Paper accepted for presentation and publication in the proceedings of the ICML98/AAAI98 Workshop on ''Predicting the future: AI Approaches to time series analysis'' Madison, July 24-26th, 1998

Early prediction of electric power system blackouts by temporal machine learning

P. Geurts and L. Wehenkel
Department of Electrical Engineering - University of Liège
Institut Montefiore - Sart-Tilman B28, B-4000 Liège, Belgium
Email : lwh@montefiore.ulg.ac.be - Fax : Int. +32 4 366 2984

Abstract

This paper discusses the application of machine learning to the design of power system blackout prediction criteria, using a large data base of random power system scenarios generated by Monte-Carlo simulation. Each scenario is described by temporal variables and sequences of events describing the dynamics of the system as it might be observed from real-time measurements. The aim is to exploit the data base in order to derive as simple as possible rules which would allow to detect an incipient blackout early enough to prevent or mitigate it. We propose a novel "temporal tree induction" algorithm in order to exploit temporal attributes and reach a compromise between degree of anticipation and selectivity of detection rules. Tests are carried out on a data base related to voltage collapse of an existing large scale power system.

Introduction

This paper presents a new method for the induction of temporal detection rules from large amounts of mixed symbolic and numerical data. The data base is composed of a large number of dynamic scenarios of a system characterized by two types of attributes : variable step-size numeric time-series and time-tagged sequences of events. Each scenario is classified with respect to a target symbolic class, and the aim of machine learning is to build a detection rule which would be able to detect as early as possible the scenarios of the target class. The devised method builds up detection rules in the form of temporal trees, using a two-stage greedy approach : first a large tree is grown, then this tree is pruned in order to avoid overfitting. The method is tested on a fairly large data base corresponding to the detection of power system voltage collapses from real-time information.

The paper is organized as follows. Next section describes the physical problem and discusses its peculiarities from the machine learning point of view. The following two sections describe in detail the devised algorithms for temporal tree induction and the results obtained in our empirical case study. The last sections of the paper discuss further research and provide a preliminary comparison with related work.

Problem statement

We start by describing the physical problem considered and then discuss its peculiarities from the machine learning point of view. We end by stating the objective of the temporal tree induction method proposed.

The physical problem

Electric power systems are among the most complex man made systems on the world. One of the main problems in power systems is to arbitrate between economy and security. Simply stated, the security of a power system denotes its capability to provide continuous operation in spite of the large diversity of disturbances (variations in consumer demand, internal failures, external perturbations like lightning strikes, storms. . .). Security is handled in practice through two complementary strategies : preventive control, which is carried out by human operators in order to maintain the system in a state where it can withstand disturbances; emergency control, which acts automatically after a disturbance has occurred in order to minimize its consequences. Since disturbances are intrinsically random, preventive control will essentially aim at balancing the economic cost of normal operation with the risk (expected severity) of instability/insecurity. On the other hand, emergency control essentially aims at reducing the severity of instabilities. It is worth noticing that due to increasing competitive pressure in the electric industry, and thanks to the possibilities offered by modern communication and computing technologies, the trend in power systems is to rely more strongly on emergency control.

One of the main problems in the design of emergency controls is to define appropriate criteria which are able to predict in real-time whether the system is in the process of loosing stability or not. This implies the selection of appropriate real-time measurements (among a multitude of possible ones), filtering out the useful information contained in them (e.g. separating short term transients from the relevant trends), and combining these in order to formulate detection rules. Notice that since emergency controls are supposed to operate under extremely stringent but very seldomly observed conditions, their design process essentially relies on numerical simula-

tion of the power system behavior under various conditions likely to drive it towards an instability. Thanks to the quickly growing amount of available computing power it is now possible to use Monte-Carlo sampling techniques in order to screen very large samples (several thousand) of large scale simulations, yielding large data bases of simulation results. These data can then be exploited using automatic learning (machine learning, statistical techniques, neural networks) and other data mining tools (visualization, sorting, subset selection...) in order to extract synthetic information.

The framework combining automatic learning with Monte-Carlo simulations in power system planning and operation is very general, and has been applied to various design, prediction and monitoring problems. We refer the interested reader to (Wehenkel 1997) for an overview and to (Wehenkel 1998) for a deeper discussion of this topic. In this paper we focus on emergency control and more specifically on the way to exploit temporal attributes by machine learning, in order to automatically build prediction rules which are at the same time selective and anticipative enough.

While we use the above power system emergency state detection problem in our experiments, we notice that there are many other practical applications which present similar characteristics, such as, for example, monitoring of other kinds of large scale system (telecommunication networks, industrial process control...) or monitoring patients in various medical applications.

The early detection problem

From the machine learning point of view, our problem is intuitively formulated as follows (to save space, we will not discuss this from the formal point of view) :

Universe of scenarios. We are given a universe U of objects (denoted by o) representing dynamic system trajectories (scenarios), which are on the one hand described by a certain number of *temporal candidate attributes*, on the other hand classified into one of two possible classes $c(o) \in \{+, -\}$.

We denote by $a^i(\cdot,\cdot)(i = 1 \dots m)$ a candidate attribute (a function defined on $U \times [0 \dots + \infty[$), and by $a^i(o,t)$ its value. We consider the two following types of temporal attributes :

Numerical : $a^i(o,t) \in I\!R$;

Event subsets : $a^i(o,t) \subseteq E_{a^i}$, where E_{a^i} is a finite set of possible events.

Detection rules. Denoting by $\boldsymbol{a}(o,\cdot)$ the attribute vector $(a^1(o,\cdot), \dots a^m(o,\cdot))$, and by $\boldsymbol{a}_{[0\dots t]}(o,\cdot)$ its restriction to the interval $[0 \dots t]$, we consider a detection rule for class $+$ as a function

$$d(\cdot,\cdot) : U \times [0 \dots + \infty[\;\rightarrow\; \{+,-\} \qquad (1)$$

which has the following property

$$\begin{array}{c} \forall o, o' \in U, \forall t \in [0 \dots + \infty[: \\ [\boldsymbol{a}_{[0\dots t]}(o,\cdot) = \boldsymbol{a}_{[0\dots t]}(o',\cdot)] \Rightarrow [d(o,t) = d(o',t)]. \end{array} \qquad (2)$$

Thus, a detection rule classifies an object at time t on the basis of $\boldsymbol{a}_{[0\dots t]}(o,\cdot)$, or, in other words, it depends only on present and past attributes values.

Monotonicity. We also assume that if an object is classified into class $+$ at some time, it will remain so for all later times (monotonicity). In other words,

$$d(o,t) = + \;\; \Leftrightarrow \;\; \forall t' \geq t \; : d(o,t') = +. \qquad (3)$$

Thus, for an object o, we will denote by $t_d(o)$ the first time it is detected

$$d(o, t_d(o)) = + \text{ and } \forall t' < t_d(o) \; : d(o,t') = -. \qquad (4)$$

Machine learning problem. Given a learning set $LS = \{o_1 \dots o_N\} \subseteq U$ (a sample of N objects) of known class, and whose attributes values are observed for some finite period of time $[0 \dots t_f(o)]$, the objective is to automatically derive a detection rule which would perform as well as possible in detecting objects from U. Clearly, a good detection rule is a rule which will detect only those objects which actually belong to class $+$, and among good rules, the better ones are those having the smallest detection times.

Comments

Note that in our formulation time is continuous. In the practical application discussed in this paper, numerical attributes are represented as piecewise linear functions of time, with step sizes varying from one object to another and possibly from one attribute to another. On the other hand, event subset attributes are explicitly represented as lists of pairs

$$((E_1,t_1) \dots (E_l,t_l)), \qquad (5)$$

where $t_i < t_{i+1}$ and $E_i \subseteq E_{a^i}$ denotes the subset of events which happen at time t_i.

Note also that in our application it is preferable to minimize the number of different attributes used in a detection rule, in order to improve comprehensibility so as to facilitate validation of results by human experts, and also to reduce the cost of actual implementation. Given the large scale of power systems, the number of potential candidate attributes is however very large. Thus, the machine learning method should be able to select among the proposed candidate attributes a small number of relevant ones, if these exist, or indicate that there are no such "ideal" measurements among those used to represent the scenarios.

The monotonicity property that we impose on detection rules may appear as quite restrictive. However, we think that in the early detection problem this assumption is rather natural, in the sense that what matters is whether and how early detection happens. As we will see in the next section, this assumption makes it possible to set up a rather simple machine learning algorithm.

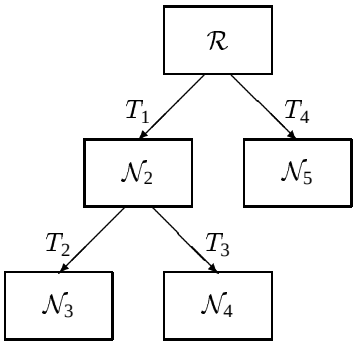

Figure 1: Example temporal tree

Temporal tree induction algorithm

The proposed algorithm is a modification of the well-known TDIDT (top down induction of decision trees) method (Breiman *et al.* 1984; Quinlan 1986). As in TDIDT, temporal tree induction is composed of two search stages : (i) tree growing, which aims at selecting the appropriate attributes and defining the tree structure as a collection of tests; (ii) tree pruning, which aims at determining the appropriate tree complexity in order to avoid overfitting the data contained in the learning set. Prior to tree induction, the overall learning set LS is first decomposed in two disjoint subsets : the growing sample (GS) used to evaluate the quality of candidate trees during growing, and the pruning sample (PS) used to evaluate trees during pruning.

We first describe the proposed semantics of temporal trees and introduce the quality measures used to evaluate them, then we discuss the proposed growing and pruning methods.

Proposed semantics of temporal trees

Figure 1 shows a simple temporal tree. The elementary bricks of this device are the tests T_i corresponding to its arcs. Each such test is a logical functional of the attributes

$$T(\cdot,\cdot) : U \times [0 \ldots +\infty[\;\rightarrow\; \{T, F\}, \qquad (6)$$

which satisfies

$$\forall o, o' \in U, \forall t \in [0 \ldots +\infty[: \\ [\boldsymbol{a}_{[0\ldots t]}(o,\cdot) = \boldsymbol{a}_{[0\ldots t]}(o',\cdot)] \Rightarrow [T(o,t) = T(o',t)], \qquad (7)$$

and which has a monotonicity property similar to the detection rules

$$T(o,t) = T \;\Leftrightarrow\; \forall t' \geq t \; : T(o,t') = T. \qquad (8)$$

Notice that in practice such tests depend only on a single attribute.

Using such a temporal tree, and given a value of t, a scenario is propagated through the tree along all paths starting at the root $\mathcal{R}$ (i.e. the top-node), until a condition T_i along the path is false or a terminal node is reached. If the scenario reaches at least one terminal node, it is classified into class $+$, otherwise it is classified into class $-$ at time t.

Notice that the monotonicity property of tests implies the monotonicity of the tree detection. It implies also that scenarios can only move downwards the tree as time advances. Indeed, as time advances, conditions which are true remain so and other conditions may become true allowing a scenario to move down the tree. For a given tree $\mathcal{T}$ and a scenario o, we denote by $t_{\mathcal{T}}(o)$ the time at which o first reaches a terminal node. Of course, some of the scenarios may never reach a terminal node and will thus remain classified permanently in class $-$.

Notice also that for the sake of consistency we impose that a trivial tree (composed of only its root) classifies all objects in class $+$ at any time t.

Given a sample S of objects, each node of such a tree corresponds to a subset of S, namely those states which eventually satisfy (for some finite time t) all the tests along the path from the root $\mathcal{R}$ to the node. In particular, the root $\mathcal{R}$ corresponds to S, and the subsets along a branch are nested. However, the subsets corresponding to different branches may overlap.

Evaluation function

In order to assess the quality of a temporal tree from a sample of objects S, we propose the following evaluation function, composed of two terms :

$$Q(\mathcal{T}, S) = \beta Q_s(\mathcal{T}, S) + (1-\beta) Q_t(\mathcal{T}, S), \qquad (9)$$

where Q_s evaluates the selectivity of discrimination between sample objects of different classes, Q_t the degree of anticipation of detection, and $\beta \in [0 \ldots 1]$ is a user defined parameter to tradeoff anticipation vs selectivity.

Q_s is defined by (e.g. see (Paliouras 1997))

$$Q_s(\mathcal{T}, S) = \frac{(1-\alpha) N_{\mathcal{T},+} + \alpha N_{\mathcal{T},-}}{(1-\alpha) N_+ + \alpha N_-}, \alpha \in [0,1], \qquad (10)$$

where N_+ (resp. N_-) denotes the number of samples of class $+$ (resp. $-$), $N_{\mathcal{T},+}$ (resp. $N_{\mathcal{T},-}$) denotes the number of them classified $+$ (resp. $-$) by the tree $\mathcal{T}$, and α is a user defined parameter to tradeoff non-detections vs false-alarms.

For Q_t we propose the following definition

$$Q_t(\mathcal{T}, S) = \frac{1}{N_{\mathcal{T},+}} \sum_{o \in \mathcal{T}^+}^{N} f(\frac{t_{\mathcal{T}}(o)}{t_f(o)}), \qquad (11)$$

where $\mathcal{T}^+$ denotes the subset of S of objects of class $+$ correctly detected by the tree, $t_{\mathcal{T}}(o)$ the time to detect object o, and $f(\cdot)$ is a monotonically decreasing function defined on $[0 \ldots 1]$ such that $f(0) = 1$ and $f(1) = 0$ (in our simulations $f(x) = 1 - x$). In eqn. (11) $t_f(o)$ denotes the maximal time after which observation of o stops. This time is fixed a priori for each object, and allows one to specify a time after which detection is not considered anymore useful.

Table 1: Temporal tree growing algorithm

INITIALIZE
I1. Denote by GS the sample (part of the learning set) used for tree growing;
I2. Denote by $\mathcal{T}$ the current tree : set $\mathcal{T}$ to the trivial tree containing only the root node $\mathcal{R}$;
I3. Push $\mathcal{R}$ on the intitially empty stack (at any time the stack contains a list of nodes candidate for expansion);
ENDINITIALIZE
BEGINLOOP
L1. If the stack is empty **Return** $\mathcal{T}$ (tree building is finished);
L2. Let $\mathcal{N}$ be the node at the top of the stack;
IF the subset of GS corresponding to $\mathcal{N}$ contains only objects of class $+$
THEN the node becomes a terminal node of the tree and is permanently removed from the stack;
OTHERWISE determine among candidate tests an optimal one T_* such that $Q(\mathcal{T} \overset{\mathcal{N}}{\oplus} T_*, GS)$ is maximal;
IF $Q(\mathcal{T} \overset{\mathcal{N}}{\oplus} T_*, GS) > Q(\mathcal{T}, GS)$
THEN set $\mathcal{T}$ to $\mathcal{T} \overset{\mathcal{N}}{\oplus} T_*$ (add arc and successor node), push the new successor on the stack and determine its subset of GS;
OTHERWISE do not modify $\mathcal{T}$ and remove the node $\mathcal{N}$ from the stack;
ENDIF
ENDIF
ENDLOOP

Temporal tree growing

The tree growing method is based on the repetitive application of a single operator which consists of expanding the tree by adding a test to one of its nodes. It uses the growing sample GS (part of the overall learning sample) in order to guide the search. Table 1 describes the overall algorithm, which proceeds until no further expansion is required. In Table 1, we denote by $\mathcal{T}$ the current tree and by

$$\mathcal{T} \overset{\mathcal{N}}{\oplus} T \tag{12}$$

the result of expanding it by adding a test T to its node $\mathcal{N}$.

Starting with the trivial tree which classifies all objects in class $+$ at time $t = 0$, the purpose of the algorithm is to add tests which should operate as filters preventing objects of class $-$ to reach terminal nodes, and, at the same time, should let objects of class $+$ reach a terminal node as quickly as possible.

The algorithm uses a stack of "open" nodes (initialized to the root node corresponding to the complete growing sample), then proceeds by considering the node at the top of the stack for expansion. At each expansion step, it pushes the created successor on the top of the stack. It thus operates in a depth first fashion : it starts by determining a test to attach at the root node, then proceeds along that branch by considering the last created successor node. While developing such a first branch, tests are added in order to prevent false alarms; development of the branch stops as soon as quality can not be increased anymore, or if all the states belonging to the tip of the branch are in class $+$. Once a branch has been fully developed, the next stage of the algorithm is to add tests in parallel to those already installed, so as to increase the number of states of class $+$ detected and/or to reduce their detection times by providing alternative detection logics.

Anticipating on the next section, let us mention that each individual test will use information provided by a single attribute. In order to determine an optimal test at a given step, the method scans a certain number of candidate tests for each candidate attribute (see below) and computes the corresponding variation of the tree quality with respect to the tree obtained at the previous step. Among all candidate tests, it will use the one yielding a maximum increase in quality.

Hypothesis space of candidate tests

Numerical temporal attributes. The algorithm considers tests in the form

$$\begin{array}{c} T(o,t) = T \\ \Leftrightarrow \\ \exists t' \leq t \ \mid \ \forall \tau \in [t' - \Delta t \ldots t'] \ : a^i(o,\tau) < v_{th}, \end{array} \tag{13}$$

and

$$\begin{array}{c} T'(o,t) = T \\ \Leftrightarrow \\ \exists t' \leq t \ \mid \ \forall \tau \in [t' - \Delta t \ldots t'] \ : a^i(o,\tau) > v_{th}. \end{array} \tag{14}$$

It thus determines i (attribute selection), Δt (filtering) and v_{th} (discretization) to maximize quality. The user defines for each candidate attribute a limited number of candidate Δt values. Then, the method uses a brute force search screening, for each candidate attribute and Δt value, all locally relevant candidate thresholds, computing their incremental quality.

Event subsets. *Event subsets* are attributes which record a time-tagged sequence of events; they are the temporal version of qualitative attributes. We considered the two following types of tests (E_{a^i} denotes the finite set of possible values of an attribute $a_i(\cdot,\cdot)$, i.e. $a^i(o,t) \subseteq E_{a^i}$)

$$\begin{array}{c} T(o,t) = T \\ \Leftrightarrow \\ \exists\ t' \leq t\ \mid\ \bigcup_{\tau \leq t'} a^i(o,\tau) \supseteq V \end{array} \tag{15}$$

and

$$\begin{array}{c} T(o,t) = T \\ \Leftrightarrow \\ \exists\ t' \leq t\ \mid\ a^i(o,t') \cap V \neq \emptyset \end{array} \tag{16}$$

where V is selected among a set of candidate subsets of $E^+_{a^i}$, which is the set of all events which appear at least in one GS object of class $+$.

Notice that in our practical example, the number k of different events in a set $E^+_{a^i}$ may be of the order of 50 to 100. Thus, it was intractable to screen all possible subsets V (their number is $2^k - 1$).

Thus, for the conjunctive tests (eqn. 15) we limited the search to subsets V containing only two events.

On the other hand, the disjunctive tests (eqn. 16) were determined using a greedy search, as follows. First, all singleton subsets of $E^+_{a^i}$ are considered and sorted by decreasing order of their incremental quality. Then the method evaluates the incremental quality of the following sequence of event subsets

$$V^i = \bigcup_{j=1,\ldots i} V^1_j, i = 1,\ldots k, \tag{17}$$

where the $V^1_1, \ldots V^1_k$ denote the singleton event subsets sorted by decreasing incremental quality. Among the candidate V^i subsets, the one yielding a maximal incremental quality is retained. Notice that this family of candidate tests contains the best singleton also.

Discussion. The above types of tests were designed in order to handle the practical problem of power system voltage collapse detection.

For example, we defined the hypothesis space of candidate tests on numerical attributes based on the fact that voltage collapse may be detected by observing monotonically decreasing voltage magnitudes and/or excitation currents reaching their upper limits. However, due to the superposition of short-term transients it is necessary to use temporizations, which should be long enough to filter out irrelevant oscillatory transients and short enough to avoid postponing the detection too much. On the other hand, due to the non-linear behavior of power systems it is necessary to select the appropriate locations in the system where voltages or excitation currents should be measured and thresholds need to be adapted to system specifics.

However, the method could be easily tailored to other types of problems by extending or modifying its hypothesis space of candidate tests. For example, in order to detect oscillatory instabilities it would be interesting to consider candidate tests which detect consecutive oscillations of a certain magnitude; they would also be characterized by two parameters : oscillation magnitude and number of consecutive oscillations.

Pruning

The temporal tree growing method described above tends to produce overly complex trees, which need to be pruned to avoid overfitting the information contained in the learning set.

The proposed pruning method is similar in principle to the standard tree pruning methods used in TDIDT. It consists in generating a nested sequence of shrinking trees $\mathcal{T}_0, \ldots \mathcal{T}_K$ starting with the full tree and ending with the trivial one, composed of a single node. At each step, the method computes the incremental quality which would be obtained by pruning any one of the terminal nodes of the current tree, and removes the one (together with its test) which maximizes this quantity. It uses an independent set of objects (i.e. different from the learning set) in order to provide honest estimates of the tree qualities, and hence to detect the irrelevant parts of the initial tree. The tree of maximal quality is selected.

Notice that, while temporal tree growing is quite heavy from the computational point of view, the overhead of pruning is negligible. As we will see below, pruning reduces significantly the tree complexity (often by about 50%) and improves also quality.

Empirical study

Brief description of the practical study

Our tests were carried out on a large scale data base built by numerical simulation for the study of blackouts of the French power system (Wehenkel *et al.* 1997). The scenarios correspond to random combinations of large external disturbances and faulting protection devices (line overload protections, generator protections, substation protections...). The analytical model of this system uses about 11,000 states variables, and the simulations were carried out in parallel using a cluster of workstations (each simulation took about 11hours CPU time). Each scenario is characterized by about 800 numerical variables, observed during about 40 minutes of real-time. In the data base, these are represented using piecewise linear functions, with variable step-size (the number of steps varies from one scenario to another, and is typically of about 50 for stable scenarios and larger than 500 for highly unstable ones). There are about 200 million values stored in the data base.

In the above data base there are various possible modes of unstable behaviour (e.g. fast electromechanical transients, mid-term voltage collapses, cascades of overloaded line tripping...) which may act in different regions of the system. In our investigations we have considered only the

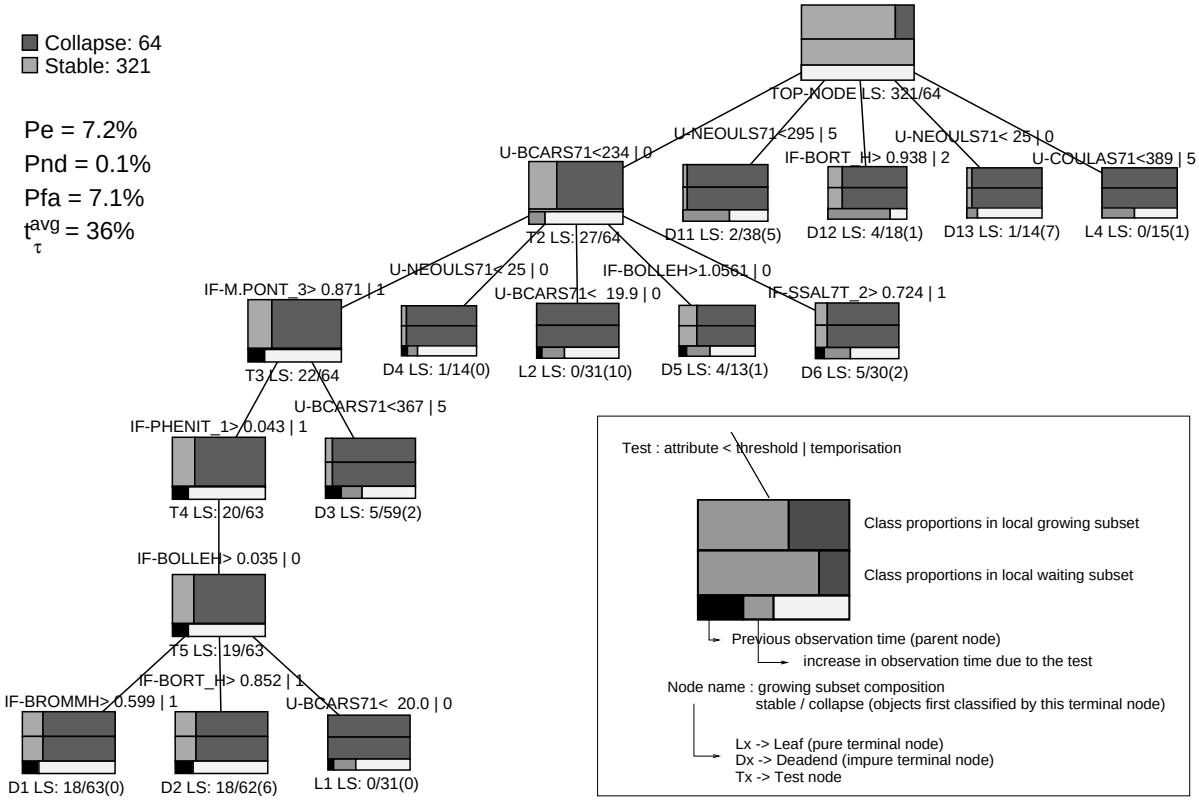

Figure 2: Voltage collapse detection tree (partial view)

voltage collapse problems, related to a particular peripheral area of the system. The data base contained 1100 scenarios, and we used a subset of about 100 candidate attributes (voltage magnitudes at different buses, generator excitation currents, various lists of events). Among the 1100 scenarios, there are about 250 unstable ones.

Obtained results

Figure 2 shows a partial view of a temporal tree built by the above method on the voltage collapse problem. The growing sample used to build the tree is composed of 385 random scenarios (64 correspond to voltage collapse). Tests are shown beneath the corresponding arcs of the tree, and each node is graphically represented by a box divided vertically into three parts : (i) the upper most part represents the proportion of object classes in the local growing subset, which are the objects which eventually reach the node (at the root this corresponds to the complete growing sample); (ii) in the middle, a similar representation shows the objects of the growing subset which never move downwards the tree (at the root this subset corresponds to the objects which are not detected by the tree, at the terminal nodes it is identical to the local growing subset); (iii) the lower part provides information about the average time needed to reach the node and its decomposition into the part required by all preceding tests (in black) and (in grey) the additional time required by the last test (notice that the time required to reach the root node is equal to zero for all states). The numbers in parentheses below a terminal node indicate the number of objects which are really detected by this node, i.e. which reach this node before any other terminal node. For example, the right most terminal node (named L4) corresponds to a growing subset composed of 15 collapse states, among which only one is detected by this node (all others are detected earlier by another terminal node of the tree).

In order to build the tree we proposed 29 numerical candidate attributes, together with three candidate values for Δt (0, 1, and 2 seconds for excitation currents, and 0, 5, and 10 seconds for voltages), and used $\alpha = 0.4$ and $\beta = 0.8$ in order to balance early detection with selectivity.

The tree building algorithm has selected 17 relevant attributes, to grow a tree comprizing all in all 26 tests, 22 terminal nodes and 5 internal nodes. This tree was then tested on the remaining (736) scenarios of the data base, yielding an overall error rate of $P_e = 7.2\%$ (1 non-detection among 167 collapse test scenarios; 52 false alarms among 569 stable test scenarios). The earliness of detection is measured by the mean detection time of unstable scenarios $t_{\mathcal{T}}^{\text{avg}} = 36\%$ (the mean value of the ratio $\frac{t_{\mathcal{T}}(o)}{t_f(o)}$). Note that the branches which were developed first are in left-most part of the tree.

Table 2: Tree characteristics under various conditions

Conditions				Fully grown tree						Pruned tree					
# LS	CA	β	α	SA	C	P_e(%)	P_{fa}(%)	P_{nd}(%)	$t_{\mathcal{I}}^{\text{avg}}$(%)	SA	C	P_e(%)	P_{fa}(%)	P_{nd}(%)	$t_{\mathcal{I}}^{\text{avg}}$(%)
200	29	1.0	0.5	2	4	4.4	3.6	0.8	57.6	2	3	4.9	4.2	0.7	56.3
200	29	0.8	0.5	15	22	10.2	9.9	0.3	47.4	5	6	6.7	5.7	1	50.2
200	29	0.8	0.3	13	16	10.2	9.9	0.3	47.2	5	6	7.1	6.5	0.6	51
200	29	0.7	0.4	19	30	11.3	10.1	1.2	45.8	6	7	9.2	6.3	2.9	45.3
385	29	0.8	0.4	17	27	7.2	7.1	0.1	36	11	17	5.5	5.3	0.2	40.3
385	13	0.8	0.4	6	22	14.8	8.4	6.4	40.4	4	10	13.3	6.9	6.3	38
600	13	0.8	0.4	6	19	8.7	7.7	1	34.1	6	15	8	7	1	34.2

CA: number of candidate attributes SA: number of selected attributes C: total number of nodes (complexity)

The tree was then pruned: this lead to a reduction in complexity (to a total of 17 nodes) and of error rate (5.5% : 5.3% false alarms and 0.2% of non-detections), and to an increase in the average detection time (to 40.3%).

Sensitivity analysis

In order to provide further insight into the behavior of the method, let us briefly discuss the various trends which appear from Table 2, which collects results obtained under various conditions (learning set size, candidate attributes, values of α and β).

Effect of pruning. A quick comparison of the pruned and unpruned trees' characteristics confirms the very strong effect of pruning : it reduces complexity by more than 50% in the mean, improves errors rates by about 10%, but may either increase or decrease degree of anticipation (which is inversely proportional to $t_{\mathcal{I}}^{\text{avg}}$).

Influence of α. Looking at the relative numbers of false alarms P_{fa} and non-detections P_{nd}, it appears that the method is not very sensitive to α.

Influence of β. Comparing the values of $t_{\mathcal{I}}^{\text{avg}}$ for the first four trees, we observe that they are indeed quite correlated to the corresponding value of β (negatively, as expected).

Increased number of learning states (# LS). The quality of pruned and unpruned trees and the complexity of the pruned ones increase significantly, as expected. Notice that quality assessment of the three last pruned trees might be slightly optimistic, since we used the same set of objects than those used for pruning.

Effect of candidate attributes. Comparing the last two trees (built using only event subsets types of attributes) with those in the first part of the table (built using only numerical attributes), we notice that the type of errors (false alarms vs non-detections) is much more sensitive to the types of candidate attributes than to α.

Computational performances

The algorithm was implemented in GNU CommonLisp (GCL). Indicative CPU times on a 200 MHz Sun UltraSparc workstation are of 15 hours for a tree built with 400 learning objects, 30 attributes, 3 candidate values of Δt. As for classical TDIDT, the computational complexity is slightly superlinear in the number of learning states, and linear in the number of candidate attributes and temporizations.

Discussion

We believe that the first results obtained with the proposed temporal tree growing method are quite encouraging, specially since they were obtained on a large scale data base from a real problem. We also believe that the method is flexible enough to allow its application to other temporal machine learning problems. However, it needs clearly further validation, and there are various problems which have not yet been addressed. We mention some of them in place of a conclusion.

Temporal constraint along a path. In our application, a path in a temporal tree is supposed to correspond physically to a particular instability mode, represented by a conjunction of logical tests. Thus, these conditions should become true altogether within a certain limited time interval. Hence, we propose to add to each path in a tree a *life-time parameter*, to constrain the time span within which its tests must be satisfied. This parameter can be derived either from the validation set used for pruning or be fixed a priori by the user. Notice that this would actually consist in relaxing a posteriori the monotonicity assumption.

Computational tradeoffs. The present (brute force) search algorithm is quite slow for large problems. In order to improve its efficiency it might be possible to add some heuristics so as to reduce the number of candidate tests evaluated at each step (e.g. reduce the number of candidate thresholds to allow for searching more systematically for op-

timal values of Δt). On the other hand, there is ample room for code improvement and parallization, if required.

Extensions to other problems, types of tests for different temporal features. In our work we concentrated on the automatic identification of thresholds with delays. The method could, however, be easily extended to identify other types of candidate tests, for example temporal patterns, such local maxima, change in pseudo-frequency, change in concavity. . . , which would be relevant in various practical applications.

Multiple classes to detect. It should be possible to extend the method so as to be able to detect several classes rather than a single one. We believe also that the approach could be adapted in order to handle numerical outputs, as would be required for example in time series forecasting applications.

Related work

In the machine learning literature the consideration of temporal problems is rather recent, and there is not yet a clear body of publications to which we could compare our work. Nevertheless, in the recent literature on the subject one may identify several trends.

A first trend consists of separating learning from temporal data into two successive steps (see e.g. (Torgo 1995; Manganaris 1997)) : (i) representation, where scalar attributes are defined in order to extract temporal features from time series; (ii) model selection, where the scalar attributes are used by an existing machine learning method in order to extract rules. With respect to this approach, we believe that the method presented in this paper may take better advantage of available data, by treating these two problems in an integrated fashion and taking into account explicitly the role of time in the evaluation of detection rules.

Another approach (e.g. (Paliouras 1997)) consists in using an a priori defined model structure (possibly more powerful than our temporal trees) and then using learning set information in order to tune the parameters of this model. This approach may be well suited in situations where the model structure is indeed known a priori, but not appropriate in applications like those considered in this paper.

Another research theme concerns unsupervised learning, such as identifying similar time series or frequent patterns of events (Mannila, Toivonen, & Verkamo 1997; Das, Gunopulos, & Mannila 1997). The latter type of methods are essentially complementary to our work.

Finally, yet another proposal which is also related to our method is the one given in (Jordan 1994; Jordan, Ghahramani, & Saul 1996). In these publications a hidden Markov decision tree model is proposed, where decisions at a node and at some time depend on the decisions taken at that node at the previous time step. One important difference with temporal trees is that the resulting model, while very powerful from the statistical point of view, is rather difficult to interprete. The other important difference is that the hidden Markov trees do not model explicitly the earliness of detection, since their aim is to predict the output variable at the current time, given past and present inputs (and the current states of the hidden Markov models corresponding to each test node). The algorithms proposed to build such models are also quite different from the temporal tree induction method proposed in the present paper.

References

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Wadsworth International (California).

Das, G.; Gunopulos, D.; and Mannila, H. 1997. Finding similar time series. In *Proc. of KDD'97*.

Jordan, M. I.; Ghahramani, Z.; and Saul, L. K. 1996. Hidden markov decision trees. Technical Report 9606, MIT, Computational Cognitive Science.

Jordan, M. I. 1994. A statistical approach to decision tree modeling. In *Proc. of the Seventh Annual ACM Conference on Computational Learning Theory*. New York: ACM Press.

Manganaris, S. 1997. *Supervised classification with temporal data*. Ph.D. Dissertation, Vanderbilt University.

Mannila, H.; Toivonen, H.; and Verkamo, A. I. 1997. Discovery of frequent episodes in event sequences. Technical Report C-1997-15, University of Helsinki, Department of Computer Science.

Paliouras, G. 1997. *Refinement of Temporal Constraints in an Event Recognition System using Small Datasets*. Ph.D. Dissertation, Department of Computer Science, University of Manchester.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1:81–106.

Torgo, L. 1995. Applying propositional learning to time-series prediction. In et al, Y. K., ed., *Proc. of ECML-95 Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*.

Wehenkel, L.; Lebrevelec, C.; Trotignon, M.; and Batut, J. 1997. A probabilistic approach to the design of power systems protection schemes against blackouts. In *Proc. IFAC-CIGRE Symp. on Control of Power Plants and Power Systems*, 506–511.

Wehenkel, L. 1997. Machine learning approaches to power-system security assessment. *IEEE Expert, Intelligent Systems & their Applications* 12(3).

Wehenkel, L. 1998. *Automatic learning techniques in power systems*. Boston: Kluwer Academic.