ACTA UNIVERSITATIS AGRICULTURAE ET SILVICULTURAE MENDELIANAE BRUNENSIS

Volume LXI 274 Number 7, 2013
http://dx.doi.org/10.11118/actaun201361072445

CLASSIFICATION OF ECONOMIC DATA INTO MULTIPLE CLASSES BY MEANS OF EVOLUTIONARY METHODS

Jiří Lýsek, Jiří Šťastný

Received: April 11, 2013

Abstract

LÝSEK JIŘÍ, ŠŤASTNÝ JIŘÍ: *Classification of economic data into multiple classes by means of evolutionary methods.* Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis, 2013, LXI, No. 7, pp. 2445–2449

In this contribution we deal with an automatic classification of economic data into multiple classes. A classifier created by grammatical evolution is used to determine the data sample membership into one of the defined classes. The grammar rules used for classifier structure creation are presented. The performance of our classifier is compared with multilayer perceptron neural network classifier and Kohonen neural network classifier. We used a survey data of consumer behaviour in food market in Czech Republic.

genetic algorithms, classification, grammatical evolution, learning

We present a method which uses grammatical evolution as a learning process for creation of a multiclass classifier. Data from economic research was used for classifier training and results are compared with previously used methods. Automatic classification performed by artificial intelligence is an important research field as the methods can help a human expert in achieving critical solutions.

Genetic algorithms are often used to find solutions of difficult optimisation problems. The search for solution performed by genetic algorithm is based on repeated evaluation of individuals in a population. The individuals in the population represent possible problem solutions. Genetic operators such as crossover and mutation are applied on the population individuals in every iteration of genetic algorithm. These operators are responsible for creation of new individuals which should be fitter for solving given problem than theirs ancestors. Individuals more fit for problem solution are given better fitness value and are more likely to be selected for reproduction. Thanks to this, the information stored in them will prevail more probably into next generation (Koza, 1992). Usually the fitness value is maximized over the iterations of algorithm and we need to specify a formula for its calculation.

The genetic algorithm, specifically grammatical evolution to search for a classifier program is used. Grammatical evolution is a process which uses principles of artificial evolution to create short computer programs (Ryan and O'Neill, 2003). We pass a list of rewriting rules, a context-free grammar G = {N, T, P, S}, to a grammatical evolution system. A context-free grammar is a set of non-terminal symbols N, these symbols will be replaced by members from set of terminal symbols T. To replace non-terminals by terminals a production rules set P is used. The starting symbol S is a member of non-terminal set (Hopcroft and Ullman, 1979).

During the population evaluation, the individuals, presented by chromosome created by a vector of integers, are translated into computer programs. The chromosome values are used in sequence to select production rules for rewriting non-terminal symbols of supplied grammar. The resulting program is executed and its performance is evaluated.

The main advantage of using grammatical evolution over a general genetic algorithm is that the grammar represents a layer between genetic algorithm and the given problem. We need only to

specify a domain specific grammar and a fitness function to evaluate individuals to solve a problem.

Techniques which use grammatical evolution or other methods based on genetic algorithms as a learning process are being investigated in last years (Ciesielski *et al.*, 2003; Škorpil *et al.*, 2012). One example of learning is a task to create a classifier. Classification is a process which takes a sample of unknown class and tries to label it with one of known class labels. A classifier is a tool to perform this operation and its quality is judged by its performance and accuracy. We distinguish between binary classification and classification into multiple classes.

Grammatical evolution is usually capable to create computer programs in a form of trees. Non-leaf nodes in a tree represent mathematical operations or other transformations of their successor nodes and leaf nodes are usually numeric constants or are mapped on inputs. The tree program is executed and the root node returns one output value.

Program output with only one value is not very convenient for multi-class classification but is well suited for binary classification tasks. For binary classification we can use, for example, positive and negative result of program to distinguish between two classes. We also tried to divide the output range into automatically selected intervals to perform multi-class classification with some success (Lýsek *et al.*, 2012; Škorpil *et al.*, 2012). The constraint which was placed on grammatical evolution process to produce individuals with non-overlapping intervals was very hard to be fulfilled. Often the process was not capable of finding good classifier and the intervals overlapped.

Due to tree form of program and one output value, it is not trivial task to design a system with vector output like neural network classifiers have, where each value of a vector is used for corresponding class index. Such output is desirable for a classifier agent. This leads us to a discovery that tree structure of a classifier is not suitable for multi-class classification task. On the other hand, tree structure is a convenient form of program representation for performing crossover and mutation operations in grammatical evolution. In following section of this article a solution based on new type of node that can be used in the tree program is presented.

The computational complexity of evolutionary methods is certainly higher than complexity of training a neural network or some data mining procedure. That is a disadvantage of evolutionary methods given by their nature of evaluating large populations of individuals. The amount of individual's evaluations can be calculated by multiplying the amount of individuals and the size of population. In comparison with neural network training, where the amount of evaluations is equal to amount of iterations, evolutionary methods require much more computational power.

METHODS AND RESOURCES

We used data from economic survey about customer behaviour in food market in Czech Republic. These data were gathered in a survey performed by Research Department of Marketing and Sales at the Faculty of Business and Economics of Mendel University in Brno. The same data were used in work (Šťastný *et al.*, 2011) where MLP neural network and Kohonen neural network were used.

Data contained response sets of participants on questions about their behaviour when purchasing food. There was also information of participant's age interval which we used as a class label for training the classifier. Response for each question was a value from a set of given possible values. There were 36 responses for each customer response set.

A conversion of collected survey responses to numerical values of approximately equal weights was needed. Also incomplete response sets were removed from the data set. These removed response sets were not used in our tests at all. This pre-processing was already performed in research performed by (Šťastný *et al.*, 2011) and we used exactly same data. As the responses of the survey suffered from noise, we removed those response sets which did not have the same class label on another response set closest to it. Euclidean distance to calculate the distance between two response sets was used. This step helped the learning process to build a classifier as the response sets with similar patterns are equally labelled.

Initially the input data had 2092 responses. By filtering incomplete response sets we obtained 1880 response sets. Removal of similar response sets with different classes gave us 1033 response sets. The

I: *Amount of samples for each class*

Class title (age of survey participant)	Amount of samples (without incomplete)	Amount of samples (filtered from outliers)
12 to 17	94	17
18 to 24	634	427
25 to 34	293	99
35 to 54	543	323
55 to 64	126	24
65 and more	190	143
sum	1 880	1 033

```
<start> ::= <expr> | <branching> | <start><start>
<expr> ::= <register_write> | <expr><expr>
<register_write> ::= register_write_0(<math>); ... register_write_5(<math>);
<register_read> ::= register_read_0() ... register_read_5()
<branching> ::= if(<logic>) {<expr>} else {<expr>}
<logic> ::= equal(<math>,<math>) | lgtn(<math>,<math>) | smtn(<math>,<math>) |
            l_and(<logic>,<logic>) | l_or(<logic>,<logic>) | l_xor(<logic>,<logic>)
<math> ::= <input> | <number> | <register_read> | <math_op>(<math>,<math>)
<number> ::= -100.0 ... 100.0
<input> ::= vector_input_0() ... vector_input_35()
<math_op> ::= add | sub | mul
```

1: *Grammar used for classifier program creation*

amounts of response sets for each class label are presented in Tab. I.

In some classes a significant drop of response sets can be observed. This denotes inconsistency in behaviour of survey participants which can be caused by different participant social category or it can mean that the class labelling is too coarse.

To train a classifier we used our grammatical evolution framework created in Java programming language (Lýsek *et al.*, 2012). To translate individual's chromosome to computer program we use backwards processing presented in (Ošmera *et al.*, 2006). Grammatical evolution uses vector of marks to perform crossover and mutation (Popelka and Šťastný, 2009). We also employed new equalisation operator to control code bloat (Silva, 2011) which selects new population members by their length. We used amount of tree nodes to measure program length.

Tree representation of program was used. Our framework can work with such structures when performing crossover and mutation operations. We employed a new operator which we called semicolon. This operator enables us to simulate sequential program behaviour. Thanks to the new node, we were able to create a vector of output values where each value corresponds to one of the classes. This was implemented by a register node which is able to store certain value produced by its sub nodes.

Values from register nodes are collected after program execution and the position of maximal value is used to select label of unknown input vector.

Fig. 1 shows grammar used for chromosome translation.

Fitness function was designed to work with relative classification success rate over each class as there were unequal amounts of samples in each class. Another factor included into fitness was amount of classes the program could recognize, not regarding the success rate in that class. This is included in the fitness formula because there is no other way to control how the chromosome of individual is translated and we needed a pressure on the evolution process to produce classifier with ability to recognize all classes of samples. Without this part of fitness formula, the evolved programs ignored some classes. Usually smaller classes were ignored and we think that this was caused by higher difficulty of finding a suitable classification pattern on a small number of samples.

The fitness value of an individual was calculated by following formula:

$$f = w_1 \times ASR + w_2 \times \frac{RCC}{CC}$$

$$w_1 = 0.9$$

$$w_2 = 0.1$$

ASR....average success rate for all classes
RCC....amount of classes recognized by classifier
CC.....amount of classes
w_1, w_2..weights

The presented formula was designed for this application and the weight values were selected after a set of experiments.

RESULTS AND DISCUSSION

The programs created by grammatical evolution process can have different form each time. In Fig. 2 we present one program evolved by our system.

Evolved programs have a simple structure and we can actually see the dependence of output values on input values. The success rate was usually around 65% but we were able to achieve even 75% on training data set. The parameters of evolutionary process set for our experiments follows:

- crossover rate: from 0.7 to 0.5
- mutation rate: from 0.1 to 0.5
- structural mutation rate: 0.5
- max. amount of samples from each class for fitness evaluation: 50
- training sample renewal period: 1 iteration
- max. chromosome length: 120
- elitism: YES
- population size: 200
- stopping criterion: max 10000 iterations.

For each iteration of genetic algorithm we selected randomly up to 50 samples from each class for fitness evaluation. This speeded up the process as we did not have to calculate the performance on the whole data set for each individual. Training data switching also helps preventing of being stuck

```
if(smtn(vector_input_33(),mul(vector_input_34(),add(vector_input_11(),vector_input_34())))) {
   register_write_3(vector_input_22());
   register_write_5(register_read_4());
   register_write_4(add(vector_input_34(),add(vector_input_3(),mul(register_read_3(),-1.0))));
   register_write_2(
    add(
     sub(-24.0,vector_input_14()),
     mul(add(-40.0,mul(vector_input_34(),vector_input_34())),vector_input_22())
    )
   );
   register_write_3(vector_input_28());
   register_write_1(vector_input_33());
  } else {
   register_write_0(vector_input_31());
   register_write_0(add(vector_input_34(),add(vector_input_11(),vector_input_34())));
   register_write_5(vector_input_15());
  }
```

2: *One of classifier program created during our experiments*

II: *Confusion matrix of presented classifier program for filtered dataset*

	12 to 17	18 to 24	25 to 34	35 to 54	55 to 64	65 >
12 to 17	13	4	0	0	0	0
18 to 24	56	312	29	22	6	2
25 to 34	0	5	46	41	3	4
35 to 54	0	0	80	199	34	10
55 to 64	0	0	2	4	11	7
65 >	0	0	0	0	4	139

III: *Confusion matrix of presented classifier program for non-filtered dataset*

	12 to 17	18 to 24	25 to 34	35 to 54	55 to 64	65 >
12 to 17	36	58	0	0	0	0
18 to 24	87	432	48	45	7	15
25 to 34	6	15	119	117	16	20
35 to 54	0	4	150	298	52	39
55 to 64	0	0	30	25	23	48
65 >	0	0	2	2	9	177

in local extreme. The main reason is that in the beginning of the process, each iteration has different form of best individual and as the evolutionary process continues, the individuals mix to form more universal classifiers.

We tested the execution of created classifier in another environment. The ability of grammatical evolution to create syntactically correct programs for any programming language was used. We selected PHP scripting language. The functions used by the evolved program were implemented into PHP script. A platform for executing and analysing the program was obtained by this without need to write our own parser.

The measurements of classification results for selected classifier program success rate follows in Tab. II. and Tab. III. The classification success rate for selected individual was 69.7% resp. 57.71% for filtered data resp. not filtered data. This is an increase of about 15% in comparison with MLP network used in (Šťastný *et al.*, 2011).

The comparison with Kohonen neural network, which was also used in (Šťastný *et al.*, 2011), is difficult to make because that type of network is based on grouping similar inputs to one of its output nodes according to trained connection weights. That is a completely different approach. We can state that in most cases the misclassification made by our program places the result into neighbouring class – that behaviour is similar to Kohonen network behaviour.

SUMMARY

We presented a method which uses grammatical evolution to create a multi-class classifier program. The grammar we used enables us to create programs in a common tree form with multiple output values. The short classifiers programs we created behave like a sequential programs but the tree structure is preserved to simplify the crossover and mutation operators.

We were able to train a classifier using a dataset with different sample amount for different classes. Also we presented used grammar and the performance of classifiers created by grammatical evolution is compared to classification performed by two types of artificial neural networks.

The benefit of using grammatical evolution is that we can easily analyse the program thanks to a human readable result and we can gain an insight of relations between input data and classification, also the structure of the program is optimised. Other benefit is the grammatical evolution capability to create programs which are syntactically correct for selected programming language and we can instantly employ created classifier program in another development project.

The success of classifier creation process also depends on the dataset used for classifier training. To train a multi-class classifier on a data from survey research with heavy noise is a difficult task. However the grammatical evolution was able to create a classifier with good performance and gained higher success rate in classification than classical methods or neural networks. Although the amount of calculations and time the process took was certainly higher.

REFERENCES

CIESIELSKI, V. B., ANDREAE, P., ZHANG M., 2003: A Domain-Independent Window Approach to Multiclass Object Detection Using Genetic Programming. *EURASIP Journal on Applied Signal Processing*, 2003, 8: 841–859. ISSN 1687-6180.

HOPCROFT, J. E., ULLMAN, J. D., 1979: *Introduction to Automata Theory, Languages and Computation.* Boston: Addison-Wesley, 521 p. ISBN 978-0201441246.

KOZA, R. J., 1992: *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge: The MIT Press, 819 p. ISBN 0-262-11170-5.

LÝSEK, J., ŠŤASTNÝ, J., MOTYČKA, A., 2012: Object Recognition by means of Evolved Detector and Classifier Program. In: *18th International Conference of Soft Computing MENDEL 2012*, Brno: Brno University of Technology, pp. 82–87. ISSN 1803-3814.

OŠMERA, P., POPELKA, O., PIVOŇKA, P., 2006: Parallel Grammatical evolution with backward processing. In: CHEAH, C. C., *9th International Conference on Control, Automation, Robotics and Vision*. Singapore, New York: IEEE, pp. 949–954, ISBN 978-1-4244-0341-7.

POPELKA, O., ŠŤASTNÝ, J., 2009: *Uplatnění metod umělé inteligence v zemědělsko-ekonomických predikčních úlohách*. Brno: Mendelova univerzita v Brně, 48 p. ISBN 978-80-7375-340-5.

RYAN, C., O'NEILL, M., 2003: *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Boston: Kluwer, 144 p. ISBN 978-1402074448.

SILVA, S., 2011: Reassembling Operator Equalisation – A Secret Revealed. *SIGEVOlution*, 5, 3: 10–22, ISSN 1931-8499.

ŠŤASTNÝ, J., TURČÍNEK, P., MOTYČKA, A., 2011: *Using Neural Networks for Marketing Research Data Classification*. Mathematical Methods and Techniques in Engeneering & Environmental Science. Catania, Sicily, Italy: WSEAS Press, pp. 252–256. ISBN 978-1-61804-046-6.

ŠKORPIL, V., LÝSEK, J., MOTYČKA, A., CEPL, M., ENDRLE, P., 2012: Grammatical Evolution as a Learning Process for Multiclass Object Detection. In: *Recent Researches in Communications and Computers*, Kos, Greece: WSEAS Press, pp. 101–105. ISBN 978-1-61804-108-1.

Address

Ing. Jiří Lýsek, prof. RNDr. Ing. Jiří Šťastný, CSc., Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, 613 00 Brno, Czech Republic, e-mail: jiri.lysek@mendelu.cz, jiri.stastny@mendelu.cz