Learning Patterns of Behavior by Observing System Events

Marlon Núñez

Departamento de Lenguajes y Ciencias de la Computación, Universidad de Málaga
Campus de Teatinos s/n, Málaga 29071, Spain
mnunez@lcc.uma.es

Abstract. The proposed algorithm (BPL) induces behavior patterns from events taking into account characteristics of observed systems and their environment. The main strategy of this method consists on building summaries of the behaviour of a system as events arrive, and take these summaries as training examples. BPL constructs summaries with new features from events, like duration of current event values, repetitions of an event in a period of time, amongst others. This algorithm has been tested in learning faulty behavior of networks with the purpose of continuously predicting alarms.

1 Introduction

The learning of behavior patterns is important for all intelligent entities and it is also useful for those researchers who want to know behavior patterns of a group of systems. This knowledge is useful in several fields. This paper is mainly focused in applying behavior knowledge for predicting events and explaining patterns. Applications can cover the control of systems, system imitation, customer behavior analysis, and alarm analysis, amongst other fields.

The techniques that could be considered nearer to this field are those related to the learning of patterns of sequences. Those technics are adapted to the analysis of unordered sets of examples. Basically, such data can be viewed as a sequence of events, where each event has an associated time of occurrence. When discovering episodes in a network alarm log, the aim is to find relationships between alarms. Such relationships can then be used in an on-line analysis of the incoming alarm stream, e.g., to better explain the problem that causes alarms, to suppress redundant alarms, and to predict severe faults.

Technical problems related to the recognition of episodes have been researched in several fields. A problem of discovering frequent episodes in a sequence of events was presented in [5]. Their patterns are arbitrary directed acyclic graphs, where each vertex corresponds to a single event (or item). An edge from event A to event B denotes that A occurred before B. They move a time window across the input sequence, and find all patterns that occur in some user-specified percentage of windows. An algorithm is designed for counting the number of occurrences of a pattern when moving a window across a single sequence. In a different approach [8] the problem is how to discover all sequential patterns with a user-specified minimum

R. López de Mántaras, E. Plaza (Eds.): ECML 2000, LNAI 1810, pp. 323-330, 2000.

support. Each sequence is a list of transactions ordered by transaction-time, and each transaction is a set of items. In Bioinformatics, the problem is to discover patterns common to a set of related protein or amino acid sequences [3].

2 BPL Algorithm

The general strategy of the BPL algorithm is to transform a problem that is seemingly not supervised, like a series of successive events in the time, into a supervised problem, where the training examples are mainly *behavior summaries*. Each summary will have several labels that are the temporal distances from the moment of the summary to the occurrence of each target event. Since these labels are numeric values, the Behavior Pattern Learner (BPL) algorithm generates regression trees, one for each target event that is wanted to be predicted.

The algorithm learns the behavior in terms of several groups of rules. Each group of rules is used for predicting a target event as well as explaining the prediction. The purpose of learning groups of rules is allowing the specialization of knowledge for each target event and improving the precision due to this specialization.

During the inference, several rules might fire simultaneously, predicting different target events, their expected intervals of time and their confidence. As the time gets closer to the expected occurrence of the target events, other rules might fire to predict the same target events for a shorter time interval; it is also possible that hypotheses change and previous hypothesis may be substituted by new hypothesis with different intervals of time, depending on the new events that have arrived.

Two important concepts used by BPL algorithm must be defined: events and BehaviorSummaries.

2.1 Events

An event indicates a change of value of an object attribute, as well as the time in which this change took place. An event is represented as *[Object. Attribute: Value, Time of occurrence]* where object is the identifier of an observed system. Attribute is the identifier of the variable that changes its value. Value is the new value of Attribute that changed at the specified time of occurrence. An examples of events is *[CAR122. Clutch: "pressed", 14:00]* which indicates that in the CAR122, the clutch was pressed at 14:00. Another example: *[PC1234. Alarm: CommunicationProblem, 12:00]* indicates that the PC1234 emitted the event *alarm = "Communication Problem"* at 12:00. If we are talking about an event in general without specifying the object, the notation [Clutch: "pressed"] is used.

2.2 BehaviorSummaries

The system learns from training examples called Behavior Summaries, made up of three kinds of preconditions and its consequences. The precondition types are Event Characteristics, SystemCharacteristics and Environment Characteristics. The possible

consequences of those characteristics are described in terms of the times of occurrence from that summary to target events.

2.2.1 EventCharacteristics

EventCharacteristics are new features calculated from events:

- Duration [*StillValidEvent*]: The duration of the validity of an event that occurred in the past but is still valid: Duration [*StillValidEvent*] = time[BehaviorSummary] - time[S*tillValidEvent*]
- Latency [*PastEventNoLongerValid*]: There is a probability > 0 that *PastEventNoLongerValid* continues having a *latent* effect during a certain time interval. The time of latency is the time past since *PastEventNoLongerValid* occurred. But there should be a limit. This limit is a time window called LatencyWindow [Event]. After this user-defined window time the event is supposed not to have any effect.
- Repetitions [*PastEvent*]: the number of repetitions of PastEvents in a period of time equal to the LatencyWindow[*PastEvent*].

These are the most useful new event characteristics. But some others can be used, like PastDuration [*PastEventNoLongerValid*] or LatencyOfRepetition [*PastEvenNo LongerValid*].

User has to provide his knowledge about the LatencyWindow of every event, namely, the maximum time of influence of every value of a variable. E.g. the event [car. clutch:"pressed"], has a LatencyWindow, that could be, say, 20 seconds, at most, which means that whatever happened to the car behavior will not be influenced by a past event [car. clutch: "pressed"] after 20 seconds. On the other hand, an event [car. brakeAlarm: " "low level"] will influence future events of the car for a running period of, say, one month [car. brakeAlarm: "low level"]. That is to say: 30x60x60x24x30 seconds.

2.2.2 SystemCharacteristics

Set of characteristics of the system that generated the events (The $SystemCharacteristics_i$ are all the values that describe the system that generated the event in time *time*[$BehaviorSummary_i$]). There are special events, called *state* events, which might update values of system attributes.

2.2.3 EnvironmentCharacteristics

Set of characteristics of the environment of the system (all the values that describe the environment in a time time[$BehaviorSummary_i$], that is to say, the instant of the $BehaviorSummary_i$).

2.2.4 Possible Consequences

The BehaviorSummary also registers the possible consequences of a situation. The Possible Consequences is a list of occurrence times from the BehaviorSummary to each target event: Then, $PossibleConsequence_i$ = (*time*[$targetEvent_1$], *time*[$targetEvent_2$],... *time*[Target $Event_k$]), where *time*[Target $Event_i$], is the time of the next occurrence of the $TargetEvent_i$.

2.3 Description of the BPL Algorithm

BPL algorithm has two phases: BehaviorSummary creation and learning phase. During the BehaviorSummary creation, BPL receives events. Each event generates several BehaviorSummaries to allow analysis of its consequences. During learning phase, BehaviorSummaries are used to grow regression trees, one for each target event. Every tree constructs a group of behavior rules. Table 1 illustrates BPL algorithm. Basic BPL algorithm, is independent of the regression tree method used.

2.3.1 Construction of BehaviorSummaries from Events

As it was mentioned previously the system learns from behavior summaries labeled with continuous classes. When an event arrives, all these variables (duration, repetition, and latency, among others) are updated in a table called Instant BehaviorSummary (IBS) table. There will be an IBS table for each observed system. When there is an order for constructing a behavior summary with an older time than the time of arrived event, a summary is constructed using values of attributes at the order time and calculated attributes like duration, repetitions and latency of events.

2.3.2 Scheduling Summary Orders

BehaviorSummaries are not generated periodically. Each event generates a schedule of summary orders that force the learning program to monitor its consequences in terms of occurrence of each Target Event.

An event at time *time*[event] generates several summary orders inserting them in chronological order in the SummaryOrdersList until a temporal position *time*[event] + *LatencyWindow*[event]. The Summary Orders are being scheduled ordered by time in the SummaryOrdersList. Once an event arrives, BPL algorithm analyses if constructing a Summary by consulting the top of the SummaryOrdersList. If the time of the incoming event is greater than that time of the summary order, then the summary is constructed. This way, BPL will not have to update IBS table continuously, but only when an event arrives.

2.3.3 Creation and Labeling of Behavior Summaries

When an event arrives, among other steps explained above, a SummaryOrder might be executed. Then BPL constructs a Behaviour summary, updating the IBS table calculating duration, repetition and latency according to the time of the BehaviorSummary. A BehaviorSummary will have as many numeric classes (also called labels) as target events are declared. Initially, BehaviorSummary labels are empty, pending on being filled in. To label a BehaviorSummary, BPL has to wait until a target event occurs. When one of the target events arrives it forces labeling the pending summaries. The label of each pending summary will be the time between that BehaviorSummary and the target event. If a target event did not arrive during a LatencyWindow (the largest one), the label is "it did not happen", namely, *d.n.h.* This value is set in the regression analysis as a specific negative value. In other words, there is a special label of BehaviorSummaries which means that a target event did not arrive or it did arrived at a time greater that the largest LatencyWindow of events. Attributes also have a special value: *n.a.* which stands for "not applicable". If the field

duration, for instance, has the value *n.a*, it means that the event occurred in the past and it is no longer valid, therefore it does not make sense to have any value. This value is also set as a specific negative value.

Table 1. BPL algorithm

Input:	Events, list of Target Events$_{1...n}$, LatencyWindows, N (Minimum new Behavior Summaries), and minimum support.
Output:	Behavior rules to predict/explain Target Events $_{1...n}$

Continually insert incoming events in the eventLog

// Behavior Summary creation Phase

While there exist not analyzed events in the eventLog,

Read the oldest not analyzed event. Consider it as *AnalizedEvent*

While time[AnalizedEvent]>time[top (BehaviorSummaryOrder List)]

Create a BehaviorSummary based on the InstantBehaviorSummary table. Insert it in BehaviorSummaryLog.

Extract the order at the Top of the BehaviorSummaryOrdersList

If AnalizedEvent is a target event

Label each BehaviorSummary$_i$ whose label is pending on update, with time[AnalizedEvent] - time[BS i]

Based on the AnalizedEvent, update their event characteristics in the InstantBehaviorSummary (IBS) table (number of repetitions, Duration/Latency, etc.)

Based on the AnalizedEvent, schedule a set of SummaryOrders inserting them into the SummaryOrdersList ordered by time until LatencyWindow $_{AnalizedEvent}$

// Learning phase

If there exist N new BehaviorSummaries with labels of a target event$_i$

Grow a Regression Tree$_i$ based on BehaviorSummaryLog

Construct Behavior Rules for target event$_i$ and replace previously learnt rules.

2.3.4 Learning Time Intervals

When there are N new BehaviorSummaries already labeled with a target event, BPL uses all summaries to generate knowledge to predict the target event. It learns as many regression trees as continuous classes, namely, target events. Although BPL can use any regression tree algorithm [1], [4], it has been tested with the EGR method [7].

EGR selects the best variable, analyzing the mixture of the data in every possible split, calculating the mean μ, standard deviation σ, and weight π of the components of the mixture, $[(\mu_1,\sigma_1,\pi_1), (\mu_2,\sigma_2,\pi_2)... (\mu_n,\sigma_n,\pi_n)]$. EGR learns regression trees using background knowledge (taxonomies and cost) associated to attributes, in a similar way that EG2 [6] for induction of decision trees.

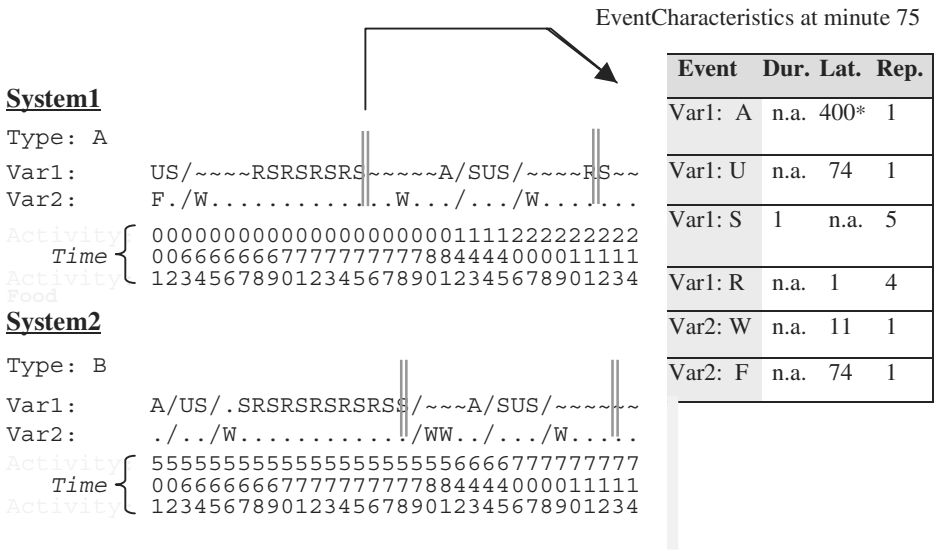
2.3.5 Construction of Behavior Rules

Regression trees do not show target events. They handle special negative numeric information, meaning *n.a.* and *d.n.h.,* this step pursues the following transformations:

- Every regression tree is transformed into a group of rules with a consequence in which the target event and the statistical support is expressed
- Negative numeric value in attributes, previously illustrated as *n.a.,* into symbolic values meaning "not applicable".
- Negative numeric value in classes, previously illustrated as *d.n.h,* into symbolic values meaning "it will not happen".

2.4 An Example

Let us suppose that we have systems. The first one is of Type A and the second one is of type B. Two variables will be observed: Var1 and Var2. The values of Var1 are A, R and S. The values of the attribute *Var2* are W and F. Figure 2 illustrates a time-line detailed by the symbols: (.), (~) and (/). The symbol (.) indicates a minute with no events; symbol (~) indicates a minute in which the last value of the attribute did not change; symbol (/) indicates an hour in which the last value of the attribute did not change. Vertical gray lines indicate BehaviorSummaries. Let's say that the target events are [Var1: A].



Event	Dur.	Lat.	Rep.
Var1: A	n.a.	400*	1
Var1: U	n.a.	74	1
Var1: S	1	n.a.	5
Var1: R	n.a.	1	4
Var2: W	n.a.	11	1
Var2: F	n.a.	74	1

Fig. 1. Evolution of attribute values and the BehaviorSummary at minute 75

Events from minute 1 to minute 75 update the ISB table. Figure1 also shows the calculated EventCharacteristics for System1 at minute 75. The asterisk "*" means that last time system1. Var1 changed its value to A was 400 minutes ago, that is to say, outside the timeline. Let's see what happen when event [Var2: W] arrives at time 78, and it is considered as the *analyzed event.* Then time[analyzed Event] >

top[BehaviorSummary Order] since minute 78 > minute 75. Therefore the **`While`** instruction applies. Consequently a BehaviorSummary is created from ISB table. This new BehaviorSummary is inserted in a log, without label yet, because its consequences are not known yet. Since the event [Var2: W] is a target event, it labels the BehaviorSummary with the distance time from the behavior summary (minute 75) to the time of the target Event (minute 78), that is to say, 3. The BehaviorSummary is a record of 20 fields: two with the current values of attributes (Var1 and Var2) and the other 18 fields regarding EventCharacteristics that summarized that situation. This training example has one numeric class. The process explained above goes on continuously. When there are N new behavior summaries, then they become training instance of a regression tree method. Figure 2 shows parts of the two regression trees generated, one for each target event.

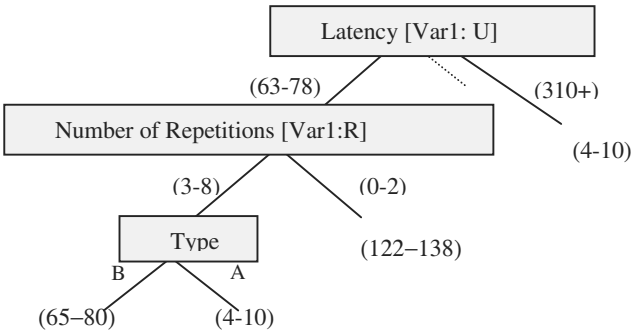


Fig. 2. *Re*gression tree for predicting [Var1: A]

3 Experimentation

Basic BPL has been evaluated trying to find faulty-behavior patterns. An analyzed domain was the faulty behavior of a PC Network in terms of its alarms. The PC network is made up of a Server, three Workstations and a LAN. The system was feed with six event logs of operating system (WindowsNT Server and Workstation).

We analyzed 450 errors and warnings messages, corresponding to 10 months of Windows NT administration. We selected 5 target events mostly related to network problems. We took the *oldest* 70% of consecutive events for training and the most recent 30% for testing.

BPL generated 52 behavior rules for predicting 5 target events. It was surprising that 48% of behavior rules have "will not happen" value as a consequent. These rules describe the preconditions for predicting that a target event will not occur in the *near future* (max. of LatencyWindows). The amount of rules of this kind is large; after analyzing these rules, we realized that there were very useful for being more specific in the prediction. For example, if we say that target event X will probably occur in [25-30] minutes and target event Y will not occur in [320+] minutes, we are giving more information about the network.

Latency Windows affects directly the rules learnt. If LatencyWindows are too small, BPL can not recognize true consequences of Alarm "Netlogon" because this event does not generate SummaryOrders to a future time when these consequences really occur. On the other hand, too big LatencyWindows adds many patterns not related to the problem, which does not help to find true patterns.

4 Conclusions

An important characteristic of the BPL algorithm is that periodic analysis does not exist. Generation of regression trees is carried out every time that N new BehaviorSummary arrives, which allows not to be necessary to establish a certain learning rhythm. If there are target events with short effect and, simultaneously, other target events with long effect, BPL will generate each tree with different rhythm and different time intervals. On the other hand, an important finding, ignored by sequence pattern learning approaches, is the importance of the characteristics of systems and environment how these factors affect systems behavior.

Acknowledgments

Special thanks to Rafael Morales for giving comments and support to this research work. I would also thank Nohelia Barreiro, Raúl Fidalgo and Ignacio Ballesteros for developing, and testing some modules. I would also thank José L. Pérez who helped with some logistical support. This work has been partially supported by project FACA PB98-0937-C04-01 of the CICYT, Spain. FACA is a part of the FRESCO project.

References

1. Breiman, L., Friedman, J.H., Olsen, R.A., & Stone, C.J., (1984). *Classification and Regression Trees*, Wadsworth Int. Group, Belmont, California.
2. Grossi, R and Luccio, F. (1989). Simple and efficient string matching with k mismatches. *Information Processing Letters*, 33:113 – 120, 1989.
3. Jonassen I., Collins J.F. and Higins D. Finding flexible patterns in unaligned protein sequences. Protein Science, 4 (8): 1587 - 1595, 1995.
4. Karalic A. (1992) Employing Linear regression in regression trees leaves. In Technical Report IJS DP-6450, Jozef Stefan Institute, Ljubljana, Slovenia.
5. Manila, H., Toivonen H., and Verkamo Inkeri (1997) Technical Report C-1997-15, Departament of Computer Science, University of Helsinki.
6. Núñez, M. (1991) The Use of Background Knowledge in Decision Tree Induction. *Machine Learning (Vol. 6, Nº3)*. Los Altos, CA. Morgan Kaufman.
7. Núñez, M. (2000). *Generalised Regression Trees,* accepted paper for the *14th Intl. Conference of Statistical Computing (Compstat),* The Netherlands.
8. Srikant, R. and Agrawal R (1996). Mining Sequential Patterns: Generalization and Performance Improvements. In *Proceedings of the 5th International Conference EDBT-96*, Avignon, France.