Learning First Order Logic Time Series Classifiers *

Juan J. Rodríguez[1], Carlos J. Alonso[1], and Henrik Boström[2]

[1] Departamento de Informática
Grupo de Sistemas Inteligentes
Universidad de Valladolid, Spain
{juanjo,calonso}@infor.uva.es
[2] Department of Computer and System Sciences
Stockholm University/KTH, Sweden
henke@dsv.su.se

Abstract. A method for learning multivariate time series classifiers by inductive logic programming is presented. Two types of background predicate that are suited for this task are introduced: interval based predicates, such as "always", and distance based, such as the euclidean distance. Special purpose techniques are presented that allow these predicates to be handled efficiently when performing top-down induction. Furthermore, by employing boosting, the accuracy of the resulting classifiers can be improved significantly. Experiments on several different datasets show that the proposed method is highly competitive with previous approaches. On all data sets, the proposed method achieves better than all previously reported results. Moreover, the comprehensibility of the induced classifiers is found to be better than for classifiers produced by previous approaches.

1 Introduction

Multivariate time series classification is useful in domains such as biomedical signals [18], continuous systems diagnosis [2] and data mining in temporal databases [6]. This problem can be tackled by extracting features of the series through some kind of preprocessing, and using some conventional machine learning method. However, this approach has several drawbacks [16]: the preprocessing techniques are usually *ad hoc* and domain specific, there are several heuristics applicable to temporal domains that are difficult to capture by a preprocess and the descriptions obtained using these features can be hard to understand. The design of specific machine learning methods for the induction of time series classifiers allows the construction of more comprehensible classifiers in a more efficient way.

When learning multivariate time series classifiers, the input consists of a set of training examples and associated class labels, where each example consists of one or more time series. The series are attributes of the examples, and are

* This work has been supported by the CYCIT project TAP 99–0344.

often referred to as variables, since they vary over time. Nevertheless, in a logic program their names will be represented by constants. The objective is to find a classifier with a low error rate and that is comprehensible.

The method for learning time series classifiers that we propose in this work is based on *inductive logic programming* (ILP) and utilises two types of background predicate: i) interval based predicates [25], such as always(Example, Variable, Region, Beginning, End) , where Region is an interval in the domain of the variable, and ii) distance based predicates [24], such as euclidean_le(Example, Reference, Variable, Value), which considers the euclidean distance between two series.

The proposed system is neither a generic ILP system, nor is it an extension of one of the ILP systems available. In our experience, it is not enough to simply give predicates of the above types as background knowledge to an existing system, but it is also necessary to incorporate knowledge about how to process them efficiently. Hence, specific methods has been developed that allow for an efficient search for hypotheses that include these types of literals.

Moreover, we have also incorporated boosting, which is a method for generating ensembles of classifiers that has been demonstrated to improve accuracy significantly. In our case, the individual classifiers are composed only of one literal in the body of a rule.

When tested on several previously proposed datasets, the new method achieves better results than all previously published results on these datasets. Furthermore, the comprehensibility of the classifiers produced by the new method is in our opinion better than the comprehensibility of the classifiers produced by previous methods.

The rest of the paper is organized as follows. Section 2 describes the special purpose background predicates for time series classifiers. The proposed method is presented in section 3, including techniques for efficiently handling the special purpose predicates. Section 4 presents experimental results when using the new method. In section 5, it is demonstrated that these results can be improved substantially by employing boosting. Finally, we give some concluding remarks in section 6.

2 Temporal Predicates

2.1 Interval Predicates

The interval predicates are based on the ones used in a visual rule language for dynamic systems [2] and were introduced in [25]. They make use of *regions*, which are intervals in the domain of the variable. The regions used are independent of the time, as is usual practice when working with series.

The interval predicates are the following:

– always(Example, Variable, Region, Beginning, End). It is true, for the Example, if the Variable is always in this Region in the interval between Beginning and End.

– sometime(Example, Variable, Region, Beginning, End). It is true, for the Example, if the Variable is sometime in this Region in the interval between Beginning and End.
– true-percentage(Example, Variable, Region, Beginning, End, Percentage). It is true, for the Example, if the percentage of the time between Beginning and End where the variable is in Region is greater or equal to Percentage.

Once that it has been decided to work with temporal intervals, the use of the predicates always and sometime seems natural, due to the fact that they are extensions of conjunction and disjunction to intervals. Since the former predicate might be too demanding and latter too flexible, a third one has been introduced, true-percentage. It is a "relaxed always" (or a "restricted sometime"). The additional parameter indicates the degree of flexibility (or restriction).

These predicates can also capture the concepts of "increasing" and "stable", which appear as predicates in [17], by including new auxiliar series as background knowledge, which are obtained from the original ones. In these auxiliar series the value in an instant is defined as the difference between the value in the original series for that instant and the value of a previous, e.g. one minute, instant [25]. In this way it is not only possible to allow for detection of increments but also to detect if the increments are big or small and with the use of the predicate true-percentage it is also possible to capture concepts like "generally increasing".

2.2 Distance Predicates

Several machine learning methods, such as instance-based learning, are based on the use of similarity functions to measure distances between examples. The framework of ILP allows us in a very flexible way to include several such definitions in the background knowledge. In our case, we use predicates on the following form:

<*distance*>_le(Example, Reference, Variable, Value)

which is true if the <*distance*>, for one Variable of the examples, between the Example considered and another Reference example is less or equal (_le) than Value.

The predicate euclidean_le uses the *euclidean* distance. It is defined, for two univariate series s and t as: $\sqrt{\sum_{i=1}^{n}(s_i - t_i)^2}$. Its execution time is $O(n)$.

Dynamic Time Warping (DTW) aligns a time series to another reference series in a way such that a distance function is minimized, using a dynamic programming algorithm [6]. This technique is frequently used in speech recognition. If the two series have n points, the execution time is $O(n^2)$. The predicate dtw_le uses the minimized value obtained from the DTW as a similarity function between the two series.

3 Top-Down Induction of Time Series Classifiers

The proposed technique follows the scheme of the top-down methods in ILP [5]. The particularities arise in the selection of literals. The use of interval literals require the definition of the different regions, which is described in section 3.1. The selection of interval and distance literals are described in section 3.2 and 3.3 respectively.

A generic method for numerical reasoning in ILP is described in [29]. It could be used to incorporate these selection mechanisms in ILP systems.

3.1 Obtaining Regions

In some cases, the definitions of the regions that are necessary for the interval literals can be obtained from an expert. Otherwise, they can be obtained with a discretization preprocess, which obtains r disjoint, consecutive intervals. The regions considered are these r intervals (equality tests) and others formed by the union of the intervals $1 \dots i$ (less or equal tests), as suggested by [8].

The reasons for fixing the regions before the induction process starts, instead of obtaining them during the process, are efficiency and comprehensibility. The literals are easier to understand if the regions are few, fixed and disjoint.

3.2 Selection of Interval Literals

Given an overly general clause that covers both positive and negative examples, the best literal to add to the body must be selected, according to some criterion. Then it is necessary to search over the space of literals. The possible number of intervals, if each series has n points, is $(n^2-n)/2$. If p is the number of predicates considered, and v the total number of regions in the different variables, the possible number of atoms is $pv(n^2 - n)/2$, and the possible number of literals (atoms possibly negated) is $pv(n^2 - n)$. In the case of predicates with additional arguments (true-percentage), it is also necessary to consider how many values are possible for them.

Linear Probing. The process starts with windows of size 1 (between two consecutive points) and the windows of size $i + 1$ are evaluated from the windows of size i with the same origin. The initialization of the windows of size 1, for each predicate, it is done in $O(e)$, where e is the total number of examples. The amplification of the size of the window in one unit is also done in $O(e)$. It is necessary to calculate the number of positive and negative examples covered. In the most complex case, true-percentage, this requires a time of $O(e + f)$, where f is the number of values allowed in the additional argument of this predicate. Summarizing, the worst execution time for finding the best literal is $O((e + f)pvn^2)$. If $e > f$ (the most common case) or true-percentage is not used, then it is $O(epvn^2)$.

Exponential Probing. With the objective of reducing the search space, not all the windows are explored. Only those that are of size power of 2 are considered. The number of these windows is of $\sum_{i=1}^{k}(n - 2^{i-1}) = kn - 2^k - 1$ where $k = \lfloor \lg n \rfloor$. Using a dynamic programming algorithm it is possible to obtain the information necessary of the window of size $2i$ from two consecutive windows of size i, with a time of $O(e)$ (in the case of `true-percentage`, $O(e + f)$). The selection of the best literal in this case requires a time of $O(epvn \lg n)$.

3.3 Selection of Distance Literals

Calculating the distances between all the examples requires computing $O(e^2)$ distances, which might be too costly. Furthermore, considering all the examples as reference for selecting a literal can also be too slow. Instead, in each iteration, r reference examples are randomly selected from the non-covered examples (it is possible to use only positive reference examples or positive and negative). Since the same example can be selected in several iterations, the calculated distances are saved.

3.4 Multiclass Problems

When there are more than two classes, it is necessary to learn a theory for each class. The question is how to apply these theories to a new example [30]. We have employed two approaches for dealing with multiclass problems. The first one is the use of ordered rules, also named decision lists [23, 19]. In this case, the first rule that covers the example assigns its label to it. The learning process consists of generating a rule for the class with most uncovered examples and iterate until all the examples are covered. Figure 1 shows an example of such a decision list.

The second approach, proposed by [30] and based on [10] is to learn different theories independently for each class, apply all the rules to the new example and if there is a conflict, solve it by considering the distribution of training examples covered by the rules.

4 Experimental Validation

4.1 Datasets Description

The characteristics of the datasets are summarized in table 1. Datasets for classification of time series are not easy to find [16]. For this reason we have used four artificial datasets and only one "real world" dataset.

Cylinder, Bell and Funnel (CBF). This is an artificial problem, introduced in [26]. The learning task is to distinguish between three classes: cylinder (c), bell (b) or funnel (f). Examples are generated using the following functions:

```
class( Example, cylinder ) :- % 213, 426
   not true-percentage( Example, x, 1_4, 22, 86, 50 ), % 193, 18
   not true-percentage( Example, x, 3, 78, 110, 30 ), % 182, 0
   !.
class( Example, bell ) :- % 213, 244
   true-percentage( Example, x, 1_4, 3, 35, 95 ), % 213, 1
   not always( Example, x, 1_4, 39, 103 ), % 213, 0
   !.
class( Example, funnel ) :- % 213, 31
   dtw_le( Example, f_35, x, 1.241447 ), % 200, 1
   not euclid_le( Example, c_162, x, 8.629407 ), % 200, 0
   !.
class( Example, cylinder ) :- % 31, 13
   not true-percentage( Example, x, 3, 9, 73, 15 ), % 31, 0
   !.
class( Example, funnel ). % 13, 0
```

Fig. 1. Rules example. The rules are ordered, organized in a decision list. They were obtained from the dataset *CBF* (Sect. 4.1). At the left of each literal, the number of positive and negative examples covered by the (partial) rule.

Dataset	Classes	Variables	Examples	Points
CBF	3	1	798	128
Control charts	6	1	600	60
Waveform	3	1	900	21
Wave + noise	3	1	900	40
Auslan	10	8	200	*20

Table 1. Characteristics of the datasets. The number of points in the original *Auslan* dataset was variable, the examples were resampled to 20 points.

$$c(t) = (6+\eta)\cdot\chi_{[a,b]}(t) + \epsilon(t)$$
$$b(t) = (6+\eta)\cdot\chi_{[a,b]}(t)\cdot(t-a)/(b-a) + \epsilon(t) \qquad \chi_{[a,b]} = \begin{cases} 0 \text{ if } t < a \vee t > b \\ 1 \text{ if } a \leq t \leq b \end{cases}$$
$$f(t) = (6+\eta)\cdot\chi_{[a,b]}(t)\cdot(b-t)/(b-a) + \epsilon(t)$$

η and $\epsilon(t)$ are obtained from a standard normal distribution $N(0,1)$, a is an integer obtained from a uniform distribution in $[16,32]$ and $b-a$ is another integer obtained from another uniform distribution in $[32,96]$. The examples are generated evaluating those functions for $t = 1,2\ldots128$. Figure 2.a shows two examples of each class. For ease of comparison with previous results, 266 examples of each class were generated.

Control Charts. In this dataset there are six different classes of control charts, synthetically generated by the process in [1]. Each time series is of length n, and is defined by $y(t)$, with $1 \leq t \leq n$:

1. Normal: $y(t) = m + s\,\mathrm{r}(t)$. Where $m = 30$, $s = 2$ and $\mathrm{r}(t)$ is a random number in $[-3,3]$.
2. Cyclic: $y(t) = m + s\,\mathrm{r}(t) + a\sin(2\pi t/T)$. a and T are in $[10,15]$.
3. Increasing: $y(t) = m + s\,\mathrm{r}(t) + gt$. g is in $[0.2, 0.5]$.
4. Decreasing: $y(t) = m + s\,\mathrm{r}(t) - gt$.
5. Upward: $y(t) = m + s\,\mathrm{r}(t) + x\,\mathrm{k}(t)$. x is in $[7.5, 20]$ and $\mathrm{k}(t) = 0$ before time t_0 and 1 after this time. t_0 is in $[n/3, 2n/3]$.
6. Downward: $y(t) = m + s\,\mathrm{r}(t) - x\,\mathrm{k}(t)$.

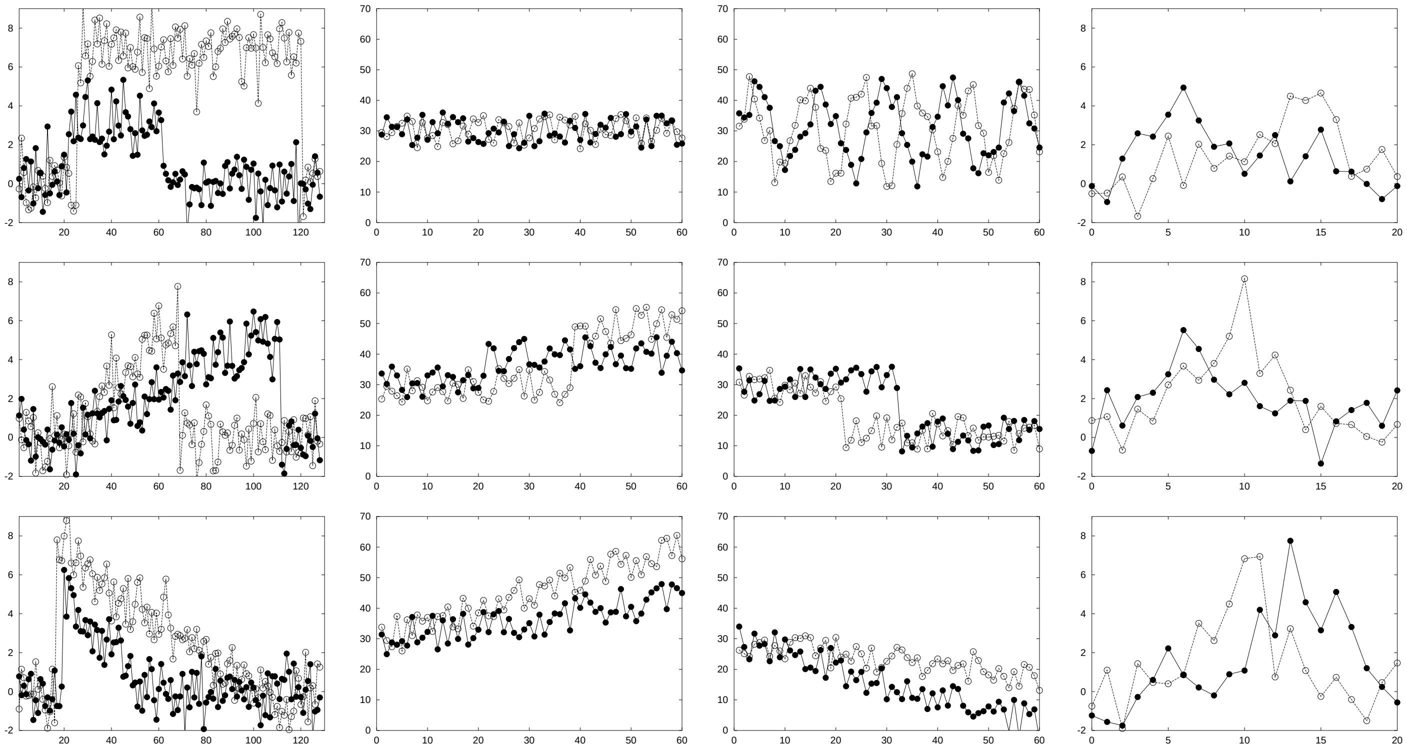
Figure 2.b shows two examples of each class. The data used were obtained from the UCI KDD Archive [4]. It contains 100 examples of each class, with 60 points in each example.

Waveform. This dataset was introduced by [9]. The purpouse is to distinguish between three classes, defined by the evaluation for $i = 1,2\ldots21$, of the following functions:

$$x_1(i) = uh_1(i) + (1-u)h_2(i) + \epsilon(i)$$
$$x_2(i) = uh_1(i) + (1-u)h_3(i) + \epsilon(i)$$
$$x_3(i) = uh_2(i) + (1-u)h_3(i) + \epsilon(i)$$

where $h_1(i) = \max(6 - |i-7|, 0)$, $h_2(i) = h_1(i-8)$, $h_3(i) = h_1(i-4)$, u is a uniform aleatory variable in $(0,1)$ and $\epsilon(t)$ follows a standard normal distribution. Figure 2.c shows two examples of each class.

We use the version from the UCI ML Repository [7]. In the experiments the first 300 examples of each class were selected. It is a normal practice to work only with 300 examples in total for this dataset [21, 31].



(a) CBF (b) Control Charts (c) Wave

Fig. 2. Some examples of the datasets. Two examples of the same class are shown in each graph.

Wave + Noise. This dataset is generated in the same way than the previous one, but 19 points are added at the end of each example, with mean 0 and variance 1. Again, we used the first 300 examples of each class of the corresponding dataset from the UCI ML Repository.

Auslan. Auslan in the Australian sign language, the language of the Australian Deaf community. Instances of the signs were collected using an instrumented glove [16]. Each example is composed by 8 series: x, y and z position, wrist roll, thumb, fore, middle and ring finger bend. The number of points in each example is variable, so they were resampled to 20 points.

4.2 Results

The results for each dataset and setting were obtained using five five-fold stratified cross-validation. For interval predicates, exponential probing was used. The percentages considered for the predicate true-percentage were 5, 15, 30, 50, 70, 85 and 95. For similarity literals, at most, 10 positive and 10 negative reference examples were considered when selecting literals.

Results are shown in Table 2. For each experiment, it includes the mean error in percentage and its standard deviation. The standard deviation is calculated from the 25 values obtained from the $5{\times}5$ cross-validation process.

The first remarkable point is that each predicate is the best for some dataset. Euclidean is the best for the *Wave* and *Wave + Noise* using decision lists or unordered rules, and for the *Auslan* dataset with decision lists. The performance of the euclidean distance is a bit surprising considering its simplicity. Nevertheless its results are the worst for *CBF* and *Control Charts*. In these cases DTW works much better. This situation is reasonable, because the *CBF* and *Control* datasets are designed specifically to test time series classifers, and hence involve situations like shifts, compresions and expansions. These situations are not present in the *Wave* datasets, and hence euclidean works better than DTW.

The use of always and sometimes gives the best results in the *Control* dataset. In the case of decision lists, the first position is shared with the use of the predicate true-percentage. The advantages of using it over using always and sometimes are only clear for the *CBF* dataset. This is probably due to the fact that the different situations that characterize the examples have a beginning and end for the *CBF* dataset but for the *Control* dataset there are no returns to the normal situation.

The results obtained using all the predicates are the best for the *Auslan* dataset with unordered rules and for the *CBF* dataset. In the rest of the cases, the results are close to the best.

With respect to the use of decision lists or unordered rules, there is no clear winner regarding the error, although for the *Auslan* dataset decision lists work much better than unordered rules. Probably the inclusion of pruning methods could alter this situation.

Predicates		Decision Lists			Unordered Rules		
		Error	Clauses	Literals	Error	Clauses	Literals
CBF	EUC	6.67 2.39	16.16 1.93	34.76 3.88	6.87 2.16	26.48 1.74	61.04 4.22
	DTW	2.16 1.22	9.00 1.41	12.69 1.60	2.11 1.02	11.80 1.04	19.44 1.90
	AST	4.51 1.38	9.72 0.89	15.16 1.82	4.31 2.03	13.52 0.87	33.08 2.12
	TRP	2.76 1.25	7.44 0.88	10.00 1.23	2.88 1.28	9.64 0.86	22.28 2.19
	ALL	**1.65** 1.10	5.92 0.91	7.12 0.97	**1.10** 0.89	7.46 0.65	12.68 1.22
Control	EUC	4.63 1.38	11.92 1.19	18.20 2.00	6.03 2.20	16.72 2.07	30.84 3.53
	DTW	3.23 1.71	10.76 0.93	13.52 1.45	4.10 1.89	13.52 1.59	20.28 1.95
	AST	**3.07** 1.64	10.88 1.01	15.44 1.45	**2.97** 1.23	11.96 0.79	23.84 1.25
	TRP	**3.07** 1.66	9.76 1.33	14.04 1.97	4.13 1.98	12.44 1.26	24.44 1.83
	ALL	3.20 1.55	8.80 1.33	11.04 1.49	3.60 1.64	11.40 1.26	17.36 1.44
Wave	EUC	**19.64** 2.82	26.28 1.65	83.48 5.69	**19.47** 2.73	49.24 2.52	177.68 9.88
	DTW	24.38 3.65	30.48 2.14	101.64 4.80	25.49 3.75	55.80 1.96	209.40 7.33
	AST	22.29 2.85	31.24 2.97	105.88 7.60	23.04 3.18	51.52 1.85	217.20 9.61
	TRP	21.44 2.96	28.12 1.36	92.56 4.69	22.64 3.09	45.68 2.17	188.44 7.08
	ALL	19.95 2.72	23.44 1.76	69.76 6.08	20.65 3.10	38.60 1.63	141.16 6.14
Wave+Noise	EUC	**21.07** 3.04	27.72 2.21	92.72 7.27	**21.64** 2.16	51.12 2.21	188.44 7.39
	DTW	26.78 2.78	32.76 1.86	109.48 4.00	28.84 2.51	60.60 2.58	226.08 7.27
	AST	24.80 3.35	27.48 1.42	90.24 3.73	24.67 2.53	44.64 1.08	183.56 6.18
	TRP	24.62 2.44	25.64 1.52	89.96 4.36	25.58 3.39	41.64 1.19	164.92 5.74
	ALL	22.31 3.17	22.00 1.19	66.68 3.45	22.91 2.54	36.88 1.92	134.64 6.47
Auslan	EUC	**15.00** 4.21	12.76 1.23	16.52 1.69	24.60 7.49	18.04 1.49	31.40 2.10
	DTW	21.60 6.33	14.40 1.29	20.44 2.22	24.20 5.94	18.68 1.07	30.64 1.58
	AST	21.40 7.00	15.16 1.31	23.56 1.78	22.80 6.55	18.88 1.30	38.64 2.22
	TRP	20.10 4.97	14.92 0.76	22.64 1.44	23.80 6.13	19.04 1.46	37.76 2.44
	ALL	16.60 5.49	12.08 0.64	14.64 0.91	**21.50** 7.74	15.64 1.50	25.44 1.76

Table 2. Experimental Results. The predicates considered are EUClidean, DTW, Always / SomeTime, TRue Percentage and ALL together. For each combination, the averaged error (in %) and its standard deviation of the 25 executions are shown. In boldface, the best results. Also, the number of clauses and the total number of literals (excluding the heads) are shown.

5 Improving Accuracy by Boosting

At present, an active research topic is the use of *ensembles* of classifiers. They are obtained by generating and combining base classifiers, constructed using other machine learning methods, typically decision tree learners. The target of these ensembles is to increase the accuracy with respect to the base classifiers.

One of the most popular methods for creating ensembles is boosting [28], a family of methods, of which AdaBoost is the most prominent member. They work by assigning a weight to each example. Initially, all the examples have the same weight. In each iteration a base classifier is constructed, according to the distribution of weights. Afterwards, the weights are readjusted according to the result of the example in the base classifier. The final result is obtained by combining the weighted votes of the base classifiers. Boosting inductive logic programming is described in [22].

Inspired by the good results of works using ensembles of very simple classifiers [14], sometimes named *stumps*, we have studied base classifiers consisting only of one literal. The reasons for using so simple base classifiers are:

- Ease of implementation. In fact, it is simpler to implement a boosting algorithm than a decision tree or rule inducer, especially if pruning methods are included.
- Comprehensibility. It is easier to understand a sequence of weighted literals than a sequence of weighted decision trees or rules.

The criterion used for selecting the best literal is to select the one with the smallest error, relative to the weights.

5.1 Multiclass Problems

There are several methods of extending AdaBoost to the multiclass case [28]. We have used AdaBoost.OC [27] since it can be used with any weak learner which can handle binary labeled data. It does not require that the weak learner can handle multilabeled data with high accuracy. AdaBoost.OC incorporates the method of error-correcting output codes [13], for handling multiclass learning problems, into the boosting algorithm.

The key idea is, in each round of the boosting algorithm, to select a subset of the set of labels, and train the binary weak learner with the examples labeled positive or negative depending if the original label of the example is or is not in the subset. In our concrete case, the rule inducer searches for a rule with the head:

$$\mathsf{class(\ Example,\ [class_1,\ \ldots class_k]\ )}$$

This predicate means that the Example is of one of the classes in the list[1]. Figure 3 shows an ensemble of these classifiers.

[1] If we are not restricted to Horn clauses, we can say that the the clause is of the form $\mathsf{class(\ Example,\ class_1),\ \ldots,\ class(\ Example,\ class_k)\ :\text{-}\ \ldots.}$

```
class( Example, [ decreasing, downward, increasing ] ) :-    % 240, 240
  not true-percentage( Example, x, 5, 43, 59, 5 ).    % 197, 26
  % 0.456818
class( Example, [ decreasing, upward, normal ] ) :-    % 240, 240
  true-percentage( Example, x, 1_4, 14, 22, 70 ).    % 229, 84
  % 0.422232
class( Example, [ cyclic, decreasing, downward ] ) :-    % 240, 240
  sometime( Example, x, 2, 22, 54 ).    % 240, 0
  % 0.700627
class( Example, [ cyclic, decreasing, normal ] ) :-    % 240, 240
  dtw_le( Example, cyclic_54, x, 59.898940 ).    % 159, 0
  % 0.536895
class( Example, [ decreasing, upward, downward ] ) :-    % 240, 240
  not true-percentage( Example, x, 5, 37, 53, 5 ).    % 179, 23
  % 0.390684
...
```

Fig. 3. Initial fragment of an ensemble of classifiers, obtained with AdaBoost.OC, for the dataset *control charts* (Sect. 4.1). The weights of each individual classifier are below each classifier.

The classification of a new example is obtained from a weighted vote of the results of the weak classifiers. For each rule, if its antecedent is true, then the weights of all the labels in the list are incremented by the weight of the rule. Finally, the label that has been given the highest weight is assigned to the example.

5.2 Results

The results obtained with the boosting process are summarized in Table 3. For all the datasets, the benefits of using boosting are substantial. Moreover, the evolution of the error with the number of iterations is rather good. Until the limit of the number of iterations considered, the increment of the number of iterations do not cause more overfitting.

Cylinder, bell and funnel. The best previously published result, according to our knowledge, with this dataset is an error of 1.9 [16], using 10 fold cross validation. From iteration no. 10, the results shown in table 3 are better than this result, and from iteration no. 20 the results are smaller than 1. Moreover, the results using rules without boosting (Table 2) are also better than 1.9 (1.65 and 1.10).

Control charts. The only previous result we are aware of regarding this dataset is for similarity queries [1], and not for supervised classification. To check if this dataset was trivial, we tested it with C4.5 [20], over the raw data, and obtained an average error of 8.6 (also using five five-fold cross validation).

Iterations	5	10	20	30	40	50
CBF	2.41 1.21	1.50 0.85	0.75 0.60	0.60 0.53	0.58 0.65	**0.50** 0.60
Control	22.30 2.58	8.70 5.38	2.27 1.62	1.80 1.22	1.60 1.25	**1.47** 1.19
Wave	19.58 2.55	18.36 2.79	15.64 2.46	15.49 2.08	**15.00** 2.32	15.04 2.36
Wave + Noise	23.87 3.13	20.56 3.24	18.29 2.73	17.60 2.41	17.18 2.36	**16.78** 2.11
Auslan	61.50 7.14	37.90 7.90	16.70 5.72	11.50 4.39	8.40 3.88	7.60 3.27
+100 iter.		3.60 2.51	3.80 2.71	3.20 2.34	3.10 2.20	2.90 1.87
+150 iter.		3.00 1.77	3.10 2.20	2.50 1.91	2.80 2.20	2.40 2.22
+200 iter.		2.40 2.22	2.30 2.16	**1.90** 2.20	2.20 1.95	2.00 1.77

Table 3. Results obtained with boosting, for several numbers of iterations, using all the background predicates. For the *Auslan* dataset, the table includes results with more iterations (110, 120 ... 150; 160 ... 200 and 210 ... 250).

Waveform. The error of a Bayes optimal classifier on this dataset, obtained analytically from the functions that generate the examples, is approximately 14 [9]. This dataset is frequently used for testing classifiers. It has also been tested with boosting (and other methods of combining classifiers), over the raw data, in different works [21, 15, 31, 12]. The best previous result we are aware of for this dataset is an error of 15.21 [12]. That result was obtained using decision trees as base classifiers, which are much more complex than our base classifiers (clauses with one literal in the body).

Wave + Noise. Again, the error of an optimal Bayes classifier on this dataset is 14. This dataset was tested with bagging, boosting and variants over MC4 (similar to C4.5) [3], using 1000 examples for training and 4000 for testing and 25 iterations. Although their results are given in graphs, their best error is apparently approximately 17.5.

Auslan. This is the dataset with the highest number of classes (10). In order to distinguish between this large number of classes, it turned out that 50 one-body-literal clauses were too few, so we incremented the number of iterations for this dataset, until 250. The best result previously published is an error of 2.50 [16].

6 Conclusions and Future Work

A multivariate time series classification system has been developed, using ILP techniques. Two types of background predicate were considered: those based on intervals and regions and those based on similarity functions. The use of similarity literals allows a smooth integration of Instance Based Learning and ILP, leading to that similarity functions can be defined by the user and incorporated as background knowledge and that these functions appear explicitly in the rules instead of being an external element to the learned theory. Special purpose techniques have been presented that allow these predicates to be handled efficiently when performing top-down induction. Furthermore, we have demonstrated that by employing boosting, the accuracy of the resulting time series classifiers can be improved significantly. Experiments on several different datasets show that the proposed method is highly competitive with previous approaches. On all data sets, the proposed method achieves better than all previously reported results. Moreover, the comprehensibility of the induced classifiers is found to be better than for classifiers produced by previous approaches.

Two methods have been used for dealing with multiclass problems: decision lists and unordered rules with associated distribution of training examples. It would also be possible to use the method of error-correcting output codes, used in AdaBoost.OC, for dealing with multiclass problems, when learning rules. We have not explored this possibility because we believe that the obtained results would be less comprehensible. When using boosting, it is also possible to learn a classifier for each class independently, as done in [24], but in this case the resulting classifiers may be much bigger, since it is necessary to generate an ensemble for each class, instead of one ensemble for all the classes.

Boosting has been used over very simple individual classifiers: clauses with a body consisting of one literal only and a head that assigns the example to one of several classes. This situation is the opposite to the one that appears when learning rules: there are typically several literals in the body, but only one label in the head. Nevertheless, it would be possible to apply boosting over more complicated classifiers, such as rules with several literals.

Boosting generic learners, which are not designed for time series, such as decision trees, can produce good results for time series classification. Nevertheless, the incorporation of temporal predicates, as background knowledge, improves the performance of boosting, especially with respect to the size of the classifiers.

Another method to improve the accuracy of a classifier is the use of pruning techniques. Pruning also enhances the comprehensibility of classifiers, while boosting worsens it. Nevertheless, pruning and boosting are not incompatible, e.g., when boosting rules it is possible to prune them [11]. Hence, it would be convenient to incorporate pruning techniques in the rule induction system, and also to apply boosting over pruned rules.

Acknowledgements

To the maintainers of the ML [7] and KDD [4] UCI Repositories. To Mohammed Waleed Kadous, David Aha and Robert J. Alcock for donating, respectively, the *Auslan*, *Wave* and *Control Charts* datasets and for their support.

References

[1] Robert J. Alcock and Yannis Manolopoulos. Time-series similarity queries employing a feature-based approach. In 7^{th} *Hellenic Conference on Informatics*, Ioannina, Greece, 1999.

[2] Carlos J. Alonso González and Juan J. Rodríguez Diez. A graphical rule language for continuous dynamic systems. In Masoud Mohammadian, editor, *Computational Intelligence for Modelling, Control and Automation.*, volume 55 of *Concurrent Systems Engineering Series*, pages 482–487, Amsterdam, Netherlands, 1999. IOS Press.

[3] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36(1/2):105–139, 1999.

[4] Stephen D. Bay. The UCI KDD archive, 1999. http://kdd.ics.uci.edu/.

[5] Francesco Bergadano and Deaniele Gunetti. *Inductive Logic Programming: from machine learning to software engineering.* The MIT Press, 1995.

[6] D.J. Berndt and J. Clifford. Finding patterns in time series: a dynamic programming approach. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 229–248. AAAI Press /MIT Press, 1996.

[7] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[8] Hendrik Blockeel and Luc De Raedt. Lookahead and discretization in ILP. In Nada Lavrac and Saso Dzeroski, editors, *7^{th} International Workshop on Inductive Logic Programming, ILP'97*, volume 1297 of *Lecture Notes in Computer Science*, pages 77–85. Springer, 1997.

[9] L. Breiman, J.H. Friedman, A. Olshen, and C.J. Stone. *Classification and Regression Trees.* Chapman & Hall, New York, 1993. Previously published by Wadsworth & Brooks/Cole in 1984.

[10] Peter Clark and Robin Boswell. Rule induction with CN2: Some recent improvements. In Yves Kodratoff, editor, *5^{th} European Conference on Machine Learning (EWSL-91)*, pages 151–163. Springer-Verlag, 1991.

[11] William W. Cohen and Yoram Singer. A simple, fast, and effective rule learner. In *AAAI-99*, 1999.

[12] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 1999.

[13] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, January 1995.

[14] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *13^{th} International Conference on Machine Learning (ICML-96)*, pages 148–156, Bari, Italy, 1996.

[15] Michael Harries. Boosting a strong learner: Evidence against the minimum margin. In Ivan Bratko and Saso Dzeroski, editors, *16^{th} International Conference of Machine Learning (ICML-99).* Morgan Kaufmann, 1999.

[16] Mohammed Waleed Kadous. Learning comprehensible descriptions of multivariate time series. In Ivan Bratko and Saso Dzeroski, editors, *Proceedings of the* 16^{th} *International Conference of Machine Learning (ICML-99).* Morgan Kaufmann, 1999.

[17] Volker Klingspor, Katharina J. Morik, and Anke D. Rieger. Learning concepts from sensor data of a mobile robot. *Machine Learning*, 23(2/3), 1997.

[18] M. Kubat, I. Koprinska, and G. Pfurtscheller. Learning to classify biomedical signals. In R.S. Michalski, I. Bratko, and M. Kubat, editors, *Machine Learning and Data Mining*, pages 409–428. John Wiley & Sons, 1998.

[19] Raymond J. Mooney and Mary Elaine Califf. Induction of first-order decision lists: results on learning the past tense of english verbs. *Journal of Artificial Intelligence Research*, 3:1–24, 1995.

[20] J.Ross Quinlan. *C4.5: programs for machine learning.* Machine Learning. Morgan Kaufmann, San Mateo, California, 1993.

[21] J.Ross Quinlan. Boosting, bagging, and C4.5. In *13th National Conference on Artificial Inteligence (AAAI'96)*, pages 725–730. AAAI Press, 1996.

[22] J.Ross Quinlan. Boosting first-order learning. In *ALT'96*, 1996.

[23] J.Ross Quinlan. Learning first-order definitions of functions. *JAIR*, pages 139–161, October 1996.

[24] Juan J. Rodríguez and Carlos J. Alonso. Applying boosting to similarity literals for time series classification. In *Proceedings of the 1st International Workshop on Multiple Classifier Systems (MCS2000)*, Lecture Notes in Computer Science. Springer Verlag, 2000.

[25] Juan J. Rodríguez Diez and Carlos J. Alonso González. Time series classification through clauses restricted to literals on intervals. In Ana M. García Serrano, Ramón Rizo, Serafín Moral, and Francisco Toledo, editors, *8th Conference of the Spanish Association for Artificial Intelligence (CAEPIA'99)*, pages 125–132, Murcia, Spain, November 1999. In spanish.

[26] Naoki Saito. *Local Feature Extraction and Its Applications Using a Library of Bases.* PhD thesis, Department of Mathematics, Yale University, 1994.

[27] Robert E. Schapire. Using output codes to boost multiclass learning problems. In *14th International Conference on Machine Learning (ICML-97)*, pages 313–321, 1997.

[28] Robert E. Schapire. A brief introduction to boosting. In Thomas Dean, editor, *16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1401–1406. Morgan Kaufmann, 1999.

[29] Ashwin Srinivasan and Rui C. Camacho. Numerical reasoning with an ilp system capable of lazy evaluation and customized search. *Journal of Logic Programming*, 40(2–3):185–213, August 1999.

[30] Wim Van Laer, Luc De Raedt, and Saso Dzeroski. On multi-class problems and discretization in inductive logic programming. In Zbigniew W. Ras and Andrzej Skowron, editors, *10th International Symposium on Methodologies for Intelligent Systems (ISMIS97)*, volume 1325 of *Lecture Notes in Artificial Intelligence*, pages 277–286. Springer-Verlag, 1997.

[31] Geoffrey I. Webb. MultiBoosting: A technique for combining boosting and wagging. *Machine learning*, 1999. In press.