XML TECHNOLOGIES IN COLABORATION WITH MACHINE LEARNING ALGORITHMS FOR BUSINESS DECISIONS

CRISTINA OFELIA STANCIU[1], ADRIAN COJOCARIU[1], LJUBICA KAZI[2]
[1]"TIBISCUS" UNIVERSITY OF TIMIŞOARA, FACULTY OF ECONOMIC SCIENCE, DALIEI STR, 1/A, TIMIŞOARA, 300558, ROMANIA
[2]UNIVERSITY OF NOVI SAD, TECHNICAL FACULTY "MIHAJLO PUPIN" ZRENJANIN, ĐURE ĐAKOVIĆA BN, 23000 ZRENJANIN SERBIA
ofelia.stanciu@gmail.com, a_cojocariu@yahoo.com, ljubicakazi@ptt.rs

Abstract:
XML provides an organized and elegant way of storing data, the main advantage is adaptability, and also that XML modeled data are readable by any user. Machine Learning algorithms, particularly decision tree generating algorithms and Reinforcement Learning algorithms can be applied upon data stored using XML technologies, in order to support business decisions.

Key words: *XML, Machine Learning, decision algorithms*

JEL classification: *D80, M15*

The management and use of information, as an essential resource of a company, acquires new particularities deriving from its use in supporting decisions and from the continuous growth of the complexity of the decision process.

The decision process is a an assemble of activities driven by only one or several individuals facing an event that generates more than one acting direction, following the optimal direction, according to the value system of the decision makers.

Data management is a very important aspect of the decision process, and as the data amount is continuously enlarging, one has to find proper solutions to store and manage data. XML (eXtensible Markup Language) offers an organized and elegant way to store data. Its main advantage is the adaptability, and the fact that data modeled with XML are human readable, which in many cases proves to be a great advantage.

1. Data modeling and management with XML

XML is a modern format and most if the visual high level programming languages, such as Visual C++, C#, Java, are capable to process and manage XML files and the data stored in these files.

An example for data modeling using XML is presented as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<Relation xmlns="urn:localhost:ML.Relation">
  <Attributes>
    <Attribute name="outlook" type="enumerated">
      <EnumeratedValue value="sunny" />
      <EnumeratedValue value="overcast" />
      <EnumeratedValue value="rainy" />
    </Attribute>
    <Attribute name="temperature" type="integer" >
    <Attribute name="humidity" type="real" />
    <Attribute name="windy" type="boolean" />
    <Attribute name="play" type="enumerated" >
```

```
            <EnumeratedValue value="yes" />
            <EnumeratedValue value="no" />
        </Attribute>
    </Attributes>
    <Instances>
        <Instance outlook="sunny" temperature="85" humidity="85"
                windy="false" play="no" />
        <Instance outlook="sunny" temperature="80" humidity="90"
                windy="true" play="no" />
        <Instance outlook="overcast" temperature="83" humidity="86"
                windy="false" play="yes" />
        <Instance outlook="rainy" temperature="70" humidity="96"
                windy="false" play="yes" />
        <Instance outlook="rainy" temperature="68" humidity="80"
                windy="false" play="yes" />
        <Instance outlook="rainy" temperature="65" humidity="70"
                windy="true" play="no" />
        <Instance outlook="overcast" temperature="64" humidity="65"
                windy="true" play="yes" />
        <Instance outlook="sunny" temperature="72" humidity="95"
                windy="false" play="no" />
        <Instance outlook="sunny" temperature="69" humidity="70"
                windy="false" play="yes" />
        <Instance outlook="rainy" temperature="75" humidity="80"
                windy="false" play="yes" />
        <Instance outlook="sunny" temperature="75" humidity="70"
                windy="true" play="yes" />
        <Instance outlook="overcast" temperature="72" humidity="90"
                windy="true" play="yes" />
        <Instance outlook="overcast" temperature="81" humidity="75"
                windy="false" play="yes" />
        <Instance outlook="rainy" temperature="71" humidity="91"
                windy="true" play="no" />
    </Instances>
</Relation>
```

Modeling data in XML format also offers the possibility to validate data by defining so called XML schemes. The XSD (*XML Schema Definition*) files define rules and patterns the XML file should fit in. The assemble made up by the data models represented in XML format together with the XSD schemes and the applications developed in programming languages that offer function libraries for processing these data models (.Net, Java packages, Qt) represents a powerful and efficient solution, but mostly an elegant one, according to the object oriented programming point of view.

There are two main methods for parsing XML files: SAX and DOM. Each of these methods has its own advantages:

- SAX (*Simple API for XML*) is an event based method: at the parsing moment of each XML entity of the model a signal is output, signal which is received by the software developer that uses the SAX functions and will be able to realize the internal model of the data from the XML file. Among the main particularities of a SAX parser, we can mention the fast processing and simplicity.
- DOM (*Document Object Model*) is based on loading the whole content of the XML file in the memory and on the disposal of its elements to the software

developer in the tree model which is typical to the format. A derived advantage of this parsing way is that in any moment one can access a node of the XML file through the functions and data types offered by the DOM function library implemented in a certain programming language. Among the main characteristics of DOM parser we mention the model hierarchy and the easy serialization.

In particular, considering an integrated decision support system, the XML format fits perfectly, as it can easily represent tree structures, is easy to use and does not make problems in structure altering, is flexible, but proves to have one major disadvantage, which is when it comes to processing very large data quantity.

Storing data for a decision support system in XML format can be a valid solution. Similar to the way that Weka – a well known open source Data Mining project – stores data in plain text files, in the XML files we first define the used attributes and their type, then the actual data, the instances, each of them representing an enumeration of the values of the previously defined attributes. The lack of some attributes from the definition of an instance shows a missing value which will be treated differently by the decision algorithms. The improvements, reported to the plain text format implemented by Weka, are that the XML file can be validated when it is loaded, using an XML validation scheme and the functions offered by the class library which also includes the XML parser. A validation can be made also when the file is written on the disk, in order to verify if the file will be correctly written and afterwards will also be validated when it will be loaded.

Decision trees are an easy applicable for classification and prediction, the result being presented in a tree form with automatically set logic rules hierarchy by exploring a set of examples. The examples are similar to records with several attributes and the rules are being established by a detailed dividing of the assemble of examples, depending on the content of the attributes.

Building the decision tree begins from its root, which shows the available examples. The initial assemble is divided into intermediary nodes. Each node is being evaluated and is divided in other nodes if possible, until the terminal undividable nodes are found. When effectively processing data, the attributes are grouped in two categories: dependent and independent attributes. There is only one dependent attribute, also named *target* attribute, onto which we are searching for influences from the other attributes, the independent ones. From all the independent attributes we select the one that has the most powerful impact on the target field, the one that eventually supports the division of the record assembly into the most relevant sub-assemblies. For each of these subdivisions we re-perform the analysis, using the same target field but considering only the attributes that were left out in the previous steps, and seeking for new subdivisions.

After the tree construction, the new data can be included, at some amount of certainty, into one of the leaf nodes, depending on their attributes values, classifying them or being able to perform predictions regarding them.

One of the most popular methods used in developing decision trees is CART (Classification and Regression Trees). This one starts with seeking out the independent variable which has values that allow the best division to take place. For this we move on to calculating a diversity index for the whole record assembly given an attribute. The procedure describes the parse of the independent attributes one by one and the evaluation of the diversity decrease obtained by the division one would make based on it. The variable that is retained as separation criteria is the one that produces the best results. We are actually looking for a binary tree. The attributes that have multiple values raise, in these conditions, a supplementary issue: regrouping the values as so the final division will lead to only two subdivisions.

A more recent algorithm is C4.5 proposed by the Australian professor Quinlan. Unlike CART, which generates only binary trees, a node can have here a variable number of branches. Another difference would derive from the treatment of nominal variables, which will now have one branch for each possible value. The precursor of this algorithm, the ID3, developed by the same author, enjoyed a vast popularity and was used in various computer products. This one uses for an evaluation criteria of divisions the information gain obtained, as well as the uncertainty degree removed, concept that derives from Shannon's information theorem.

Because its usage comes in favor of numerous branches to which a small number of records from the example set will correspond, C4.5 uses the ratio between the total information gain obtained by the corresponding division and the information gain that is only due to the sub-assemblies count it generates. The tree pruning is also made in a different manner that the one CART practices; the analysis is based also on the teaching data, without having to invoke the test or evaluation distinct data.

In its informatics representation, the C4.5 can automatically generate rules. Beginning with the complete set, generated directly based on the tree, the application follows a generalization chain meant to decrease the rules count. For this purpose we remove certain conditions for each given rule and we verify how this maneuver increases the error rate. A series of other transformations can also be operated for this goal, as so, in the end, the rule count can be smaller than the leaf count.

Decision trees are a standard Data Mining tool and many of them are available in the C4.5 package. Decision trees are generally preferred due to the comprehensivity of their hypothesis and the efficiency of their learning and evaluation.

Decision trees are usually binary trees with simple classifiers associated to each internal node and with a classification associated to each leaf. In order to evaluate a *T* tree for an input *x*, the *x* attribute will be given to each classifier. The outputs of simple classifiers associated to the nodes determine a unique path from the root to a certain leaf of the decision tree.

Decision trees are generally understood through a descendant development procedure that begins with the root node and chooses a part of the data that maximize a cost function, usually a measurement of the subassemblies' "impurities" that is implicit defined in the moment of data partitioning. Afterwards the subassemblies are associated to two decision trees. The procedure is recursively applied to the child nodes and the tree is being enlarged until a stop condition is met.

The C4.5 algorithm only generates binary trees, a node having a variable number of branches. C4.5 is able to automatically generate rules. Starting with the complete set of rules, generated directly on the tree's basis, there will be a generalization part, in order to reduce the number of rules. This way, for each rule certain conditions are eliminated and the increase of the error rate must be verified. A lot of other transformations can also be made in order to decrease the number of rules.

The steps of the C4.5 algorithm are presented as follows:

1) selects the attributes that have proven the most quantity of gained information

2) given two classes P and N:

a) given a set of examples S that contain p elements from the P class and n elements from the N class

b) the information quantity required to decide if a random example from S belongs to the P or N class is defined in relation (1):

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n} \quad (1)$$

3) assuming that by using an attribute A, a set S will be partitioned in the following sets {S1, S2, ...,Sv}

a) if S_i contains p_i examples from P and n_i examples from N, then the entropy or the necessary information for the classification of all the objects from all the S_i trees is (2):

$$E(A)=\sum_{i=1}^{n}\frac{p_i+n_i}{p+n}I(p_i,n_i) \quad (2)$$

4) the codification information gained on the A branch would be:
Gain(A) = I(p, n) - E(A)

C4.5 is an algorithm for the induction of decision trees, being, as mentioned before, an extension of the ID3 algorithm which unlike C4.5 solves some problems such as data extra-matching, treating continuous attributes and attributes with missing values, increasing the computational efficiency. The C4.5 generates a decision tree by recursively partitioning the data amount, using a depth-first parsing strategy. The algorithm takes into consideration all the possible tests for partitioning the data and selects the tests that will lead to the best information gain.

Considering the entropy concept as the "impurity" of a set of training examples S, the efficiency of an attribute for classification of these examples can be estimated. The information gain measures the expected reducing of the entropy caused by partitioning the set according to the values of an attribute A (3).

$$IG(S,A)=H(S)-\sum_{v\in Val(A)}\frac{card(S_v)}{card(S)}H(S_v) \quad (3)$$

where *Val*(*A*) is the set of A attribute values, *Sv* is the subset of *S* for which the *A* attribute has the v value, and *H*(*S*) is the entropy of the set S with *n* classes, each with a *pi* appearance probability (4):

$$H(S)=\sum_{i=1}^{n}-p_i\log_2 p_i \quad (4)$$

Another important feature of the algorithm is the pruning of the decision tree, once the learning is done, meaning that the tests that are not really helpful for the decision problem are being eliminated. A later version of the C4.5 algorithms is the C5.0, used mostly in commercial systems.

Reinforcement Learning is a very interesting Machine Learning algorithm. The idea of Reinforcement Learning is very simple: an agent is exploring an environment and acting upon it, and in the end it receives a reward or a penalty. The agent will find out whether it acted correctly or not, without having the reasons explained.

In Reinforcement Learning, also called *learning with a critic* or *rewarded learning*, no hints are offered regarding the expectations; the only feedback is that the result will be categorized as correct or wrong. The situation is similar to the one of a critic that only claims that a certain thing is right or wrong, but doesn't explain it. Often the reward is being delayed.

The Reinforcement Learning algorithms are looking for a way of action in order to maximize the reward. The subject that has to learn isn't told the direction to action, as in most of Machine Learning techniques, instead it has to discover which action might bring the most efficient reward. In the most complex situations, the actions will not only affect the immediate rewards but also the future rewards.

The Reinforcement Learning technology is generally used for solving the so called Markov Decision Problems (MDP). The structure of a MDP consists in the following elements: the system's state, the actions, the transition probability, the transition rewards, a policy and a performance measurement mode.

The system's state is a parameter or a set of parameters that are supposed to describe the system. If a system's state is modifying in time, it is called to be a dynamic system. A dynamic system can be considered the queue formed at the cashier in a store. In this conditions, the x state, represented by the number of individuals that form the

queue, becomes x+1, if a new individual is joining the queue, and becomes x-1, when an individual has paid and leaves the queue.

Actions represent situations in which a system may or may not fulfill one of the options it has available. We consider an action *a* to be selected in the state *i*, and let *j* be the next state. The probability of transition is expressed by *p(i, a, j)* which depicts the probability of transitioning from the state *i* to state *j* through action *a*.

The system will usually be rewarded when a transition from one state to another is performed, reward described by *r(i, a, j)*.

The choice of the action in each state the system transitions through is established by a rule. In certain states no actions will be performed. The states in which decisions should be taken are naturally named *decision states*.

The rule for selecting an action must be designed as so it would be capable of selecting the optimal action, thus a means of measuring performance becomes available, means we shall define as the *medium reward for a rule*.

If we have a rule π then $\pi(i)$ will be the selected action by this rule for the state *i*. Let x_s identify system state before the *s*-th transition. The next formula (Gosavi, 2004) will describe the medium reward for the rule π starting with state *i*, considering $x_1 = i$. The medium reward, ρ_i, expresses the sum of all immediate rewards divided to the number of transitions (*k*), calculated on a longer period of time.

$$\rho_i = \lim_{k \to \infty} \frac{E\left[\sum_{s=1}^{k} r(x_s, \pi(x_s), x_{s+1}) \middle| x_1 = i\right]}{k} \qquad (5)$$

E – the medium value of the sum above

i – the initial system state

$\pi(x_s)$ – the action in state x_s

The limit in formula (5) is constant for any value of x_1 if the Markov problem satisfies certain conditions, thus we are going to have $\rho_i = \rho$ for any value of *i*. The goal of Markov's decision problem is finding a rule that will maximize the medium reward.

MDP can be solved through the dynamic programming method, which requires all the transition probabilities, *p(i, a, j)*, and the rewards *r(i, a, j)*.

The SMDP (Semi-Markov Decision Problem) variant requires an extra parameter that is the time span required for each transition. The duration of transition from state *i* to state *j* influenced by another state *x* will be expressed by *t(i, a, j)*. For SMDP the medium reward within the given conditions, using an initial state *i*, is defined through the next formula (6).

$$\rho_i = \lim_{k \to \infty} \frac{E\left[\sum_{s=1}^{k} r(x_s, \pi(x_s), x_{s+1}) \middle| x_1 = i\right]}{E\left[\sum_{s=1}^{k} t(x_s, \pi(x_s), x_{s+1}) \middle| x_1 = i\right]} \qquad (6)$$

An example of Reinforcement Learning is the so-called Q-learning algorithm, an extension of the traditional dynamic, which allows an agent to learn some rules from an arbitrary environment.

Q-learning eliminates the need of considering the maximum in a set of integrals, succeeding to map values (Q-values) from state/action pairs. Rather than associating the value of a function, Q-learning will use the so-called Q-functions. In every state there is a Q-value associated with each action. This Q-value is the sum of rewards achieved for performing the associated actions and following the given rules. In the Q-learning context, the value of a state is defined as the maximum of a Q-value in the given state.

Having the estimated utility Q-function which describes how useful an actions is, given a certain state. *Q(s, a)* is the immediate reward achieved for performing an action that leads to a maximum usability of the resulting state.

The formal definition of the Q-function is the following:

$$Q(s,a) = r(s,a) + \gamma \max_a (Q(s',a')) \qquad (7)$$

where:

r(s, a) is the immediate reward;

γ is the relative value of the delayed rewards versus the immediate rewards (*0* or *1*);

s´ is the new state following after action *a*;

a, a´ are the actions in states *s* and *s´*.

The selected actions are defined by the following function:

$$\pi(s) = \arg\max_a Q(s,a) \qquad (8)$$

Q-learning represents an algorithm without predefined learning rules. It can be demonstrated that for a sufficient amount of training under any rule ε-soft, the algorithm converges with a probability of 1 towards a small approximation of an action-value function for an random rule-target group. Q-learning teaches optimal rules even when the actions are selected from larger or even random rules.

In conclusion, from the point of view of storing data for using it in a decision support system, the XML format remains as elegant in structure as well as in utilization, but it is not recommended for huge amount of data; storage is possible in this case, but further processing of these data becomes way to difficult compared to the alternatives involving database systems. A problem in any given field of interest can be translated into a Markov decision process and solved using this technique. Reinforcement Learning is an extension of the classical dynamic programming and covers a set of problems it can solve. As opposed to the supervised learning, Reinforcement Learning does not require I/O data. It is foreseen that this kind of technology, combined with others, will be able to finally solve problems that could not be solved before.

REFERENCES

[1] Bishop, C., *Pattern Recognition and Machine Learning*, Springer Publishing, 2006

[2] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. Witten, "The WEKA data mining software: an update", SIGKDD Explorations, Volume 11, Issue 1, 2009

[3] A.R. Ganguly, A., Gupta , Data Mining Technologies and decision Support Systems for Business and Scientific Applications, Encyclopedia of Data Warehousing and Mining, Blackwell Publishing, 2005

[4] Hamilton, H., Gurak, E., Findlater, L., Olive, W., *Knowledge Discovery in Databases*, University of Regina, Canada, 2002, http://www2.cs.uregina.ca/~hamilton/courses/831/notes/ml/dtrees/c4.5/tutorial.html

[5] C. Mitra, G., *Models for decision making: an overview of problems, tools and major issues*, Mathematical Models for Decision Support, NATO ASI Series, Vol. 48, Springer Publshing, 1988

[6] D. Power, Categorizing Decision Support Systems: A Multidimensional Approach, in volume Decision Making Support Systems: Achievements, Trends and Challenges for New Decade, Idea Group Publishing, 2003

[7] C. O.Stanciu, supervisor: Professor Ioan Ştefan Nițchi Ph.D., "Contributions to using decision support systems based on machine learning technologies in business management, Ph.D. thesis, 2010

[8] C. O.Stanciu, "Solutions for the development of decision support systems", ANALE Seria Ştiinţe Economice (Universitatea „Tibiscus" din Timişoara), Vol. XV, ISSN. 1582 – 6333, 2009

[9] C. O.Stanciu, A. Cojocariu, "XML Technologies for Improving Data Management for Decision Algorithms", 20th DAAAM International Conference, Vienna, 2009

[10] I.H. Witten, E. Frank, "Data mining – practical machine learning tools and techniques", Second Edition, Norgan Kaufmann Publishing, 2005