Chapter 12

Time series classification by pattern extraction

In this chapter, we propose a new decision tree based method for time series classification which aims at providing interpretable rules. The method is based on the extension of the set of candidate tests so as to enable them to detect local shift invariant properties or patterns in time series. These patterns are extracted from a piecewise constant representation of time series derived automatically by a regression tree induction method from the time series provided in the learning sample. The accuracy and the interpretability of the method is verified and discussed on the basis of our six test problems.

12.1 Introduction

The conclusion of the previous chapter is that in the context of time series classification there is a stronger need for interpretable models than for better accuracy since some of the techniques (e.g. "segment and combine") already yield very good results in terms of accuracy. From the discussion of section 11.2.3, it appears also that the lack of interpretability partially comes from the fact that some characteristic patterns of a class of signals are generally not easily representable with simple tests involving only one low-level attribute at the time. In this chapter, we thus propose to extend decision tree classifiers by explicitly allowing them to detect local shift invariant properties or patterns in time series. The underlying hypothesis is that it is possible to identify these patterns from the learning sample and to classify the scenarios by combining in a more or less simple way such pattern-detection tests.

In what follows, we first define the algorithm which is used to search for patterns at decision tree nodes. Then, we validate the method in terms of accuracy and interpretability on our test problems. Eventually, we discuss related work and some future works directions.

12.2 Proposed algorithm

The proposed algorithm is similar to classical decision tree induction, where the set of candidate tests are extended from elementary tests based on scalar attributes (e.g. $a(o) < a_{th}$) to more complex tests defined on temporal attributes. All other parts of the standard decision tree algorithm are kept unchanged (i.e the score measure, top-down induction, stop-splitting rules,...) and like they are defined in Chapter 5 and 6. Thus, in what follows we will essentially focus on the mechanism proposed to generate these new candidate tests and how they are used to classify objects described by temporal attributes. Before going into the details of the algorithm, we start with an intuitive presentation of the ideas which have motivated our extension of decision tree tests for handling the time series classification problem.

12.2.1 Intuitive presentation

With low-level features, tests at decision tree nodes are of the form $[A(o,t') < a_{th}]$. The use of such simple tests makes it difficult to represent temporal peculiarities such as shift invariant properties, and above all, leads to complex decision trees which are hardly interpretable. If we want to provide interpretable rules, we need to extend the representation power of decision trees by taking into account the temporal ordering of low-level attributes, in such a way that the individual tests can be translated into some easily understandable properties characterizing the signals. We limit our ambition here to extend tests at tree nodes to detect local shift invariant properties in temporal attributes. Furthermore, like in standard decision tree induction and for the same reasons (interpretability and efficiency), we restrict our method to tests involving only one attribute at a time.

Let us introduce our test definition by an example. In the CBF problem, scenarios of the bell class are characterized by an increasing ramp which may appear at any temporal position in the signal. It would be convenient to be able to represent such characteristics with one single decision tree test in order to produce understandable rules, like: "if there is a portion of the signal which looks like an increasing ramp then it is very likely to be a scenario of the bell class". What defines the pattern here is the description "increasing ramp". To determine such patterns during decision tree induction, we could provide several predefined types of patterns (local minimum, maximum, increasing ramp, decreasing ramp,...) together with an algorithm able to detect these latter in a signal. One drawback of this approach is that it would need prior information about a problem in order to define the dictionary of candidate patterns in an appropriate way.

For the sake of flexibility, we propose here to let the shape of patterns free. Thus, an *elementary* pattern is a limited support signal of any shape, and we consider that such a pattern is present in a signal, if there exists a portion of this signal which is close enough to the pattern, in other words if the *euclidian distance* between the pattern and some portion of the signal is smaller than a given threshold. So, decision tree tests could be defined by the choice of a temporal attribute, an elementary pattern, and a threshold on the distance. An object would be propagated to the left successor if some portion of the given attribute signal is close enough to the pattern according to the threshold and to the right successor otherwise. For example, a pattern defining the bell class is given in Figure 12.1. Its detection in a particular scenario is also plotted in Figure 12.1. With this definition, a pattern is easily interpretable since it can be plotted like any signal.

Unfortunately, this definition of patterns turns out to be too restrictive and sometimes unable to uncover the temporal structure of a problem. Indeed, with this definition there is no way to represent in a simple way temporal constraints among patterns, like for example the fact that two patterns must appear in a given order. Therefore, we further extent our set of candidate tests to incorporate what we call *complex* patterns, namely ordered lists of several elementary patterns; we say that a complex pattern is detected in a signal if all its elementary patterns are detected *in the specified order*. In what follows we will generally leave the qualification of *complex* implicit.

At each step of decision tree induction, among a set of candidate tests one which realizes the best score is selected to split a node. Thus, in the present context we need to define a set of candidate patterns to evaluate at each test node. Since such patterns

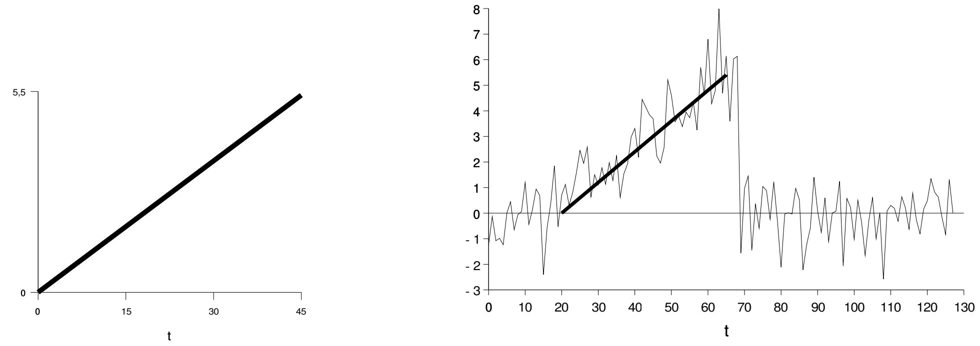

Figure 12.1: A pattern which characterizes the bell class in the CBF problem and its detection in one scenario

are useful only if they appear in at least some of the time series present at this node, we propose to extract the set of candidate patterns directly from the sample of signals appearing in the local learning set at the current node. Candidate elementary patterns will thus be chosen from subsignals of the signals appearing in the learning sample.

Nevertheless, for each scenario, there are a priori many possible subsignals (e.g. there are $C_{128}^{2} = 8128$ elementary subsignals in a time series defined on 128 time points) and they may be possibly corrupted by noise (see the signal of Figure 12.1 for example). So, it is not practically feasible nor desirable to consider all possible sequences of all subsignals of all samples as candidate patterns. The solution adopted here to circumvent these problems is to extract patterns from a piecewise constant approximation derived from the sample signals. This approach will both filter the noise and reduce the size of the search space of candidate patterns. In our work, we build such piecewise constant models with an efficient algorithm inspired from regression tree induction. In practice, it is still too costly to consider every sample signal of the local learning set to define such candidate patterns. Therefore, in the method we propose, we randomly draw only one scenario from each class and extract the best patterns from these scenarios. This should not be too limitative since if there exist patterns typical of some class, they will presumably appear in at least one of these scenarios selected at random.

Eventually, the general form of the algorithm which searches for the best test is given in Table 12.1 and illustrated in Figure 12.2. It computes the best pattern for each temporal attribute and then selects the optimal test among all attributes. Each step of the procedure is described in detail in what follows. First, patterns and tests are formally defined. Then, the algorithm to compute piecewise constant models for time series is presented. Eventually, we detail the heuristics we use to extract patterns from piecewise constant representations.

12.2.2 Elementary and complex patterns and detection tests

Elementary patterns

Like temporal attributes, an *elementary* pattern $p(.)$ is defined as a time series $(p_0, p_1, \ldots, p_{P-1})$ (with the same discretization step as temporal attributes in the dataset). We say that this pattern is detected at a certain position in a time series if the distance between the pattern and the time series at this position is lower than a given threshold. More formally, denoting by $a(.) = (a_0, a_1, \ldots, a_{N-1})$ a time series, with $N \geq P$, and by d_p the threshold on distance, the pattern $p(.)$ is detected in the time series $a(.)$ if there exists

Table 12.1: Overview of the algorithm for pattern extraction during tree growing

---

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

- select an object o of class c from the local learning sample ls;
- compute a piecewise constant model for $A(o,.)$, denoted $\hat{a}(.)$;
- search for the best pattern from $\hat{a}(.)$ and define an optimal test for this pattern;
- if the score of this test is better than the best score so far, retain this test as the current best test.

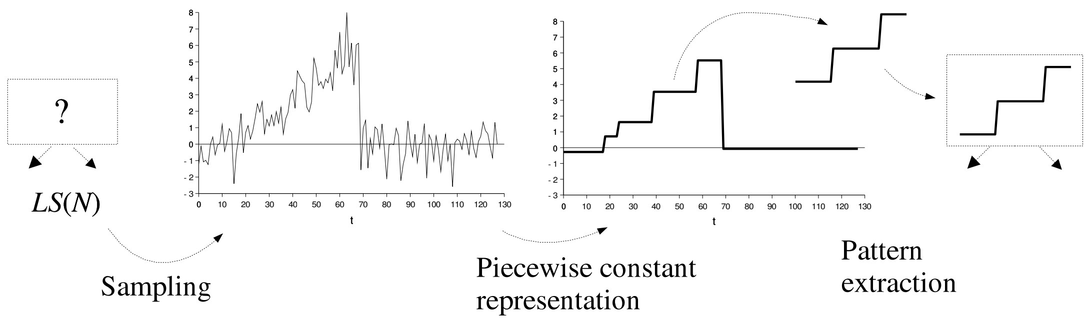

Figure 12.2: Pattern extraction to define decision tree tests

some index i ($i = P-1, P+1, \dots, N-1$) such that:

$$d(p(.), a(.), i) = \sum_{j=i-P+1}^{i} (p_{j-i+P+1} - a_j)^2 < d_p. \tag{12.1}$$

The idea of using a threshold on the euclidian distance to detect a pattern allows some variation (or noise) on the amplitude with respect to the reference pattern. A binary classification rule is constructed from this pattern by means of the following test:

$$\begin{aligned} T(o) = \text{True} \quad &\Leftrightarrow \quad \exists i \in \{P-1, P, \dots, n(o)-1\} | d(p(.), A(o,.), i) < d_p & (12.2) \\ &\Leftrightarrow \quad [\min_{i \in \{P-1, P, \dots, n(o)-1\}} d(p(.), A(o,.), i)] < d_p & (12.3) \end{aligned}$$

where A is a temporal attribute and $n(o)$ is the number of time points for the time series representation of the object o. So, to detect a pattern, a window of size P is slided along the time axis and the distance between the pattern and the signal on this window is computed. If it becomes lower than the threshold, the pattern is detected. Note that distance (12.1) looks like a convolution product between the pattern and the signal at position i.

Complex patterns

The notion of pattern is extended to take into account a succession of elementary patterns. A *complex* pattern is defined as an ordered list of several elementary patterns $\mathcal{P} = (p_1(.), p_2(.), \dots, p_K(.))$, defined respectively by P_k ($k = 1, \dots, K$) time points. Given a distance threshold d_p, a complex pattern is detected in a temporal signal $a(.)$ defined on N time points ($N \geq \sum_{k=1}^{K} P_k$) if there exist i_1, i_2, $\dots$,i_K such that:

$$\sum_{k=1}^{K} d(p_k(.), a(.), i_k) < d_p \tag{12.4}$$

where $d(.)$ is defined by 12.1 and i_k ($k = 1, \dots, K$) are integers in $\{0, 1, \dots, N-1\}$ satisfying:

$$\begin{aligned} i_1 &\geq P_1 - 1, & (12.5) \\ i_k &\geq i_{k-1} + P_k, \forall k = 2, \dots, K, & (12.6) \\ i_K &< N. & (12.7) \end{aligned}$$

The first two inequalities translate the fact that elementary patterns may not overlap; the last one means that they must all be fitted in the signal duration. Figure 12.3 illustrates one possible configuration of i_k values in the case of three patterns.

The derivation of a binary test for a temporal attribute a from eqn. (12.4) is naturally following eqn. (12.2):

$$T(o) = \text{True} \Leftrightarrow [\min_{i_1, \dots, i_K | C_{\mathcal{P}}(i_1, \dots, i_K)} \sum_{k=1}^{K} d(p_k(.), A(o,.), i_k)] < d_p, \tag{12.8}$$

where $C_{\mathcal{P}}(i_1, \dots, i_K)$ corresponds to conditions (12.5),(12.6), and, (12.7).

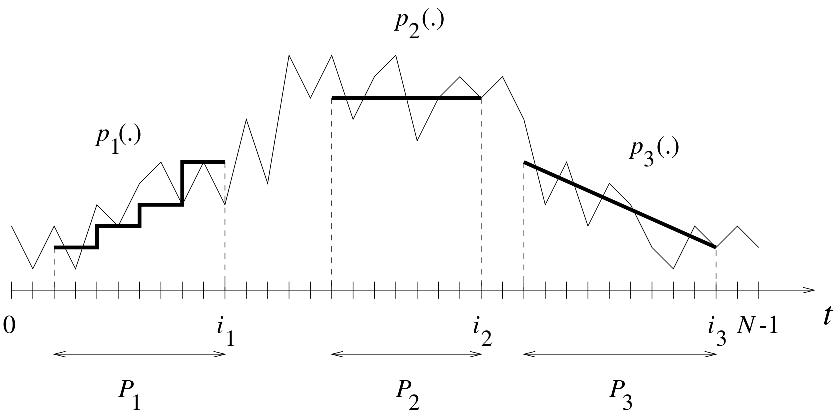

Figure 12.3: One possible configuration of three patterns in a time series

Candidate test threshold optimization

Candidate tests during tree induction are thus defined by triplets $< A, \mathcal{P}, d_p >$ where A is a temporal attribute, $\mathcal{P}$ is a complex pattern (denoted by its sequence of elementary patterns), and d_p is a distance threshold. Of course, other more complex definitions are possible. For example, we could use instead a different threshold for each subpattern or constrain the difference between the time of occurrence of two successive patterns $i_{k+1} - i_k$. However, we will see in what follows that this simple definition still results in a quite complex induction algorithm.

Notice that the threshold d_p corresponding to a combination $< A, \mathcal{P} >$ plays exactly the same role than thresholds used in standard decision trees to test numerical attributes. Actually, for a given choice of $< A, \mathcal{P} >$, the optimal value of this threshold can be determined with the same exhaustive search procedure (or any other technique) used in standard tree growing. Indeed, for a given sample of objects ls and combination of attribute and pattern $< A, \mathcal{P} >$, it is sufficient to first compute for each object in ls the minimum distance

$$d_{A,\mathcal{P}}(o) = \min_{i_1,\ldots,i_K | C_{\mathcal{P}}(i_1,\ldots,i_K)} \sum_{k=1}^{K} d(p_k(.), A(o,.), i_k), \tag{12.9}$$

and then build an optimal test in the form $[d_{A,\mathcal{P}}(o) < d_p]$ using the standard discretization technique to determine d_p.

Thus, during tree induction, the minimum in (12.9) will be computed for many objects and many candidate patterns and this computation might be very time consuming. For example, by taking into account the constraints (12.5), (12.6), and, (12.7) but assuming that the lengths of patterns are small with respect to N, there are about C_N^K possible values for the vector $(i_1, \ldots, i_K)$. Thus a computation of the minimum in (12.9) by enumeration is practically feasible only for a small number of elementary patterns. Fortunately, there exists an exact Viterbi-like algorithm based on the dynamic programming principle that solves this problem with a complexity $O(N \sum_{k=1}^{K} P_k)$. This algorithm is presented in Appendix D.

Candidate complex patterns

As we will explain in the sequel, we propose to generate for each attribute, a set of candidate patterns in three steps: (i) we randomly select an object for each class; (ii) for each such selected object we generate a piecewise constant approximation of the given attribute; (iii) from the piecewise constant approximation we generate a set of candidate patterns in a combinatorial way while imposing some constraints. The following two subsections explain the two last steps in more detail.

12.2.3 Piecewise constant modeling with regression trees

A general piecewise model for a time series is obtained by segmenting the temporal interval into S segments and representing the signal in these segments by some parametric model of a given form. Here, we consider the simplest piecewise model where the time signal is represented in each segment by a constant. More complex models include for example piecewise linear models where the signal is represented by a linear function of time, $a.t+b$, which parameters, a and b, are different from one segment to another. Given the number of segments S, the goal of piecewise modeling is to find parameters which yield a model as close as possible to the initial signal. Here, the distance between the signal and the model is computed as the squared error.

Let us denote by

$$(a_0, a_1, \ldots, a_{N-1}) \tag{12.10}$$

the time series we want to model (in our case, this will correspond to the time series representation of a temporal attribute for a given object). A piecewise constant model with S segments for this time series is represented for example by a sequence of pairs:

$$((d_0, \hat{a}_0), \ldots, (d_{S-1}, \hat{a}_{S-1})), \sum_{s=0}^{S-1} d_s = N \tag{12.11}$$

where d_s is an integer denoting the length of the s^{th} segment (in time points) and $\hat{a}_s$ is its amplitude. Denoting by i_s the index in the time series at which the s^{th} segment starts:

$$i_s = \sum_{j=0}^{s-1} d_j, s = 1, \ldots, S, \tag{12.12}$$

with $i_0 = 0$ and $i_S = N$, the error of this model at predicting the value of the original time series is given by:

$$Err(a, \hat{a}) = \sum_{s=0}^{S-1} \sum_{j=i_s}^{i_{s+1}-1} (\hat{a}_s - a_j)^2. \tag{12.13}$$

The goal of piecewise constant model induction is to determine from the time series the parameters S, $\hat{a}_s$, and d_s. The induction of this model is generally divided in two separate steps:

- Determine the value of S (i.e. the complexity of the piecewise constant model);
- For a given number of segments S, determine $\hat{a}_s$ and d_s which minimize the error (12.13).

Note that since error (12.13) is null when S is equal to the number of time-points in the series (in which case the piecewise constant model corresponds to the time series itself), the value of S can not be determined as the one which minimizes the error (this is the bias/variance tradeoff applied to this problem). The approaches we adopt to resolve these tasks, in the context of our algorithm, are developed in turn below. Note that in the literature several alternative algorithms have been proposed to build piecewise models for time series data. We postpone their discussion until Section 12.4.

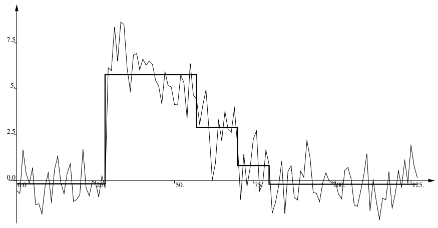

Figure 12.4: the regression tree modeling of a signal with 5 segments.

Determination of $\hat{a}_s$ and d_s by regression trees

Given the value of S, the goal of piecewise constant model induction is to select the parameters $\hat{a}_s$ and d_s (or i_s) that minimize (12.13). Once the lengths of the segments are fixed, the estimation of the values $\hat{a}_s$ that minimize (12.13) are given by averaging the signal on each segment:

$$\hat{a}_s = \frac{1}{d_s} \sum_{j=i_s}^{i_{s+1}-1} a_j, s = 0, \ldots, S-1. \qquad (12.14)$$

So the main problem is the determination of the segment length d_k. A dynamic programming algorithm produces an exact (optimal) solution in order $O(SN^2)$ time. In this work, we propose a solution based on regression tree induction which yields a suboptimal solution in order $O(SN)$.

Indeed, using as input attribute the time t and as output attribute the signal values $A(o,t)$ of a given object o, regression tree induction will precisely build a piecewise constant model as the one we search for. To build a tree from a time series like (12.10), every time point is considered as a classical object where the index i represents the value of the only input attribute for this object and a_i corresponds to the output value. The resulting algorithm for the segmentation of a signal in S segments is described in Table 12.2 (on page 205). Since the goal of learning is to produce a partition of the time interval into a fixed number of segments, a best-first strategy is used for the expansion of the tree[1] and further splitting is stopped when there are already S segments. The discretization by this algorithm of an example signal in 5 segments is reproduced in Figure 12.4.

Besides efficiency, the main advantage of this algorithm is that the segmentation for increasing values of S are nested and hence all models for S going from 1 to N are built in order $O(N^2)$ (vs $O(N^4)$ with the dynamic programming algorithm). We will see below that this facilitates the search for the right value of S.

Determination of the number of segments S

The piecewise constant model is introduced in our extension of decision trees in two goals: reduce the search space for patterns and filter out the noise in time series to enhance interpretability. Hence, two criteria may direct the search for the right number of segments:

- select S in such a way that low error rates are obtained when the resulting patterns are used in the time series classification trees.
- model at best the time series at hand by filtering noise.

[1]Contrary to the "most promising" strategy applied in the experiments of Chapter 5, algorithm of Table 12.2 follows a true best first approach.

Table 12.2: Discretization of time series by regression trees

---

Let us define the following quantities:

$$\begin{aligned}\text{mean}(i,j) &= \frac{1}{j-i}\sum_{l=i}^{j-1} a_i && (12.15)\\ \text{var}(i,j) &= \frac{1}{j-i}\sum_{l=i}^{j-1}(a_l - \text{mean}(i,j))^2 && (12.16)\\ \text{Score}(i,j,l) &= (j-i)\text{var}(i,j) - (j-l)\text{var}(l,j) - (l-i)\text{var}(i,l), i < l < j && (12.17)\\ & && (12.18)\end{aligned}$$

As for regression trees, Score(i,j,l) is the variance reduction caused by the split of the segment between index i and j at the position l.

Build_PCM($(a_0,\ldots,a_{N-1})$,S):
(build a piecewise constant model into S segments for the time series $(a_0,\ldots,a_{N-1})$.)

Set $L = \{(\text{mean}(0,N),0,N)\}$, the ordered list of current segments described by their mean value, the start index of this segment and the start index of the next segment. Then proceed as follows:

1. If the length of L is equal to S then go to step 7.
2. For each triplet (a,i,j) in L:
   - Determine $\text{Score}(i,j) = \max_{l=i+1,\ldots,j-1}\text{Score}(i,j,l)$;
   - Determine $l^*(i,j) = \arg\max_{l=i+1,\ldots,j-1}\text{Score}(i,j,l)$;

   NB. These optimal scores and partitions do not need to be recomputed at each iteration for all segments in L. Only the newly added segments actually need to be recomputed.
3. Find $(a^*,i^*,j^*) = \arg\max_{(a,i,j)\in L}\text{Score}(i,j)$;
4. Remove (a^*,i^*,j^*) from L;
5. Insert in order in L the two new segments:
   - $(\text{mean}(i^*,l^*(i^*,j^*)),i^*,l^*(i^*,j^*))$,
   - $(\text{mean}(l^*(i^*,j^*),j^*),l^*(i^*,j^*),j^*)$.
6. Go to step 1
7. From the s^{th} element of L, denoted by (a,i,j), define the s^{th} segment as:

$$\begin{aligned}\hat{a}_s &= a\\ d_s &= j-i\end{aligned}$$

8. Return $((d_0,\hat{a}_0),\ldots,(d_S,\hat{a}_S))$;

In both cases, the number of segments should regulate a bias/variance tradeoff. In the first goal, the more segments we use, the more candidate patterns are considered during node splitting (see Section 12.2.4) and hence bias and variance of the resulting decision trees should be affected by this parameter. On the other hand, as S increases, error (12.13) (and hence, bias) decreases but the model starts matching the noise in the time series (overfitting) and hence the optimal value of S should not be the highest.

In our experiments, we compare two approaches to fix the number of segments. The first one consist in using the same value of S throughout the construction of a decision tree and select the value which minimizes the error rate (estimated by cross-validation or by validation on an independent sample). The second method will consist in determining S without taking into account the future use of the model for pattern extraction but rather focusing on producing good models for the time series under consideration. To select the right complexity in this goal, we can take advantage of the method used to post-prune regression trees. The post-pruning approach we propose in the context of time series is given below.

Post-pruning piecewise constant models

In the context of regression trees, post-pruning algorithms aim at selecting the right complexity of the model. To this end, a validation set is used to give an independent estimate of the error and this estimate is used to select the right complexity of the model. Here, the only available data is a series of N time points. One possible way to split this data set into two sets is to subsample the time series with twice the initial discretization step. A time series $(a_0, \ldots, a_{N-1})$ gives thus rise to two[2] time series $(a_0, a_2, \ldots, a_{N-2})$ and $(a_1, a_3, \ldots, a_{N-1})$. The first series is used to learn the model parameters according to the algorithm of Table 12.2 and the second one is used to estimate the error of this model. If the discretization step is sufficiently small with respect to the system dynamics, there will not be an important loss of information by using subsampling. on the other hand, this approach will be useful to determine when the regression tree starts overfitting the data.

The algorithm which automatically learns the number of segments is depicted in Table 12.3. It builds a sequence of piecewise models of increasing complexity. Although that does not appear explicitly in Table 12.3, the whole sequence is obtained by applying only once the algorithm of Table 12.2 with $S = N/2$ and saving at each step intermediate models. In consequence, the complexity of this algorithm is at worst $O(N^2)$.

For example, right part of Figure 12.5 plots resubstitution error (estimated on a_{even} in Table 12.3) and validation error (estimated on a_{odd}) for the discretization of the signal represented in the right of the same figure with an increasing number of segments. The validation error goes through a minimum for 6 segments. Eventually, the model of 6 segments built from the whole time series is represented in the right part of Figure 12.5.

12.2.4 Search strategy for candidate tests

Tests are defined by triplets $< A, \mathcal{P}, d_p >$ where A is some attribute, $\mathcal{P}$ is a complex pattern, i.e. a sequence of elementary patterns, and d_p is the distance threshold. During node splitting, search is carried out to find a good test in terms of the score. We already noticed that once we have chosen an attribute A and a pattern $\mathcal{P}$, the value of d_p which realizes the best score may be computed like in the standard tree growing algorithm as

[2] the notations assume that N is even.

Table 12.3: Automatic selection of the number of segments by cross-validation

---

Build_PCM_with_pruning($(a_0, \ldots, a_{N-1})$)**:**
(build a piecewise constant model for a time series by selecting its complexity by cross-validation.)

- Split the time series into $a_{\text{even}} = (a_0, a_2, \ldots, a_{N-2})$ and $a_{\text{odd}} = (a_1, a_3, \ldots, a_{N-1})$.
- Compute the sequence of piecewise constant model $\{M_1, \ldots, M_{N/2}\}$, where $M_s =$ Build_PCM(a_{even}, s),
- Select the number of segments s^* which minimizes the error of M_{s^*} at predicting a_{odd}. If the s^{th} model is written $M_s = ((d_0, \hat{a}_0), \ldots, (d_{s-1}, \hat{a}_{s-1}))$, then this error is computed by:

$$Error(s) = \sum_{k=0}^{s-1} \sum_{j=i_k}^{i_{k+1}-1} (\hat{a}_k - a_{2j+1})^2.$$

  and $s^* = \arg\min_{s=1,\ldots,N} Error(s)$.
- Return Build_PCM($(a_0, \ldots, a_{N-1})$,s^*);

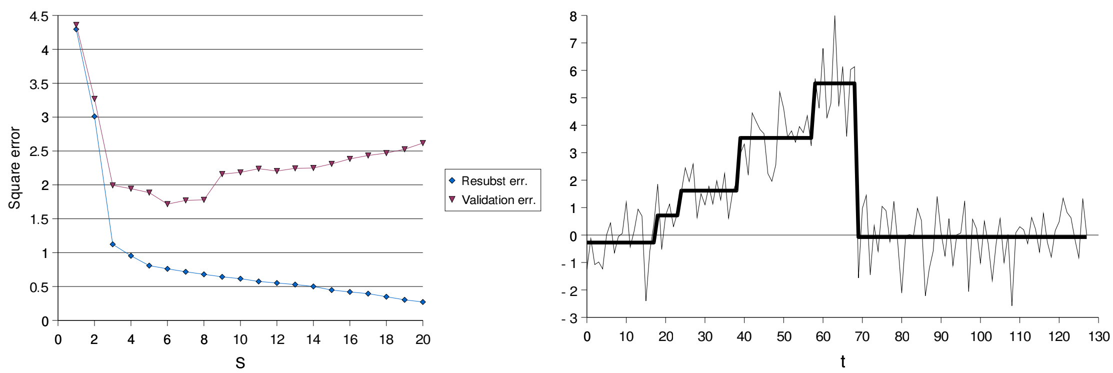

Figure 12.5: Left, resubstitution and validation errors with increasing segment numbers, right, the optimum model with six segments for the time series

the optimum discretization threshold on the numerical attribute:

$$d_{A,\mathcal{P}}(o) = \min_{i_1,\ldots,i_K | C_{\mathcal{P}}(i_1,\ldots,i_K)} \sum_{k=1}^{K} d(p_k(.), A(o,.), i_k). \tag{12.19}$$

So, the main problem is the determination of a set of candidate complex patterns $\mathcal{P}$ and thus, in other words a sequence of elementary patterns $(p_1(.),\ldots,p_K(.))$ for arbitrary K.

As suggested in the introduction of this section, in our approach candidate patterns are extracted from the piecewise representation of some temporal attribute of an object appearing in the local learning sample (selected at random, for each class). From a time series $A(o,.)$ corresponding to an object o, the segmentation algorithm described above produces a piecewise constant model $\hat{a}(.)$ which can be represented by a sequence:

$$\hat{a}(.) = ((d_0, \hat{a}_0),\ldots,(d_{S-1}, \hat{a}_{S-1})). \tag{12.20}$$

Every subsequence of this sequence equivalently represents a piecewise constant signal defined on a smaller time interval. We propose to consider as candidate elementary patterns $p_k(.)$ subsequences $((d_{i_k}, \hat{a}_{i_k}),\ldots,(d_{j_k}, \hat{a}_{j_k}))$, with $0 \leq i_k \leq j_k < S$. To define complex patterns, it seems natural to impose the further constraint that elementary patterns extracted in this way are not overlapping each other in $\hat{a}(.)$ and also that their ordering of appearance in $\hat{a}(.)$ is respected in the definition of the complex pattern, i.e. $j_k < i_{k+1}$, $k = 1,\ldots,K-1$. We further impose that two elementary patterns are separated in $\hat{a}$ by at least one segment, i.e. $j_k < i_{k+1} + 1$, $k = 1,\ldots,K-1$.

Following these lines, we define a complex pattern from the representation (12.20) by a mask of S boolean values:

$$\underline{b} = (b_0,\ldots,b_{S-1}) \tag{12.21}$$

Each boolean variable b_i is true if the corresponding segment is taken into account to define the pattern. A sequence of consecutive true values for b_i delimited by false values defines an elementary pattern $p_k(.)$. For example, the boolean vector (F,T,T,F,T,T,F) defined on a piecewise representation with 7 segments defines a complex pattern composed of two elementary patterns. The first elementary pattern is the subsignal corresponding to the second and the third segment and the second elementary pattern is defined by the fifth and the sixth segments. When applied to the piecewise representation of Figure 12.6 (p.210), this boolean vector defines a complex pattern which detects the strict succession of the elementary patterns p_1 and p_2 represented in the same figure.

So searching for interesting patterns amounts to searching the space of boolean vectors of size S. There are in principle 2^S different complex patterns in a signal decomposed into S pieces. If we want the search algorithm to remain tractable in most cases, we thus need a way to reduce the search complexity. We propose here to use a greedy strategy. The resulting algorithm which is looking for a pattern in a signal $\hat{a}$ is described in Table 12.4. Starting from a vector of only true values[3], we flip at each step the previously unchanged boolean value which leads to the pattern which together with the optimum threshold on (12.19) causes the highest increment (or lowest decrease) in the score. So, the search path in the space of patterns is composed of S steps, starting from a unary pattern composed of all S segments ($b_i = \text{True}, \forall i$) and ending with a unary pattern composed of only one segment among the S ones ($b_i = \text{False}\ \forall i \neq j$). Thus, this heuristic reduces the number of patterns to evaluate from 2^S to $\frac{S(S+1)}{2}$.

To summarize, for a given temporal attribute, the procedure of Table 12.4 should be applied on every signal $A(o,.)$ corresponding to some *master* objects selected from the

[3]We could also start from a vector of only false value.

Table 12.4: Pattern extraction from a segmented signal

Pattern_search(ls,A,$\hat{a}$):
(search the best test from the signal $\hat{a}$ for the attribute A and on the subset ls.)

- Denote by S the number of segments in the representation of $\hat{a}(.)$.
- Compute the following sequence of pattern definition masks:

$$\{\underline{b}^0, \underline{b}^1, \ldots, \underline{b}^{S-1}\},$$

  where:

  - $b_i^0 = \mathrm{T}$, $\forall i = 0, \ldots, S-1$,
  - $\underline{b}^{k+1}$, for $k = 1, \ldots, S-1$, is obtained from $\underline{b}^k$ by flipping the previously unflipped true value (there are $S-k$ of them in $\underline{b}^k$) which yields the highest score:

$$\mathrm{Score}(ls, < A, \mathcal{P}(\underline{b}^{k+1}, \hat{a}(.)), d_p^*(\underline{b}^{k+1}, \hat{a}(.)) >),$$

    where $\mathcal{P}(\underline{b}^{k+1}, \hat{a}(.))$ is the complex pattern defined by $\underline{b}^{k+1}$ from $\hat{a}(.)$ and $d_p^*(\underline{b}^{k+1}, \hat{a}(.))$ is defined by:

$$d_p^*(\underline{b}^{k+1}, \hat{a}(.)) = \arg\max_d \mathrm{Score}(ls, < A, \mathcal{P}(\underline{b}^{k+1}, \hat{a}(.)), d >).$$

- Among this sequence, select the vector $\underline{b}^*$ which yields the best score.
- Return the test $< a, \mathcal{P}(\underline{b}^*, \hat{a}), d_p^*(\underline{b}^*, \hat{a}) >$.

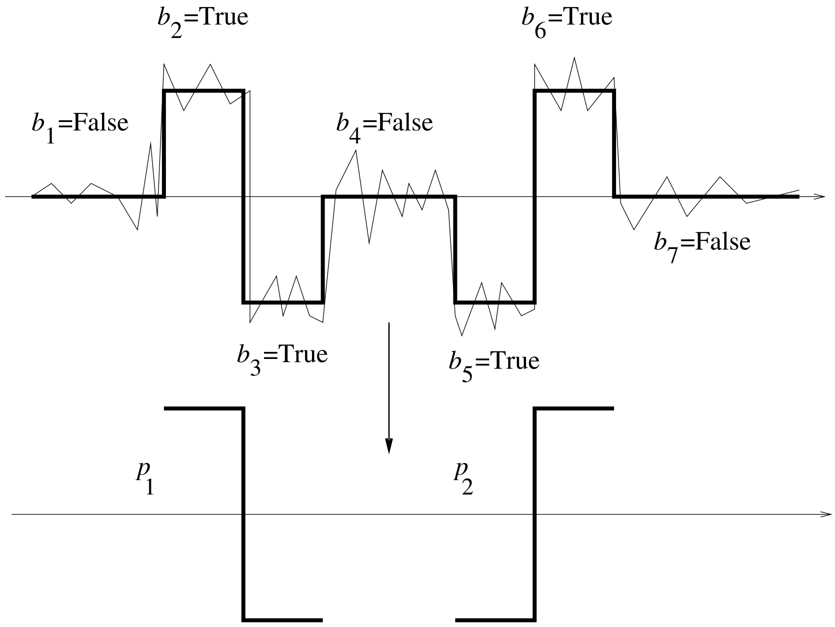

Figure 12.6: Definition of a pattern (composed of two elementary patterns) by a boolean mask $\underline{b}$

local subset ls. As already explained, in our experiments only one master object will be drawn from each class (and for each candidate attribute) at each node of the tree. The resulting search algorithm for candidate tests when splitting decision tree nodes is depicted in Table 12.5. It replaces the determination of the optimal test in the standard decision tree induction algorithm of Table 5.1 of Chapter 5 (p. 86).

12.2.5 Computational complexity

Let us estimate the complexity of the algorithm of Table 12.5 for a learning sample ls of size N. Let N_t be the number of time points used to define scenarios of the datasets and assume that this number is the same for every scenario. The complexity of the segmentation of a time series (construction and pruning) is at most $O(N_t^2)$. Assuming that the resulting piecewise constant model gives S segments, $\frac{S(S+1)}{2}$ different patterns are considered by the algorithm of Table 12.4. For each of them, the minimal distance to the N objects must be computed and the best threshold on distance searched. The complexity of the computation of the minimal distance for a complex pattern in one scenario is at most $O(N_tS)$ for patterns defined on S segments (see Appendix D for the algorithm) and thus this computation for the N scenarios is $O(NN_tS)$. The determination of the best threshold requires to sort the learning sample and to pass once through all objects. This can be done in $O(N \log N)$.

The whole process of segmentation and best pattern search must be repeated one time for each temporal attribute and each class. If we assume that the number of segments given by piecewise modeling is always the same and denoted by S, all in all, the complexity of the procedure of Table 12.5 is:

$$O(c_1 nmN_t^2 + c_2 nm\frac{S(S+1)}{2}(c_3 NN_tS + c_4 N \log N))), \tag{12.22}$$

where n is the number of temporal attributes and m is the number of classes. In practice, the dominant part of this algorithm results from the computation of the minimal distance

Table 12.5: Search algorithm for candidate tests during tree growing

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

- select an object o of class c from ls,
- compute either:
  - $\hat{a}(.) = \text{Build\_PCM}(A(o,.), S)$ for a predefined number of segments S, or
  - $\hat{a}(.) = \text{Build\_PCM\_with\_pruning}(A(o,.))$ by using the pruning algorithm;
- compute the test $< A, \mathcal{P}, d_p >= \text{Pattern\_search}(ls, A, \hat{a})$;
- if the score of this test is better than the best score so far, retain the triplet $< A, \mathcal{P}, d_p >$ as the best current test

of the pattern to each scenario of the learning sample. On the other hand, the computation of the piecewise models and the search for the best threshold are negligible.

Globally, this procedure must be repeated at each test node of the tree and hence the complexity of the complete tree induction algorithm depends of course on the complexity of the tree. In practice however, the number of nodes often increases less than linearly with respect to the size of the learning sample and hence, the average complexity of the full algorithm is in practice closer to linear than to quadratic with respect to the size of the learning sample. It is on the other hand linear with respect to the number of candidate attributes and quadratic with respect to the number of sampling steps and with the number of segments used for piecewise constant modeling.

12.2.6 Discussion

In this section, we have proposed a way to extend decision trees to handle the time series classification problem. Tests are extended to detect local shift invariant properties or patterns in time series. These patterns are extracted from a piecewise constant representation of time series of scenarios which appear in the local subsample at decision tree node. The price to pay for the increase of flexibility and interpretability is in terms of computational efficiency, which is mainly governed by part of the dynamic programming algorithm which computes the distance from a complex pattern to a time series. The resulting method is a combination of bottom-up and top-down approaches in machine learning: bottom-up because properties of an object of one class are generalized to classify all objects of the same class and top-down because the tree growing algorithm essentially finds more and more specific tests to classify objects.

Note that the algorithm is random since it defines patterns from scenarios which are randomly selected in the local sample. However, the hope is that specificity of the chosen scenario will be filtered by the pattern extraction technique and hence this will not result in too much variance of the resulting models.

Finally, note also that this extension of the decision tree induction method is especially adapted to cope with shift invariant properties and our pattern definition is not invariant to changes in the time scale. We will discuss in the last section of this chapter some directions to extend this method so as to handle scale invariant properties and other

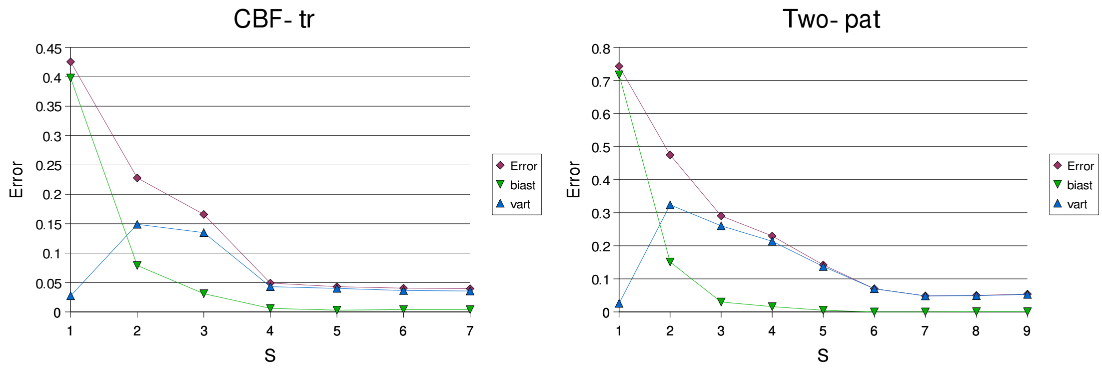

Figure 12.7: Effect of the number of segments on bias and variance

peculiarities relevant in temporal problems.

12.3 Validation of the method

In this section, the proposed pattern detection based tree induction technique is validated on our test problems. First, we assess the accuracy of the method. Among other things, we study the effect of the way of choosing the number of segments for piecewise constant modeling on bias, variance, and error rates. Then, some indicative results in terms of computation times are given. Eventually, we discuss the interpretability of the models which are produced.

12.3.1 Accuracy

Bias/variance study

One parameter which should regulate some bias/variance tradeoff in this method is the number of segments used to represent signals by a piecewise constant model. Indeed, when S increases the number of patterns which are searched during induction increases. Figure 12.7 plots the evolution of classification bias and variance with the number of segments and, left on the CBF-tr problem and right on the Two-pat problem. For each value of S, 50 fully grown trees are learned from learning samples of size 300 drawn from a pool sample of size 3000 and tested on a test sample of size 1000. During tree induction, time series are segmented by the algorithm of Table 12.2 without pruning and with the same value of S at each tree node.

As expected, on both problems, bias decreases with the number of segments. It is almost null on the CBF-tr problem at 4 segments. From our knowledge of this problem, this is not surprising since this is the minimum number of segments that allows to distinguish between an increasing and a decreasing ramp. On the two-pat problems, the minimum bias is reached for larger values of S. And indeed, from the problem definition, we know that the signals in this problem are best modeled by a piecewise constant model in 7 segments. On both problems however, variance first increases from 1 to 2 segments and then monotonically decreases with the segment number. On the Two-pat problems, we observe a very slight increase after 7 segments.

The fact that variance does not increase with the number of segments may be attributed to two effects. Figure 12.8 shows on the two problems the evolution of the tree complexity (the total number of nodes) with respect to the number of segments. When the number of segments increases, the complexity of decision trees decreases very much. It goes from 201

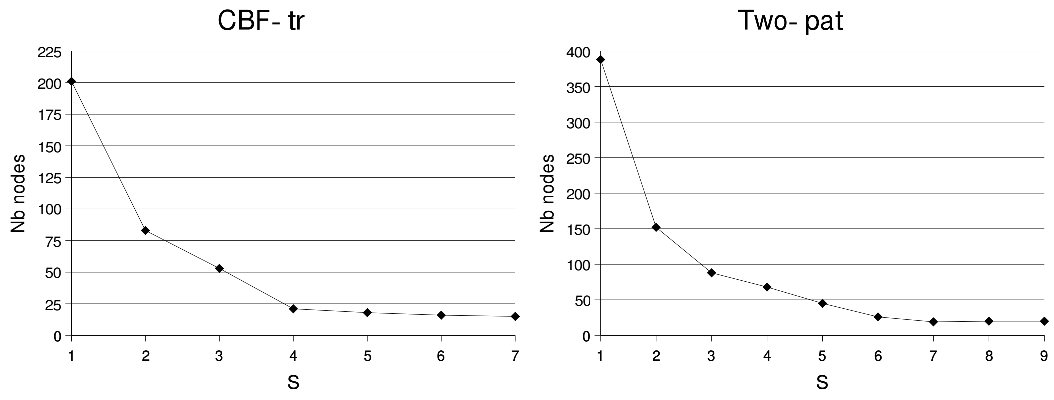

Figure 12.8: Evolution of decision tree complexity with respect to the number of segments

nodes to 15 nodes on the CBF-tr problem and from 388 nodes to 19 nodes on the Two-pat problem. When S is small, we know from the problem definitions that it is impossible to build a decision tree which generalizes well (the bias is non null). However, because of the noise, it is nevertheless possible to discriminate perfectly between the different classes in the learning sample and hence tests at deep levels in the tree overfit very much the data. The small variance and high bias with only one segment is due to the fact that in this case, it is very difficult to discriminate between patterns even on the learning sample (which is confirmed by the high tree complexity). On the other hand, when S is large, a small tree suffices to realize low error on the learning sample and this small tree turns out to generalize better than larger trees obtained with smaller values of S.

Second, even if more patterns are considered when the number of segments increases, they do not necessarily translate into more discriminative power for the test. Indeed, the distance to a complex pattern (12.4) is the sum of many terms. The addition or deletion of a segment in the piecewise representation does not necessarily change the ordering of objects with respect to this distance. In other word, the minimal distance is quite stable with respect to the model (at least for large enough values of S). This is confirmed by the fact that the complexity of the tree remains almost constant when S increases behind the value of S which minimizes the bias (see Figure 12.8). However, these two phenomena are probably related to the fact that the method is well adapted to this kind of problems. On other problems, variance could increase with the number of segments.

Table 12.6 compares three variants of the algorithm on both problems. The first line corresponds to the minimum of error on Figure 12.7. The second line is obtained by using the post-pruning algorithm presented in Table 12.3 in the same condition. The third line is obtained from the second line by pruning the decision trees from a validation sample of size 1000 (according to the algorithm described in Chapter 6). On both problems, the bias is very small. This shows that the model and the search procedure is well adapted to the problems. In consequence, the variance is responsible for almost all the error. The effect of pruning is not important: it reduces slightly the variance and the complexity while leaving the bias unchanged. The complexity of full and pruned trees is small. On CBF-tr, in average, decision trees use 5 tests and on Two-pat, they use 7 tests (the minimum is 2 tests to separate 3 classes and 3 tests to separate 4 classes). On these problems, using post-pruning to select the number of segments for the piecewise representation is beneficial. It reduces both error rates and complexity with respect to the utilization of the same value for all time series.

Table 12.6: DT with pattern extraction on CBF-tr and Two-pat

	Mean error	compl	bias_T	var_T	var_{KW}	$Err_{\hat{P}}$	$\text{bias}^2_{\hat{P}}$	$\text{var}_{\hat{P}}$
CBF-tr								
Full DT ($S = 7$)	0.0396	15	0.0040	0.0356	0.0316	0.0264	0.0053	0.0211
Full DT (S learned)	0.0327	14	0.0060	0.0267	0.0260	0.0218	0.0044	0.0174
Pruned DT (S learned)	0.0322	11	0.0060	0.0262	0.0251	0.0208	0.0048	0.0160
Two-pat								
Full DT ($S = 7$)	0.0484	19	0.0000	0.0484	0.0427	0.0242	0.0029	0.0214
Full DT (S learned)	0.0472	19	0.0000	0.0472	0.0423	0.0236	0.0025	0.0212
Pruned DT (S learned)	0.0445	15	0.0000	0.0445	0.0396	0.0216	0.0026	0.0189

Table 12.7: DT with pattern extraction on other problems

	Error	Compl
CC		
Full DT ($S = 6$)	2.67 ± 1.53	22
Full DT (S learned)	3.33 ± 2.11	24
CBF		
Full DT ($S = 6$)	1.00 ± 0.75	15
Full DT (S learned)	1.25 ± 0.56	13
JV		
Full DT ($S = 6$)	19.19	45
Full DT (S learned)	21.62	49
Auslan		
Full DT ($S = 4$)	15.50 ± 6.50	32
Full DT (S learned)	17.5 ± 10.55	29

With respect to the approach of the previous chapter, the error on CBF-tr is slightly greater than with the segment and combine algorithm while it is slightly lower on the Two-pat problem. However, although very close, a comparison of the bias/variance profile of both methods shows that they work differently. Segment and combine is characterized by a low variance while the present method is characterized by a low bias and a relatively high variance. We have seen in previous chapters that a high variance seems to be the price to pay for interpretable models.

Other problems

We further experiment with this method on the other test problems used in the preceding chapter. Table 12.7 reports for each problem the best error obtained with a constant value of S and the error obtained by determining the value of S with the post-pruning algorithm. In each case, trees are fully grown and the average complexity is reported in the table.

On all problems, the results are better without the post-pruning algorithm although both approaches give trees of comparable complexity. The difference is less pronounced on CC and CBF. On JV and Auslan, it may be attributed to the fact that the number of time-points is lower in average and hence subsampling is not appropriate to select the right complexity.

On CC and CBF, the method gives better results than the one obtained with decision trees and low-level features. On CBF, the result are even competitive with the best results obtained with variance reduction techniques (random trees and segment and combine). However, while on Auslan, results are only slightly better than the results obtained by decision trees with low-level attributes, on JV, there are significantly worse.

These rather bad results on the two last problems can be explained. Indeed, on JV, we have seen that the temporal behavior was not important for classification (since models built with only two low-level features give very good results) and hence the increase of flexibility of our approach for handling temporal behaviors is useless in this case. Also, both problems are characterized by a large number of classes and many attributes in comparison to the size of the learning sample and we know that decision tree induction especially suffers in such situations because of the recursive partitioning.

12.3.2 Computing times

To give an idea of the computing times of the proposed algorithm, we ran it with the selection of segment numbers by post-pruning on a learning sample of size 500 on the CBF-tr problems. It took $123s$ to build a tree of 17 nodes. For comparison, it takes only $14s$ to build a tree of 89 nodes from 128 low-level attributes and $24s$ to build a set of 25 random trees with the same set of attributes (which however is less accurate on this problem). So, the pattern detection based tree method is quite demanding with respect to other approaches but computation times still remain reasonable, especially since they are increasing only in (approximately) order $O(N \log N)$ with respect to the learning sample size.

12.3.3 Interpretability

Since one of the main reasons for the introduction of a new method is interpretability, it is important to verify that the proposed method indeed produces interpretable models. To this end, we have built a decision tree from 500 random scenarios on the CBF-tr and Two-pat problems, by fixing the value of S with post-pruning. Figure 12.9 reproduces the shallow nodes of both trees. Complex patterns are plotted in each box corresponding to an interior node. When these patterns are composed of several elementary patterns, they are drawn in order.

On the CBF problem, the first pattern is a signal at mid-level with respect to the maximum amplitude. Since the cylinder class does not go through this mid-level, scenarios of this class are directed to the right successors. Bell and funnel classes are then separated by a pattern which interpretation is obvious from the knowledge of the bell class: an increasing trend from the mid-level to the maximum level. On the Two-pat problem, the first test is composed of two elementary patterns which only appear in this order in the DU and UU classes (U for upward step and D for downward step). The first elementary pattern which appears in both upward and downward step is used by the method to impose that the second elementary pattern appears in second position. Then at the left successor, DU and UU classes are separated by a less interpretable pattern which nevertheless only appears when there is a downward step in the signal (which may go from -5 to 0 while the upward step always goes from -5 to 5). UD and DD classes are separated by repeating the "trick" of the top level test. The first elementary pattern is an upward step and the second elementary pattern imposes that the first one appears in first position.

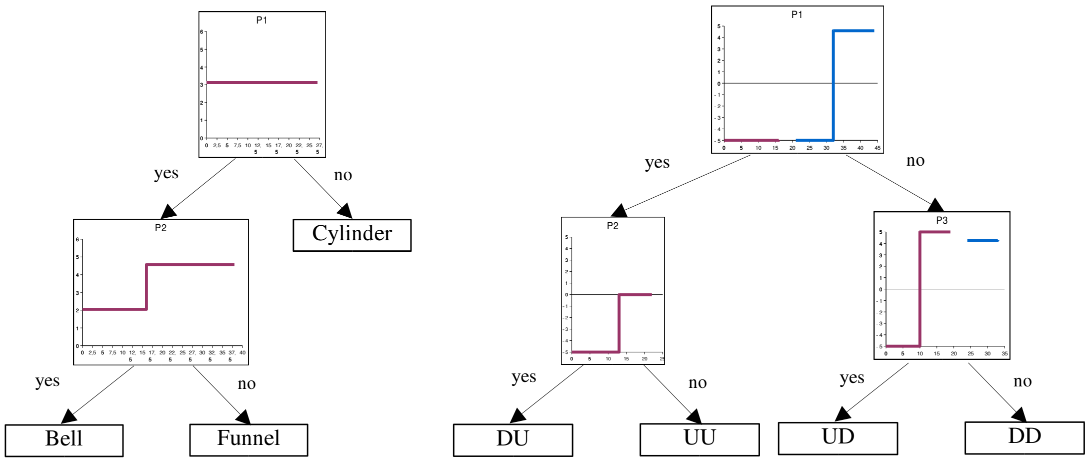

Figure 12.9: Decision trees with pattern extraction, left on CBF-tr, right on Two-pat.

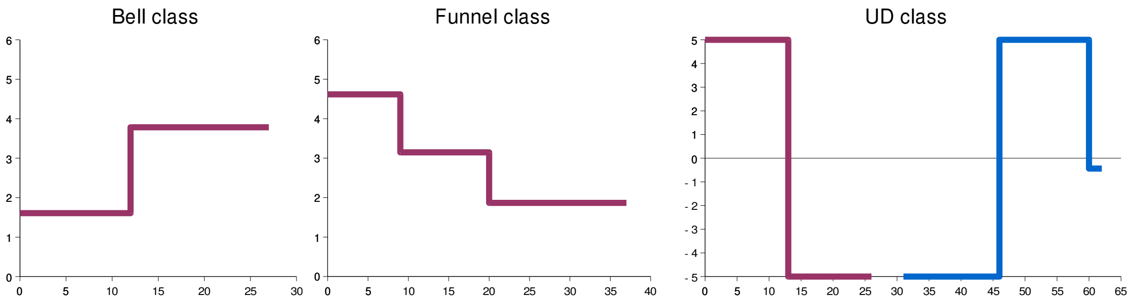

Figure 12.10: Descriptive patterns for the bell and funnel classes on the CBF-tr problem and for the UD class on the Two-pat problem

Although interpretable, the patterns produced by these trees are by construction more discriminative than descriptive. They do not explicitely characterize elements of one class. In some cases, it is useful to give descriptive rules for each class. More descriptive patterns may be extracted by our method if we search for patterns which separate one class from all other ones and if we further impose that patterns are extracted only from scenarios of this class (by selecting only scenarios of this class in the procedure of Table 12.5). For example, Figure 12.10 shows typical patterns of the bell and funnel classes on the CBF-tr problem and of the UD class on the Two-pat problem. These patterns were extracted from a scenario of the given class by using the algorithm of Table 12.5 in trying to separate this class from all the other ones in a learning sample of size 500. This time, patterns are very readable and really characterize scenarios of each class.

To illustrate the interest of such methods in a more practical problem, we carried out one further experiment on a dataset of ECG signals which is discussed in [Ols01]. Each scenario of this dataset records the sequence of measurements obtained by two electrodes during one heartbeat. Each heartbeat is further classified as normal or abnormal. In the latter case, it corresponds to a cardiac pathology known as supra-ventricular premature beat which it is of course important to detect in practice. We run our decision tree algorithm on a sample of 200 cases. At the top node of this tree, a pattern is detected on the second electrode which detects in majority normal heartbeat. This pattern is reproduced in the left part of Figure 12.11 and its occurrence in a normal heartbeat is plotted in the center part of the same figure. It tells us that normal heartbeats are

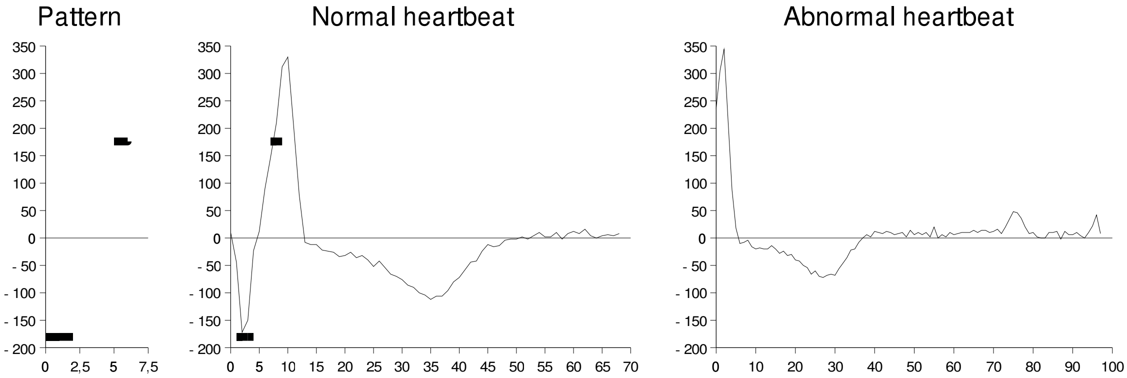

Figure 12.11: From left to right, a pattern characterizing a normal heartbeat, its detection in a normal heartbeat, and, an abnormal heartbeat

characterized by a low value during at least 3 time steps followed by a high value during at least 2 time steps. Such a model gives us the information that abnormal heartbeat are characterized by the fact that they do not present this pattern (Figure 12.11 plots one abnormal heartbeat which indeed does not present this properties). This single pattern yields an error rate of 15% on the learning sample. Although prior knowledge of the problem (which we do not have) can either confirm or infirm that such a pattern is really relevant for classification, we believe that such results are interesting in themselves since they are confirmed by visual inspection of the signals appearing in the learning sample.

12.4 Related work

The work presented in this chapter is essentially a combination and an adaptation of several ideas which have been studied separately by others in the literature. We first give some guide on the work on piecewise modeling for time series data. Then, we review pattern matching algorithms for time series. Eventually, we provide pointers to work which also proposes interpretable models for time series classification.

Piecewise models for time series

General (not only constant) piecewise representations for time series have been introduced in many works for several purposes, e.g. for time series classification in [Man97] or to fasten the search for similar time series in large databases in [KP99]. The more general problem of image segmentation is also tackled in the very well-known textbook by Duda and Hart [DH73]. The most used type of piecewise model is the piecewise linear model. Finding the optimal segmentation for polynomial models in S segments in general requires an order $O(N.S^2)$ dynamic programming algorithm (see for example [Man97] or [Ols01] for a description of the exact algorithm). Several suboptimal algorithms have been devised. A review of these algorithms is given in [KCHP01]. They can be divided into three classes: bottom-up algorithms (which recursively merges consecutive segments), top-down algorithms, and online algorithms (which segments the time series as data arrive). With respect to these approaches, our algorithm is a combination of top-down and bottom-up (at the post-pruning stage).

Also, a main difference of our approach is the criterion proposed to decide on the number of segments to use, which is essentially guided in our case by the bias/variance tradeoff, whereas, in the related literature, the criterion used is generally a “fidelity” criterion, similar to those used in lossy data compression. Indeed, usually, the selection of the optimal number of segments simply relies on a predefined threshold on error. In the top-down approach, this amounts at stopping the development of a node if the local error in this segment is lower than a given threshold. Possible criteria use the MDL principle [Man97] or statistical tests [LSL+00]. A more original approach is adopted in [Ols01] where the optimal number of segments is determined by the discontinuity point in the approximation of the error curve (the resubstitution error in Figure 12.5) by a piecewise linear model with two segments. Pruning for piecewise modeling was also proposed in [SÚW98], however in the more general context of scatter-plot smoothing by linear hinges models.

Pattern matching and extraction in time series

In our algorithm, patterns are detected in the time series by using the euclidian distance and an exact algorithm based on dynamic programming ideas. The resulting pattern definition is certainly not invariant with respect to several interesting transformation in the temporal domain (such as time scaling) but it was chosen so as to make learning quite efficient. Nevertheless, several works exist on pattern definition and pattern matching in time series. The main motivations of such works are generally the design of a distance measure which is invariant to some transformations in the temporal domain and/or the efficiency of the corresponding pattern detection algorithm for searching in large databases. With these goals, distance measures have been proposed which are based on Fourier transformation [AFS93], dynamic time warping [BC96], or probabilistic models [KS97, GS00]. In all these works, it is however assumed that the patterns are previously defined by the user and to our knowledge, none of these works have been coupled with a pattern extraction algorithm for time series classification purpose.

The problem of pattern extraction in time series is tackled for example in [DLM+98] and [Oat99]. Patterns are defined as subsequences of a fixed number of time points and relevant patterns are found by unsupervised clustering techniques. In [DLM+98], these patterns are used to find association rules (i.e. conditional rules of the form "pattern A is always followed by pattern B"). In [Oat99], the goal is to find distinctive patterns, i.e. patterns which appear frequently in the scenarios of one class but not in scenarios which do not belong to this class. Although the goal of the author is not directly time series classification, it is clear that his technique may be used in this purpose.

Interpretable models for time series classification

Several machine learning approaches have been developed recently to solve the time series classification problem with a focus on interpretable models. Manganaris [Man97], for example, constructs piecewise polynomial models for univariate time series and directly uses the parameters of this representation as attributes for classification with decision trees. Olszewski [Ols01] uses a very similar approach but with composite piecewise models (with an automatic choice among linear, quadratic, and exponential models in each segment). He also feeds decision trees with the parameters of these models. Although they both use piecewise models for feature extraction, these approaches are very different from our own approach. The piecewise model is used in their work to provide a small set of attributes while it is used to support pattern extraction in our method.

At least three works are based also on pattern extraction. Kadous [Kad99] defines "parame-trized event primitives" which are extracted from the time series of the learning sample. The number of such event primitives is reduced by using clustering techniques in their parameter spaces and then characteristic functions of the resulting clusters are used with standard automatic learning algorithms (Naive Bayes or decision trees). Kudo et al. [KTS99] transforms multivariate signals into binary vectors. Each element of these vectors corresponds to one rectangular region of the "value-time" plane and tells if the signal passes through this region. The number of regions is reduced by a discretization of time and value. A method of their own, subclass, builds rules from these binary vectors. Gonzales et al. [AR00] extends (boosted) inductive logic programming systems with predicates that are suited for the task of time series classification. The number of possible predicates considered during induction is reduced by discretization of their possible defining parameters.

These three approaches share some common characteristics with our decision tree extension. First, authors are all interested in getting interpretable rules and so use an interpretable classifier. Second, they use some discretization techniques to reduce the search space for patterns (by discretization of the parameter space in [KTS99] and [AR00] and unsupervised clustering in [Kad99]). In the first two methods, patterns are extracted globally before the construction of the classifier while, in the last one, the search for patterns is incorporated in the induction algorithm like in our work.

The main difference with our approach is that primitive patterns are defined by some parametric models which are chosen so as to be useful in most application domains. In our method, patterns are extracted from the time series of the learning sample and hence they do not have to satisfy any predefined form. Another difference is that patterns in these works are defined on absolute time values (even if some variations about their start time is allowed by the pattern definitions) while our pattern definition explicitly captures shift invariant properties. This may be an advantage over these methods in problems which really present such patterns but also a disadvantage when shift invariant patterns are not relevant for classification.

12.5 Conclusions and future research directions

In this chapter, we have presented a new tool to handle time series in classification problems which produces interpretable results. 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. With respect to the use of low-level features, the advantage of this technique in terms of interpretability is undeniable. In terms of accuracy, better results can be obtained by using either random trees or the segment and combine approach. However, in problems where we know that classes are actually characterized by local shift invariant properties (like in CBF-tr or Two-pat), the results given by this algorithm are competitive with these latter methods.

From this preliminary study, two future work directions seem particularly interesting: extensions of the model to handle other temporal peculiarities and combination of this techniques with perturb and combine algorithms.

Extensions of the model

The presented method is only a first attempt to produce interpretable models for time series classification problems and it suffers from several shortcomings essentially in terms of representation capabilities.

First, the representation of time series with piecewise constant models while computationally efficient is somewhat crude and often needs a lot of segments to reach a high precision. It could be advantageously exchanged for a more powerful decomposition, for example using piecewise linear models. Although the computational cost of extracting such piecewise linear models from the attribute values of an object would be higher, we believe that both the interpretability of the resulting rules and the computational complexity of the resulting automatic learning method could be significantly improved in this way. Indeed, the optimum number of segments with a piecewise linear model should be lower than the same number with a piecewise constant model and hence this should reduce the number of candidate patterns to consider during the development of decision tree nodes.

Second, our model was tailored for applications where classes are characterized by local shift invariant properties but it does not allow the detection of shrunk or extended versions of the reference pattern along the time or value axis. This limitation comes from the use of euclidian distance which is sensitive to change in time scale and furthermore can only compare portions of signals of the same size. At the moment, the only way to take into account such scale invariant properties within our method is to introduce several patterns corresponding to this property at different time scales.

One way to circumvent the problem is to detect our patterns by using another distance measure which would be invariant or at least less sensible to scale changes in patterns. One possible distance measure is dynamic time warping (see for example [BC96]) which finds the point-to-point alignment between two time series (possibly of different lengths) which minimizes the distance between them. Another possibility is to use probabilistic pattern matching like in [GS00] or [KS97]. More flexibility could be for example added to our system by putting a probability distribution on the lengths and amplitudes of segments which would naturally take into account time and amplitude deformations of the pattern. Actually, this transforms our piecewise constant model into a particular case of a segmental hidden Markov model which was studied in the literature (see [ODK96] or [AMGD00] for applications). In this approach, a pattern would be viewed as a probabilistic model generating time series and the pattern matching based on euclidian distance would be replaced by the computation of the (log-)likelihood of the signal given that model. We could then take advantage of the existing learning algorithms for probabilistic models to learn the pattern parameters from the whole set of time series of one class (instead of only one randomly drawn from the learning sample).

When considering extensions of the algorithm, great care should be taken for not increasing too much the variance. If we know from prior knowledge that such extensions are not necessary for classification, then they should be avoided.

Perturb and combine algorithm

With decision trees and pattern extraction, the bias is low but the variance is high (because of the increase of the size of the hypothesis space). This suggests that perturb and combine algorithms would give good results in this context. As it selects scenarios at random in the learning sample, our algorithm is actually random. The simplest perturb and combine variant would thus consist in running the algorithm several times on the same data and combine the predictions of the resulting models to reduce variance. However, the resulting method will not be computationally efficient since the search for one model is already quite demanding.

A better approach would either consist in extending the dual perturb and combine algorithm to the present method or to design a random tree growing algorithm in this context, specifically so as to reduce computation times. For example, the bit-vector masks that define complex patterns might be selected at random to develop tree nodes and hence result in a fast random tree growing algorithm. Nevertheless since this approach would result in non interpretable classification models, it should be compared with variance reduction algorithms which use low-level attributes. If both give the same error, then the approach which is the most efficient (for learning or for testing), should be preferred.