Pattern Extraction for Time Series Classification

Pierre Geurts †

University of Liège, Department of Electrical and Computer Engineering
Institut Montefiore, Sart-Tilman B28, B4000 Liège, Belgium
geurts@montefiore.ulg.ac.be
†Research fellow, FNRS

Abstract. In this paper, we propose some new tools to allow machine learning classifiers to cope with time series data. We first argue that many time-series classification problems can be solved by detecting and combining local properties or patterns in time series. Then, a technique is proposed to find patterns which are useful for classification. These patterns are combined to build interpretable classification rules. Experiments, carried out on several artificial and real problems, highlight the interest of the approach both in terms of interpretability and accuracy of the induced classifiers.

1 Introduction

Nowadays, machine learning algorithms are becoming very mature and well understood. Unfortunately, most of the existing algorithms (if not all) are dedicated to simple data (numerical or symbolic) and are not adapted to exploit relationships among attributes (such as for example geometric or temporal structures). Yet, a lot of problems would be solved easily if we could take into account such relationships, e.g. temporal signals classification. While a laborious application of existing techniques will give satisfying results in many cases (our experiments confirm this), we believe that a lot of improvement could be gained by designing specific algorithms, at least in terms of interpretability and simplicity of the model, but probably also in terms of accuracy.

In this paper, the problem of time signals classification is tackled by means of the extraction of discriminative patterns from temporal signals. It is assumed that classification could be done by combining in a more or less complex way such patterns. For the aim of interpretability, decision trees are used as pattern combiners. The first section formally defines the problem of multivariate time-series classification and gives some examples of problems. Related work in the machine learning community is discussed here as well. In the next section, we experiment with two "naive" sets of features, often used as first means to handle time series with traditional algorithms. The third section is devoted to the description of our algorithm which is based on pattern extraction. The last section presents experiments with this method. Finally, we conclude with some comments and future work directions.

2 The (multivariate) time-series classification problem

2.1 Definition of the problem

The time series classification problem is defined by the following elements:

- A universe U of objects representing dynamic system trajectories or scenarios. Each object[1], o, is observed for some finite period of time $[0, t_f(o)]$[2].
- Objects are described by a certain number of temporal candidate attributes which are functions of object and time, thus defined on $U \times [0, +\infty[$. We denote by $a(o, t)$ the value of the attribute a at time t for the object o.
- Each object is furthermore classified into one class, $c(o) \in \{c_1, ..., c_M\}$.

Given a random sample LS of objects from the universe, the goal of the machine learning algorithm is to find a function $f(o)$ which is as close as possible to the true classification $c(o)$. This function should depend only on attribute values (not on object), ie. $f(o) = f(\mathbf{a}(o, .))$ where $\mathbf{a}$ denotes the vector of attributes. The classification also should not depend on absolute time values. A consequence of this latter property is that the model should be able to classify every scenario whatever its duration of observation.

Note that alternative problems should also be addressed. For example, in [7], a temporal detection problem is defined where the goal is to find a function $f(o, t)$ of object and time which can detect as soon as possible scenarios of a given class from past and present attribute values only.

Temporal attributes have been defined here as continuous functions of time. However, in practice, signals need to be sampled for representation in computer memory. So, in fact, each scenario is described by the following sequence of vectors: $(\mathbf{a}(o, t_0(o)), \mathbf{a}(o, t_1(o)), ..., \mathbf{a}(o, t_n(o)))$ where $t_i(o) = i \cdot \triangle t(o)$ and $i = 0, 1, \cdots, n(o)$. The number of time samples, $n(o)$, may be object dependent.

2.2 Description of some problems

The possible application domains for time series classification are numerous: speech recognition, medical signal analysis, recognition of gestures, intrusion detection... In spite of this, it is difficult to find datasets which can be used to validate new methods. In this paper, we use three problems for evaluation. The first two datasets are artificial problems used by several researchers in the same context. The last one is a real problem of speech recognition.

Control chart (CC) This dataset was proposed in [1] to validate clustering techniques and used in [2] for classification. Objects are described by one temporal attribute and classified into one of six possible classes (see [1] or [2] for more details). The dataset we use was obtained from the UCI KDD Archive [3] and contains 100 objects of each class. Each time series is defined by 60 time points.

[1] In what follows, we will use indifferently the terms scenario and object to denote an element of U.

[2] Without loss of generality we assume start time of scenario being always 0.

Cylinder-Bell-Funnel (CBF) This problem was first introduced in [12] and then used in [11, 8, 2] for validation. The goal is to separate three classes of object: cylinder(c), bell(b) and funnel(f). Each object is described by one temporal attribute given by:

$$a(o,t) = \begin{cases} (6+\eta)\cdot\chi_{[a,b]}(t)+\epsilon(t) & \text{if } c(o)=c, \\ (6+\eta)\cdot\chi_{[a,b]}(t)\cdot(t-a)/(b-a)+\epsilon(t) & \text{if } c(o)=b, \\ (6+\eta)\cdot\chi_{[a,b]}(t)\cdot(b-t)/(b-a)+\epsilon(t) & \text{if } c(o)=f, \end{cases}$$

where $t \in [1,128]$ and $\chi_{[a,b]}(t) = 1$ if $a \leq t \leq b$, 0 otherwise.

In the original problem, η and $\epsilon(t)$ are drawn from a standard normal distribution $N(0,1)$, a is an integer drawn uniformly from $[16,32]$ and $b-a$ is an integer drawn uniformly from $[32,96]$. Figure 1 shows an example of each class. As in [11], we generate 266 examples for each class using a time step of 1 (i.e. 128 time points per object).

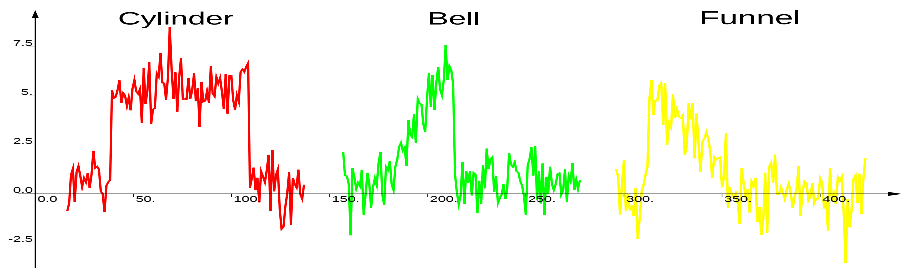


Fig. 1. An example of each class from the CBF dataset

This dataset attempts to catch some typical properties of time domain. Hence, the start time of events, a, is significantly randomized from one object to another. As we will argue later that our algorithm can cope with such temporal shifting of events, we generate another version of the same datasets by further emphasizing the temporal shifting. This time, a is drawn from $[0,64]$ and $b-a$ is drawn from $[32,64]$. We will call this dataset CBF-tr (for "CBF translated").

Japanese vowels (JV) This dataset is also available in the UCI KDD Archive and was built by Kudo et al. [10] to validate their multidimensional curve classification system. The dataset records 640 time series corresponding to the successive utterance of two Japanese vowels by nine male speakers. Each object is described by 12 temporal attributes corresponding to 12 LPC spectrum coefficients. Each signal is represented in memory by between 7 to 29 time points. The goal of machine learning is to identify the correct speaker from this description.

2.3 Related work in machine learning

Several machine learning approaches have been developed recently to solve the time series classification problem. Manganaris [11] for example constructs piecewise polynomial models for univariate signals and then extracts features from this representation for classification. Kadous [8] extracts parameterized events from signals. These events are clustered in the parameters space and the resulting prototypes are used as a basis for creating classifiers. Kudo et al. [10] transforms multivariate signals into binary vectors. Each element of this vector corresponds to one rectangular region of the space value-time and tells if the signal passes through this region. A method of their own, subclass, builds rules from these binary vectors. Gonzales et al. [2] extends (boosted) ILP systems with predicates that are suited for the task of time series classification.

All these approaches share some common characteristics. First, authors are all interested in getting interpretable rules more than in accuracy. We will give a justification for that in the next section. Second, they use some discretization techniques to reduce the search spaces for rules (from simple discretization to piecewise modeling or clustering). All of them extract rules which depend on absolute time value. This makes difficult the detection of properties which may occur at variable time position and can be a serious limitation to solve some problems (for example the CBF-tr problem). The technique we propose does not have this drawback.

3 Experiments with naive sampling and classical methods

Before starting with dedicated approaches for time series classification, it is interesting to see what can be done with classical machine learning algorithms and a naive approach to feature selection. To this end, experiments were carried out with two very simple sets of features:

- Sampled values of the time series at equally spaced instants. To handle time series of different durations, time instants are taken relative to the duration of the scenario. More precisely, if n is the number of time instants to take into account, each temporal attribute a gives rise to n scalar attributes given by: $a(o, t_f(o)\frac{i}{n-1})$, $i = 0, 1, \cdots, n-1$.
- Segmentation (also proposed in [8]). The time axis of each scenario is divided into n equal-length segments and the average value of each temporal attribute along these segments are taken as attributes.

The two approaches give $n \cdot m$ scalar attributes from m temporal ones. Note that while the first approach is fully independent of time, the second one takes into account the temporal ordering of values to compute their average and so is doing some noise filtering.

These two sets of attributes have been tried on the three problems described in section 2.2 as inputs to three different learning algorithms: decision tree (the particular algorithm we used is described in [13]), decision tree boosting [5] and the one-nearest neighbor. Results in terms of error rates are summarized in Table 1 for increasing value of n. They were obtained by ten-fold cross-validation for the first three problems and by validation on an independent test set of size 370 for the last one[3] (JV). The best result is boldfaced in every row.

[3] This division was suggested by the donors of this dataset [10]

Table 1. Results with simple sampling methods

CC	Number of steps				
Sampling	3	5	10	30	60
DT	31.67 ± 4.94	17.00 ± 4.40	11.67 ± 4.08	8.17 ± 3.53	**6.83 ± 2.29**
Boosting	24.33 ± 4.67	12.33 ± 3.96	5.17 ± 3.20	2.00 ± 1.63	**1.50 ± 1.17**
1-NN	31.00 ± 4.72	17.00 ± 3.14	8.66 ± 3.78	1.50 ± 1.38	**1.83 ± 1.38**
Segment	3	5	10	30	60
DT	12.33 ± 4.73	16.33 ± 4.46	**3.50 ± 2.52**	7.50 ± 2.50	6.83 ± 2.29
Boosting	6.67 ± 4.01	11.83 ± 5.45	**1.50 ± 1.38**	2.00 ± 2.08	1.50 ± 1.17
1-NN	8.16 ± 4.24	12.00 ± 4.46	**0.50 ± 1.07**	1.33 ± 1.00	1.83 ± 1.38

CBF	Number of steps				
Sampling	8	16	32	64	128
DT	9.33 ± 3.35	7.83 ± 3.42	7.50 ± 2.61	**7.33 ± 2.49**	9.83 ± 3.83
Boosting	6.50 ± 3.02	4.00 ± 2.00	2.33 ± 1.70	**2.17 ± 1.83**	3.50 ± 2.17
1-NN	7.66 ± 4.16	3.83 ± 2.24	2.00 ± 1.63	1.33 ± 1.63	**1.16 ± 1.30**
Segment	8	16	32	64	128
DT	4.67 ± 2.45	**2.67 ± 1.33**	4.33 ± 2.38	7.67 ± 4.29	9.83 ± 3.83
Boosting	3.17 ± 1.57	**0.67 ± 1.11**	1.67 ± 1.83	2.17 ± 1.98	3.50 ± 2.17
1-NN	2.33 ± 2.00	**0.50 ± 0.76**	0.50 ± 1.07	1.00 ± 1.10	1.16 ± 1.30

CBF-tr	Number of steps				
Sampling	8	16	32	64	128
DT	**19.17 ± 3.18**	23.50 ± 6.81	20.67 ± 3.82	21.67 ± 3.80	23.83 ± 6.95
Boosting	14.17 ± 3.52	10.50 ± 4.54	6.67 ± 2.79	**5.00 ± 2.36**	7.17 ± 3.58
1-NN	19.33 ± 3.89	8.00 ± 4.70	5.33 ± 2.56	**3.50 ± 2.03**	3.83 ± 2.89
Segment	8	16	32	64	128
DT	14.17 ± 5.34	14.17 ± 4.55	12.83 ± 5.58	**12.67 ± 3.82**	23.83 ± 6.95
Boosting	12.67 ± 5.23	6.00 ± 3.27	**4.17 ± 1.86**	5.33 ± 2.08	7.17 ± 3.58
1-NN	12.00 ± 2.33	3.00 ± 2.66	**1.66 ± 1.82**	2.66 ± 1.52	3.83 ± 2.89

JV	Number of steps				
Sampling	2	3	5	7	
DT	14.86	**14.59**	19.46	21.08	
Boosting	6.76	**5.14**	5.68	6.22	
1-NN	**3.24**	3.24	3.24	3.78	
Segment	1	2	3	4	5
DT	18.11	17.30	**12.97**	19.46	17.03
Boosting	9.46	7.84	6.76	6.76	**6.22**
1-NN	6.49	**3.51**	3.51	3.78	4.05

There are several things to say about this experiment. There exists an optimal value of n which corresponds to the best tradeoff between bias and variance for each problem[4]. This optimal value could be automatically fixed by cross-validation. Segmentation rather than simple sampling is highly beneficial on all datasets except for the last one. The best error rates we get with this simple approach are very good with respect to previously published results on these problems (i.e. with dedicated temporal approaches). The best method is 1-NN on all problems. Decision trees do not work well while boosting is very effective. As boosting works mainly by reducing the variance of a classifier, the bad results of decision trees may be attributed to a high variance. Furthermore, their interpretability is also questionable because of the choice of attributes. Indeed, how to understand for example a rule like "if $a(o,32) < 2.549$ and $a(o,22) < 3.48$ then $c(o) =$ bell" which was induced from the CBF dataset? Although very simple and accurate, this rule does not make obvious the temporal increase of the signal peculiar to the bell class (see Figure 1).

One conclusion of this experiment is that very good results can be obtained with simple feature sets but by sacrificing interpretability. This observation justifies the fact that most of the machine learning research on temporal data have focused on interpretability rather than on accuracy.

4 Pattern extraction technique

Why are the approaches adopted in the previous section not very appropriate? First, even if the learning algorithm gives interpretable results, the model will not be comprehensible anymore in terms of the temporal behavior of the system. Second, some very simple and common temporal features are not easily represented as a function of such attributes. For example, consider a set of sequences of n random numbers in $[0,1]$, $\{a_1, a_2, ..., a_n\}$, and classify a sequence into the class c_1 if three consecutive numbers greater than 0.5 can be found in the sequence whatever the position. With a logical rule inducer (like decision trees) and using a_1, a_2,...,a_n as input attributes, a way to represent such a classification is the following rule: if ($a_1 > 0.5$ and $a_2 > 0.5$ and $a_3 > 0.5$) or ($a_2 > 0.5$ and $a_3 > 0.5$ and $a_4 > 0.5$) or ... or ($a_{n-2} > 0.5$ and $a_{n-1} > 0.5$ and $a_n > 0.5$) then return class c_1. Although the initial classification rule is very simple, the induced rule has to be very complex. This representation difficulty will result in a high variance of the resulting classifier and thus in poor accuracy. The use of variance reduction techniques like boosting and bagging often will not be enough to restore the accuracy and anyway will destroy interpretability.

In this paper, we propose to extend classifiers by allowing them to detect local shift invariant properties or patterns in time-series (like the one used to define the class c_1 in our example). The underlying hypothesis is that it is possible to classify a scenario by combining in a more or less complex way such pattern detections. In what follows, we first define what patterns are and how

[4] for a comprehensive explanation of the bias/variance dilemna, see for example [13]

to construct binary classification tests from them. Then, we propose to combine these binary tests into decision trees. In this context, piecewise constant modeling is proposed to reduce the search space for candidate patterns.

Pattern definition. A possible way to define a pattern is to use a limited support reference signal and then say that the pattern is detected at a particular position of a test signal if the distance between the reference signal and the test signal at this position is less than a given threshold. In other words, denoting by $p(.)$ a signal defined on the interval $[0, t_p]$ and by $a(.)$ a signal defined on the interval $[0, t_a]$ with $t_a \geq t_p$, we would say that the pattern associated to $p(.)$ is detected in $a(.)$ at time t' ($t_p \leq t' \leq t_a$) if:

$$d(p(.), a(.), t') = \frac{1}{t_p} \int_{t'-t_p}^{t'} (p(t - t' + t_p) - a(t))^2 dt < d_p, \tag{1}$$

where d_p is the minimal allowed distance to the pattern (euclidian distance). A binary classification rule may be constructed from this pattern by means of the following test:

$$T(o) = \text{True} \Leftrightarrow \exists t' \in [t_p, t_f(o)], : d(p(.), a(o, .), t') < d_p \tag{2}$$

$$\Leftrightarrow [\min_{t' \in [t_p, t_f(o)]} d(p(.), a(o, .), t')] < d_p \tag{3}$$

where a is a temporal attribute.

Integration with decision trees. As we are mainly interested in interpretable classifiers, the way we propose to combine these binary tests is to let them appear as candidate tests during decision tree induction. Each step of decision tree induction consists in evaluating a set of candidate tests and choosing the one which yields the best score to split the node (see [13] for more details). In the present context, candidate tests are all possible triplets (a, p, d_p) where a is a temporal candidate attribute.

Once we have chosen an attribute a and a pattern p, the value of d_p which realizes the best score may be computed similarly as the optimum discretization threshold for numerical attribute. Indeed, test (3) is equivalent to a test on the new numerical attribute $a_n(o) = \min_{t'} d(p(.), a(o, .), t')$.

The number of candidate patterns $p(.)$ could be a priori huge. So, it is necessary to reduce the search space for candidate patterns. A first idea is to construct patterns from subsignals extracted from the signals appearing in the learning set (each one corresponding to temporal attributes of objects). However, it is still impossible in practice to consider every such subsignal as a candidate pattern. Even assuming a discrete time representation for the datasets, this step will remain prohibitive (e.g. there are 8256 different subseries in a time series of 128 points). Also, patterns extracted from raw signals may be too complex or too noisy for interpretation. The solution adopted here is to first represent the signal by some piecewise model and then use the discontinuity points of this model to define interesting patterns. By choosing the complexity of the model (the number of time axis pieces), we are thus able to fix the number of patterns to consider.

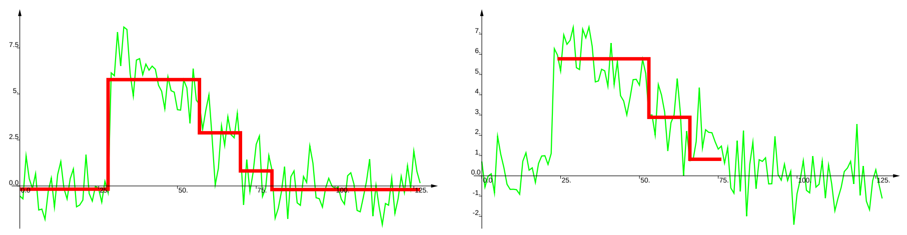


Fig. 2. Left, the regression tree modeling of a signal with 5 intervals. Right, the detection of a pattern extracted from this signal in another signal of the same class

Regression tree modeling. In this paper, a simple piecewise constant model is computed for each signal. Regression trees are used to build recursively this model. The exact algorithm is described in Table 2. It follows a best first strategy for the expansion of the tree and the number of segments (or terminal nodes) is fixed in advance. The discretization of an example signal in 5 segments is reproduced in the left part of Figure 2.

From a discretized signal $\hat{a}(.)$, the set of candidate signals $p(.)$ is defined as follows:

$$P = \{p(.) \text{ on } [0, t_j - t_i] | t_i, t_j \in D, t_i < t_j, p(t) = \hat{a}(t_i + t)\},$$

where D is the set of discontinuity points defined in Table 2. The size of this set is $n \cdot (n+1)/2$ if n is the number of segments. The right part of Figure 2 shows a pattern extracted from the left signal and its minimal distance position in another signal.

Node splitting for tree growing. So, candidate signals $p(.)$ during node splitting will be extracted from piecewise constant modeling of learning set time series. Unfortunately, even with this discretization/segmentation, it will be intractable to consider every subsignals in the learning set, especially when the learning set is large. A simple solution to overcome this difficulty is to randomly sample a subset of the scenarios from the learning set as references for defining the subsequences. In our experiments, one scenario will be drawn from each class. This further simplification should not be limitative because interesting patterns are patterns typical of one class and these patterns (if they exist) will presumably appear in every scenario of the class. Eventually, our final search algorithm for candidate tests when splitting decision tree nodes is depicted in Table 3.

5 Experiments

We first experiment with the pattern extraction technique described in the previous section. Then, as a byproduct of regression tree modeling is a reduction of the space needed to store the learning set, we further test its combination with the nearest neighbor algorithm.

Table 2. Discretization of time signals by regression trees

Let us denote by $\text{mean}_{[t_1,t_2]}(a(.))$ and $\text{var}_{[t_1,t_2]}(a(.))$ respectively the mean and the variance of the signal $a(.)$ on the interval $[t_1, t_2]$:

$$\text{mean}_{[t_1,t_2]}(a(.)) = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} a(t)dt$$

$$\text{var}_{[t_1,t_2]}(a(.)) = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} (a(t) - \text{mean}_{[t_1,t_2]}(a(.)))^2 dt$$

To discretize $a(.)$ on $[t_1, t_2]$ with N_{max} pieces:

1. set $D = \{t_1, t_2\}$, the set of discontinuity points; set $L = \{[t_1, t_2]\}$, the set of intervals; set $\hat{a}(t) = mean_{[t_1,t_2]}a(.)$ on $[t_1, t_2]$, the model for $a(.)$;
2. set $N_p = 1$, the current number of time segments (pieces);
3. if $N_p = N_{max}$ then stop and return $\hat{a}(.)$;
4. find $[t_i, t_j]$ in L such that $(t_j - t_i).\text{var}_{[t_i,t_j]}(a(.))$ is maximal (best first strategy),
5. remove $[t_i, t_j]$ from L,
6. find $t^* \in [t_i, t_j]$ which maximizes the variance reduction:

   $$\triangle\text{var}(t^*) = (t_j - t_i)\text{var}_{[t_i,t_j]}(a(.)) - (t^* - t_i)\text{var}_{[t_i,t^*]}(a(.)) - (t_j - t^*)\text{var}_{[t^*,t_j]}(a(.))$$

7. set $\hat{a}(t) = mean_{[t_i,t^*]}a(.)$ on $[t_i, t^*]$ and $\hat{a}(t) = mean_{[t^*,t_j]}a(.)$ on $[t^*, t_j]$;
8. $N_p = N_p + 1$; add $[t_i, t^*]$ and $[t^*, t_j]$ to L; add t^* to D.
9. go to step 3

5.1 Decision tree with pattern extraction

Experiments have been carried out in exactly the same test conditions as in section 3. For regression tree modeling, increasing values of the number of time segments, N_{max}, were used (11 only for CC). Results are summarized in Table 4 and commented below.

Accuracy. On the first three datasets, the new method gives significant improvements with respect to decision tree (compare with table 1). As expected, the gain in accuracy is especially impressive on the CBF-tr problem (from 12.67 to 2.33). This problem is also the only one where our temporal approach is better than boosting with simple features. On JV, our approach does not improve accuracy with respect to naive sampling. Several explanations are possible. First, this is the only dataset with more than one attribute and our method is not able to capture properties distributed on several signals. Second, there are 9 classes and only 270 examples in this problem and the recursive partitioning of decision tree is known to suffer in such conditions. Third, it seems also that the temporal behavior is not very important in this problem, as 1-NN with only two values (the start and end values of each attribute) gives the best results (3.24 %).

From this experiment, the optimal number of segments appears to be problem dependent. In practice, we would thus need a way to tune this parameter. Besides cross-validation, various methods have been proposed to fix the number of segments for piecewise modeling (for example, the MDL principle in [11]).

Table 3. Search algorithm for candidate tests during tree growing

For each temporal attribute a, and for each class c:

- select an objet o of class c from the current learning set,
- discretize the signal $a(o,.)$ to obtain $\hat{a}(o,.)$
- compute the set P of subsignals $p(.)$ from $\hat{a}(o,t)$
- for each signal $p(.) \in P$
  - compute the optimal threshold d_p,
  - if the score of this test is greater than the best score so far, retain the triplet $(a(.), p(.), d_p)$ as the best current test.

Table 4. Results of decision tree with patterns

	Number of pieces			
DB	3	5	7	11
CC	**2.33 ± 1.70**	3.17 ± 2.03	3.33 ± 2.58	3.00 ± 1.63
CBF	4.00 ± 2.71	2.00 ± 1.45	**1.17 ± 1.67**	
CBF-TR	9.33 ± 5.68	3.83 ± 2.24	**2.33 ± 1.33**	
JV	22.97	21.62	**19.4**	

In our case, we could also take advantage of the pruning techniques (or stop-splitting criteria) in the context of regression tree induction. We have still to experiment with these methods.

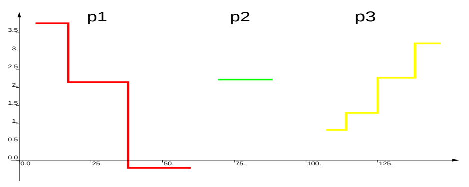
Interpretability. By construction, the rules produced by our algorithm are very readable. For example, a decision tree induced from 500 examples of the CBF problem gives the very simple rules described visually at Figure 3. The extracted patterns are confirmed by the definition of the problem. This decision tree gives an error rate of 1.3% on the 298 remaining examples. For comparison, a decision tree built from the mean values on 16 segments (the features set which yields the best result in Table 1) contains 17 tests and gives an error rate of 4.6%.

5.2 Regression tree modeling with 1-NN

As the discretization by regression trees yields a compact version of the original time-series, it would be interesting to combine it into a nearest neighbor classifier. The main advantage will be a reduction of the space needed to memorize the learning set. The algorithm proceeds as follows. First, all signals in the learning set are discretized by regression trees using the same maximum number of pieces. Then, the distance between a test object o and an object o' of the learning set is defined by:

$$d(o,o') = \frac{1}{\min(t_f(o), t_f(o'))} \sum_{i=1}^{m} \int_0^{\min(t_f(o),t_f(o'))} (a_i(o,t) - \hat{a}_i(o',t))^2 dt. \quad (4)$$

So, discretized signals $\hat{a}_i(o',t)$ for learning set objects are compared to full signals for test objects. To deal with objects which are defined on different intervals, we simply truncate the longest one to the duration of the shortest one.



if p1 then funnel
else if p2 and p3 then bell
else if p2 and not p3 then funnel
otherwise cylinder

Fig. 3. classification rules for the CBF problems

Table 5. Results of 1-NN with regression tree modeling

	Number of pieces				
DB	1	3	5	7	11
CC	38.83 ± 9.40	0.50 ± 0.76	**0.17 ± 0.50**	0.33 ± 1.00	0.33 ± 0.67
CBF	46.17 ± 6.15	10.50 ± 2.69	1.33 ± 1.45	0.50 ± 0.76	**0.33 ± 0.67**
CBF-tr	43.00 ± 5.57	23.00 ± 7.18	4.50 ± 3.58	**2.33 ± 1.70**	2.50 ± 1.86
JV	11.35	5.67	4.86	**4.59**	4.59

Experiments have been carried out with increasing values of the number of time segments (from 1 to 11). Results are reported in Table 5. On the first three problems, the accuracy is as good as the best accuracy which was obtained in Table 1 and the number of time segments to reach this accuracy is very small. The compression of the learning set is particularly impressive on the CC dataset where only three values are enough to reach an almost perfect accuracy. On the other hand, regression tree modeling decreases the accuracy on JV with respect to 1-NN and only two values per attribute. Again, the optimal number of pieces is problem dependent. In the context of 1-NN, leave-one-out is an obvious candidate method to determine this parameter.

6 Conclusion and future work

In this paper, we have presented a new tool to handle time series in classification problems. This tool is based on a piecewise constant modeling of temporal signals by regression trees. Patterns are extracted from these models and combined in decision trees to give interpretable rules. This approach has been compared to two "naive" feature selection techniques. The advantage of our technique in terms of interpretability is undeniable. In terms of accuracy, better results can be obtained by using either boosting or 1-NN with naive features. However, in some problems where start time of characteristic events are highly variable (like in CBF-tr), accuracy can be improved by pattern extraction. Eventually, even if our main goal was interpretability, our extended decision trees can also be combined in boosted classifiers where they are very unlikely to destroy accuracy with respect to the naive feature selection.

In the future, we will consider extensions of our method along several axis. First, there are still many possible improvements of the pattern extraction algorithm. For instance, we can experiment with other piecewise models (linear by hinges model, polynomial,...) or with more robust sampling strategies during node splitting. As already mentioned, we also need a way to automatically adapt the number of time segments during tree growing. Second, one limitation of our pattern definition is that it does not allow the detection of shrunk or extended versions of the reference pattern along the time axis. Several distances have been proposed to circumvent this problem, for example dynamic time warping [4,9] or probabilistic pattern matching [6]. We believe that such distances could be combined with our approach but at the price of a higher complexity. Eventually, there exist many problems where the exact ordering of patterns appearing in signals is crucial for classification. In these cases, the combination of patterns by simple logical rules would not be enough and dedicated methods should be developed which could take into account temporal constraints between patterns.

References

1. R. J. Alcock and Y. Manolopoulos. Time-series similarity queries employing a feature-based approach. In *Proc. of the 7th Hellenic Conference on Informatics*, Ioannina, Greece, 1999.
2. C. J. Alonso Gonzalez and Juan J. Rodriguez Diez. Time series classification by boosting interval based literals. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, (11):2–11, 2000.
3. S. D. Bay. The UCI KDD archive, 1999. http://kdd.ics.uci.edu.
4. D. J. Berndt and J. Clifford. Finding patterns in time series: A dynamic programming approach. In *Advances in Knowledge discovery and data mining.* AAAI Press/MIT Press, 1996.
5. Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proc. of the 13th International Conference on Machine Learning*, 1996.
6. X. Ge and P. Smyth. Deformable markov model templates for time-series pattern matching. In *Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 81–90, Boston, MA, August 2000.
7. P. Geurts and L. Wehenkel. Early prediction of electric power system blackouts by temporal machine learning. In *Proc. of ICML-AAAI'98 Workshop on "AI Approaches to Times-series Analysis"*, Madison (Wisconsin), 1998.
8. M. W. Kadous. Learning comprehensible descriptions of multivariate time series. In *Proceedings of the Sixteenth International Conference on Machine Learning, ICML'99*, pages 454–463, Bled, Slovenia, 1999.
9. E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proc. of the 6th Intl. Conf. on Knowledge Discovery and Data Mining (KDD)*, pages 285–289, Boston, MA, August 2000.
10. M. Kudo, J. Toyama, and M. Shimbo. Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, 20(11-13):1103–1111, 1999.
11. S. Manganaris. *Supervised classification with temporal data.* PhD thesis, Vanderbilt University, 1997.
12. N. Saito. *Local feature extraction and its application using a library of bases.* PhD thesis, Department of Mathematics, Yale University, 1994.
13. L. Wehenkel. *Automatic learning techniques in power systems.* Kluwer Academic, Boston, 1998.