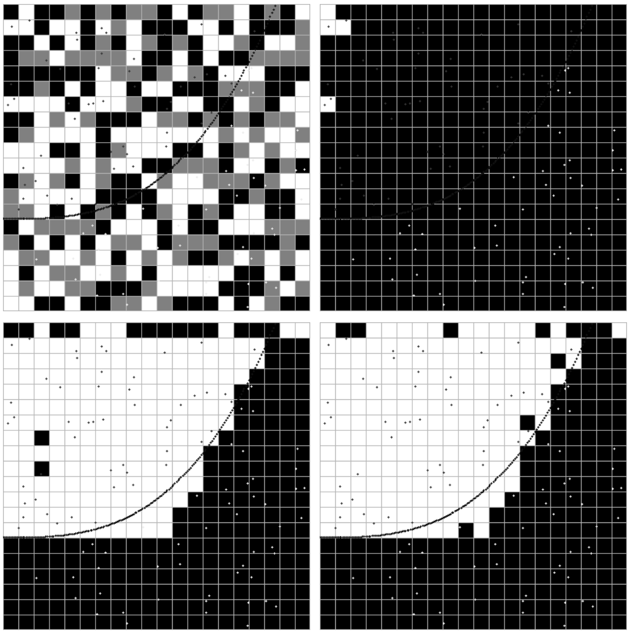

Fig. 3. Problem 2, instance 10, random initial population: the final configurations of the best rule in the GA generations: 0, 20, 150, 500

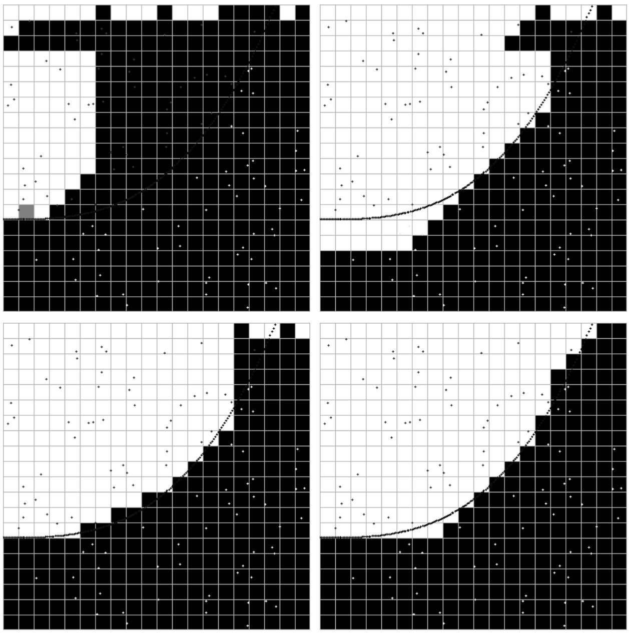

Fig. 4. Problem 2, instance 10, initial population with discovered rules: the final configurations of the best rule in the GA generations: 0, 8, 20, 60

4 Conclusions

In this paper we have presented the new approach concerning binary classification in the context of CAs. The main purpose of the paper was to study possibilities of the GA in discovering CA rules which are able to perform binary classification task. Results of presented experiments show that the GA is able to discover rules appropriate to solve this task for a given instance of a problem. The best found rules perform better than the heuristic rule designed by human and better than the $k-NN$ algorithm.

Conducted experiments showed very interesting ability of discovered rules, i.e. the ability of quickly adopting to larger grid size. During artificial evolution rules store some kind of knowledge about instance which is solved. This knowledge can be successfully reused in the process of discovering rules defined on larger grid size. We can interpret the first run of the GA on the grid size 10×10 as the preprocessing phase. When more precise results are needed, the best rules can be again used by the GA searching on more fine grid partition.

References

1. Bailey, T., Jain, A.: A note on distance-weighted k-nearest neighbor rules. IEEE Transactions on Systems, Man and Cybernetics 8(4), 311–313 (1978)
2. Das, R., Crutchfield, J., Mitchell, M.: Evolving globally synchronized cellular automata. In: Proceedings of the 6th International Conference on Genetic Algorithms, pp. 336–243 (1995)
3. Fawcett, T.: Data mining with cellular automata. ACM SIGKDD Explorations Newsletter 10(1), 32–39 (2008)
4. Ishibuchi, H., Nozaki, K., Yamamoto, N.: Selecting fuzzy rules by genetic algorithm for classification problems. Fuzzy Systems 2, 1119–1124 (1993)
5. Maji, P., Sikdar, B.K., Pal Chaudhuri, P.: Cellular Automata Evolution for Pattern Classification. In: Sloot, P.M.A., Chopard, B., Hoekstra, A.G. (eds.) ACRI 2004. LNCS, vol. 3305, pp. 660–669. Springer, Heidelberg (2004)
6. Mitchell, M., Hraber, P., Crutchfield, J.: Revisiting the edge of chaos: Evolving cellular automata to perform computations. Complex Systems 7, 89–130 (1993)
7. Omohundro, S.: Modelling cellular automata with partial differential equations. Physica 10D 10(1-2), 128–134 (1984)
8. Povalej, P., Lenič, M., Kokol, P.: Improving Ensembles with Classificational Cellular Automata. In: Gallagher, M., Hogan, J.P., Maire, F. (eds.) IDEAL 2005. LNCS, vol. 3578, pp. 242–249. Springer, Heidelberg (2005)
9. Wolfram, S.: A New Kind of Science Wolfram Media (2002)