Machine Learning Methods and “Real-Time” Economics

Volodymyr Dorozhinsky[1], Arsen Khadikov[1], and Nina Zholtkevych[2]

School of Math. and Comp. Sci., V.N. Karazin Kharkiv National University[1]
School of Economics, V.N. Karazin Kharkiv National University[2]
vdorozhinsky@gmail.com

Abstract. In the paper some machine-learning method for synthesis an on-fly event acceptor is proposed. Such an acceptor has become necessary for the technique of the on-fly event processing. This technique is increasingly being used in the design of information systems of economic destination. In the paper the synthesis problem for such an acceptor is considered in a rigorous mathematical formulation. The method for solution of the problem and the computer experiment to study the method are described. The directions for the further research are proposed.

Keywords: on-fly event processing, acceptor, regular language, prefix-free language
Key Terms: Component, MathematicalModel, MachineIntelligence

1 Introduction

Modern Information Communication Technology (ICT) causes intensive changes in, practically, all areas of human activity. These changes, in particular, impose non-traditional restrictions for decision processes in the field of economy. New approaches for modelling economic processes such as “econophysics” [6] are responses on these challenges in the information society. The key reason of a huge number of the changes is a reduction of the time scale for decision making to manage of economic processes. This reduction can be argued that business management systems gain more and more features inherent in real-time control systems for complex technical objects. Thus, we may talk about real-time economic processes and use the corresponding technique to analyse and design them on the base of ICT. Today Event-Driven Architecture (EDA) is the generally accepted architecture solution to construct an information system that is scalable, adaptable, and capable to operate in real-time [3]. Mathematical fundamentals to analyse component behaviour of such a system have been established in the papers [1, 7, 8]. The practically important class of message detectors for systems built on EDA has been defined and considered in [2]. The implementation of these ideas and theoretical results in the development process of information systems for business a priori constrained by the complexity of identifying message patterns of system events. In such situations, the usage of the machine learning methods is one possible way to overcome the complexity of the problem in progress of solving it. The advantage of this approach is needlessness to construct a general theory, covering all logically possible occasions. Instead, it creates a mechanism to adapt the system to new situation that is not consistent with the current knowledge of the system. The main goal of the paper is to present the principal suggestions of our approach and the preliminary results.

2 Basic Mathematical Model and Problem Statement

Here we describe the model proposed in [7] for a component of a system based on EDA.

2.1 Basic Notation and Definitions

Below we use the following notation:

$f\colon X \dashrightarrow Y$	denotes that f is a partial mapping from X into Y;
$f(x)\uparrow$	denotes that $f(x)$ is not defined for the member x of X;
$f(x)\downarrow$	denotes that $f(x)$ is defined for the member x of X;
$f(x) \downarrow= y$	denotes that $f(x)\downarrow$ and $y = f(x)$ for the member y of Y;
$\boldsymbol{\epsilon}$	denotes the empty (zero-length) sequence;
X^+	denotes the set of all non-empty finite sequences composed of elements of X;
X^*	denotes the set $\{\boldsymbol{\epsilon}\} \bigcup X^+$;
X^ω	denotes the set of all infinite sequences composed of elements of X;
X^∞	denotes the set $X^* \bigcup X^\omega$;
$\lvert\boldsymbol{x}\rvert$	denotes the length of the finite sequence $\boldsymbol{x}$;
$x[0]$	denotes the first element of a finite or infinite sequence $\boldsymbol{x}$;
$\boldsymbol{x}[1:]$	denotes the sequence obtained by removing the first element of the sequence $\boldsymbol{x}$.

Definition 1. *For a finite set X a subset L of X^* is called a language and, in the context, X is called an alphabet, and its members are called symbols.*

We interpret symbols as a prime messages informing that the corresponding elementary event has happened. Some finite sequences of symbols inform about complex events and below these sequences are called events. Other finite sequences of symbols do not carry information about complex events and below we call them words. Sets of complex events must meet the certain conditions. The most important of these conditions is that any stream of elementary events is uniquely subdivided into a series of complex events by directed viewing the stream from left to right. This condition leads to the requirement that L is prefix-free [7]. Now we can require that any set of complex events related with a system would be prefix-free.

2.2 Mathematical Model

Let us briefly remind the principal tenets of the used model. Any component of a system based on EDA is modelled by using the concept of a ***CEP-machine***.

Definition 2 (Structure of CEP-machine). *Any CEP-machine is a quintuple $\boldsymbol{M} = (X, Y, H, h_0, \alpha)$ with the following constituents:*

- *the* ***alphabet of atomic messages*** *X, which is a finite set;*
- *the* ***alphabet of machine responses*** *Y, which is a finite set;*
- *the* ***set of handlers*** *H, which is a finite set, whose each member is a partial mapping $h\colon X^+ \dashrightarrow Y$ such that its domain is a prefix-free language;*
- *the* ***initial handler*** *h_0, which is some fixed element of H;*
- *the* ***response function*** *α, which is a mapping with domain Y and codomain H.*

The general behaviour of any CEP-machine is described in [7]. However in the general case it is possible that a CEP-machine can become hung while trying to recognize an event. But in our study we consider simpler case, which was first considered in [2]. In this case the situation, when the CEP-machine is hanging, is impossible. To specify this special case we are in need in the following definition.

Definition 3 (Regular handler). *A handler $h\colon X^+ \dashrightarrow Y$ is called regular if there exist some finite set Z with the marked element $z_0 \in Z$*

and some mapping $\delta\colon Z \times X \to Z \bigcup Y$ such that for any $\boldsymbol{u} \in X^+$ and $y \in Y$ the condition $h(\boldsymbol{u}) \downarrow= y$ is fulfilled iff there exist $z_1, \ldots, z_{|\boldsymbol{u}|-1} \in Z$ such that

$$\begin{aligned} z_{i+1} &= \delta(z_i, u[i]) \quad \textit{for} \quad 0 \le i < |\boldsymbol{u}| - 1 \quad \textit{and} \\ y \ &= \delta(z_{|\boldsymbol{u}|-1}, u[|\boldsymbol{u}| - 1]). \end{aligned}$$

In this case we say that the handler h is realized by the triple (Z, z_0, δ).

Remark 1. It is evident that a handler is regular if there exist a finite state machine realising it.

Definition 4 (Regular CEP-machine). *A CEP-machine is called regular if all its handlers are regular.*

2.3 Problem Statement

In practice we propose restrict our technique by methods of synthesis for regular CEP-machines. This restriction is caused by technical and theoretical difficulties of more general technique. Further, it is evident that the synthesis problem for a regular CEP-machine is decomposed into series of synthesis problems for regular handlers each of which has only one possible response "`accepted`". In this case we use the term "a regular acceptor" instead the term "a regular handler". Thus, a machine-learning problem for synthesis process of a regular handler can be formulated in the following manner.

Problem. Let $E = \{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_M\}$ be a finite prefix-free set of finite sequences composed by elements of X and $C = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N\}$ be a finite set of finite sequences composed by elements of X such that $E \bigcap C = \varnothing$ then we interpret elements of set E as examples and elements of set C as counterexamples; we need to find a regular acceptor $h\colon X^+ \dashrightarrow \{\texttt{accepted}\}$ such that

1. $h(\boldsymbol{u_i}) \downarrow= \texttt{accepted}$ for all $0 \le i < M$;
2. $h(\boldsymbol{v_i})\uparrow$ for all $0 \le i < N$; and
3. the corresponding set Z has the least number of elements among all possible.

3 Progress Review

Here we present a method to build a regular acceptor and describe a computer experiment that instills confidence in the existence of a mathematical justification for this method.

3.1 Some Theoretical Background

We premise our presentation with a few simple theoretical results. The techniques used to prove these results are quite common in automata theory therefore we omit the corresponding proofs for simplicity of the presentation. First of all, let us return to Remark 1 and discuss more detail the interrelation between regular acceptors and finite state machines. Namely, if we consider for any regular acceptor $h : X^+ \dashrightarrow Y$ that realized by the triple (Z, z_0, δ) the machine $(Q = Z \bigcup Y \bigcup \{q_{trap}\}, X, \overline{\delta} : Q \times X \to Q, q_0 = z_0, Q_{accept} = Y)$ where $q_{trap} \notin Z \bigcup Y$ and $\overline{\delta}(y, x) = q_{trap}$; $\overline{\delta}(q_{trap}, x) = q_{trap}$ for any $x \in X$ and $y \in Y$ then the regular language accepted by this machine coincides with the set of events accepted by the regular acceptor. After this brief theoretical review, we can return to our problem.

3.2 Proposed Method

The general view of the method to find an acceptor is presented by Alg. 1. This method consists of the series of redirections for acceptor transitions leading into the trap starting with the minimal acceptor for the set of examples. Two functions used by the algorithm `init` and `modify` are specified separately.

To complete the specification of the proposed method we need to describe algorithms for the function `init` (see item 3 of the Alg. 1) and for the function `modify` (see item 7 of the Alg. 1).

Function `init`. To build the minimal acceptor the following ideas are used:

1. states of the acceptor are defined recursively as special sets of words;
2. we choose the set E as z_0 and add it to Z;
3. we choose the empty set as `trap`;
4. if for $x \in X$ in E there is not a word with the first symbol x then assign $\delta(z_0, x) = \texttt{trap}$ else the set $\{\boldsymbol{u} \in X^* \mid x\boldsymbol{u} \in E\}$ add to Z;
5. repeat recursively this consideration for all member of Z until Z is stabilized;
6. denote the set $\{\boldsymbol{\epsilon}\}$ by `accepted`.

The acceptor obtained in this manner is assigned as a result of the function `init`.

Algorithm 1. Specification of the proposed method

```
1  def learning_method($E$, $C$):
       Require : the finite prefix-free set of events $E$
                 the finite set of words that are not events $C$
       Ensure  : the required acceptor
2      do that:
3          initiate the learning process by applying function init to the set $E$
             and denoting the result by acceptor
       # initialize the set of transitions that cannot be redirected
4      frozen_transitions = set()
5      while halting condition is not fulfilled:
6          do that:
7              modify acceptor by redirecting a transition leading into the trap
                 and minimize the resulting acceptor wherein the redirected
                 transition cannot belong to frozen_transitions
8          do that:
9              check that acceptor is admissible in the sense that it does not
                 accept any word from $C$
10         if the checking is successful: continue
11         else:
12             do that:
13                 roll-back the modification and add the redirected transition
                     into frozen_transitions
```

Function `modify`**.** To select a transition for redirection we use the following simple remark: the minimal regular acceptor has at most one state that is an attractor, i.e. any transition that goes out from this state has this state as a target. Moreover, if this acceptor accepts a finite language then the existence of the attractor is guaranteed. Further, to minimize the new acceptor we use standard Hopcroft's algorithm [4].

3.3 Computer Experiment Schema

Thus, we assume that the described above method gives a solution of our problem. To check reasonability of this assumption we designed the computer experiment for searching counterexamples to the assumption. The Alg. 2 specifies the schema of the experiment.

3.4 Case Study

To implement the mentioned experimental schema we have used language Python with libraries "scipy" and "numpy" [5]. Particularly, all random

Algorithm 2. The schema of the computer experiment

```
for _ in range(given_number_of_experiments):
    do that:
        generate randomly a regular acceptor acceptor
    do that:
        generate randomly sets $E$ and $C$ using acceptor
    acceptor′ = learning_method($E$,$C$)
    do that:
        compare acceptor′ and acceptor
```

choices have been provided by the standard function `random.choice` contained in the library “numpy”. To randomly generate a regular expression the following schema has been used. An expression is represented by a syntactic tree. Each leaf of this tree is marked by a token and each internal node of the tree is marked by a functor. Moreover, the number of children for an internal node equals the arity of the corresponding functor. The recursive structure of the syntactic tree that represents some regular expression indicates the way of this tree random generation. To make decision whether the root of the current subtree is an internal node or a leaf we propose to use the following function $p(n)$ that determines the conditional probability to mark the current node as internal if its depth is equal to n

$$p(n) = \begin{cases} 1 - \frac{1}{2}\left(\frac{n}{\mu}\right)^2 & \text{if } 0 \leq n < \mu \\ \frac{1}{2}\exp\left(1 - \frac{n}{\mu}\right) & \text{if } n \geq \mu \end{cases} .$$

The results of the more than 10,000 experiments have shown that for a randomly generated acceptor with the obtained sets E and C, the presented method of learning was restoring this acceptor using these sets. Thus, we can assume that the proposed method is precise on regular acceptors. The last assumption can be considered as evidence in favour of the validity of the proposed method of machine learning.

4 Conclusion and Future Study

The presented paper cannot be considered as a complete research. The obtained results are preliminary, but they are very important because they demonstrate a chance to substitute the complete logical analysis of situation by the learning on the examples and counterexamples during software development of the on-fly processing systems. Also these results make the need for further research in the following directions: future experimental study of the proposed method in order to clarify the boundaries of its applicability; finding rigorous formulations of the method convergence conditions; mathematical justification of the method; evaluation of the effectiveness of the method.

Acknowledgement. The authors thank Prof. Gregoriy Zholtkevych for the idea proposed to them: to use machine learning methods for the synthesis of the logical structure of the on-fly event processing mechanism.

References

1. Dokuchaev, M., Novikov, B., Zholtkevych G.: Partial Actions and Automata. Alg. Discr. Math. 11(2), 51–63 (2011).
2. Dorozhinsky, V.: Regular Complex Event Processing Machines. Systemy obrobky informacii. 8, 82–86 (2015).
3. Etzion, O., Niblett, P.: Event Processing in Action. Manning Publications (2010).
4. Hopcroft, J.: An $n \log n$ algorithm for minimizing states in a finite automaton. In: Proc. Internat. Sympos., Technion, Haifa. Theory of machines and computations, pp. 189–196. Academic Press, New York (1971)
5. Scientific Computing Tools for Python. `http://scipy.org/`
6. Stanley, H.E.: Interview on Econophysics. Published in: "IIM Kozhikode Society & Management Review", Sage publication (USA). 2(2), pp. 73–78 (2013) `http://www.saha.ac.in/cmp/camcs/Stanley-interview.pdf`.
7. Zholtkevych, G., Novikov, B., Dorozhinsky, V.: Pre-Automata and Complex Event Processing. In: V. Ermolayev et al. (eds.) ICTERI 2014. CCIS, vol. 469, pp. 100–116. Springer International Publishing, Switzerland (2014).
8. Zholtkevych, G.: Realisation of Synchronous and Asynchronous Black Boxes Using Machines. In: V. Yakovyna et al. (eds.) ICTERI 2015. CCIS, vol. 594, pp 124–139. Springer International Publishing, Switzerland (2016).