Boosting Interval Based Literals*

Juan J. Rodríguez † Carlos J. Alonso ‡ Henrik Boström §

December 26, 2000

Abstract

A supervised classification method for time series, even multivariate, is presented. It is based on boosting very simple classifiers: clauses with one literal in the body. The background predicates are based on temporal intervals. Two types of predicates are used: i) relative predicates, such as "increases" and "stays", and ii) region predicates, such as "always" and "sometime", which operate over regions in the domain of the variable. Experiments on different data sets, several of them obtained from the UCI ML and KDD repositories, show that the proposed method is highly competitive with previous approaches.
Keywords: time series classification, interval based literals, boosting, machine learning.

1 Introduction

Multivariate time series classification is useful in domains such as biomedical signals [KKP98], continuous systems diagnosis [AGRD99] and data mining in temporal databases [BC96]. 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 [Kad99]: 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 could allow for 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 often referred to as variables, since they vary over time. From a machine learning point of view, each point of each series is an attribute of the example.

The method for learning time series classifiers that we propose in this work is based on literals over temporal intervals (such as increases or always in region) and boosting (a method for the generation of ensembles of classifiers) [Sch99].

Nevertheless, the method is also of interest in problems were the examples are not time series. In fact, several of the data sets used in the experimental valdiation are not time series problems. This method can be used whenever i) the attributes (or a subset of them) have values in the same domain and ii) there is an order relation between these attributes. However, in order to get some advantage of using intervals, this method is adequate when the values of the attributes are someway related, i.e., they are not independent.

This method was introduced in [RAB00], in a less focused way: more learning methods (also rule learning) and more predicates (also distance based), less details and with a preliminar experimental validation.

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

†*Affiliation:* Lenguajes y Sistemas Infromáticos. Universidad de Burgos. *Address:* Escuela Universitaria Politécnica. Avenida de Cantabria s/n. 09006 Burgos. Spain. *e-mail:* jjrodriguez@ubu.es

‡*Affiliation:* Departamento de Informática. Universidad de Valladolid. *Address:* Edificio Nuevas Tecnologías. Campus Miguel Delibes s/n. 47009 Valladolid. Spain *e-mail:* calonso@infor.uva.es

§*Affiliation:* Department of Computer and System Sciences. Stockholm University/KTH. *Address:* Electrum 230. S-164 40 Kista, Sweden. *e-mail:* henke@dsv.su.se

```
class( E, [ thank, maybe, name, man, science ] ) :- true_percentage( E, z, 1_4, 12, 16, 70 ). % 79, 30. 0.312225
class( E, [ thank, science, right, maybe, read ] ) :- true_percentage( E, roll, 1_4, 2, 10, 50 ). % 77, 8. 0.461474
class( E, [ maybe, right, man, come, thank ] ) :- true_percentage( E, z, 1_3, 2, 18, 50 ). % 70, 18. 0.339737
class( E, [ thank, come, man, maybe, mine ] ) :- true_percentage( E, roll, 5, 6, 14, 50 ). % 64, 27. 0.257332
class( E, [ girl, name, mine, right, man ] ) :- not true_percentage( E, z, 1_2, 6, 14, 5 ). % 78, 22. 0.361589
```

Figure 1: Initial fragment of an ensemble of classifiers, obtained with AdaBoost.OC, for the data set *Auslan* (section 4.8). At the right of each clause the number of positive and negative covered examples and the weight of the classifier.

The rest of the paper is organized as follows. Section 2 is a brief introduction to boosting, suited to our method. The base classifiers are described in section 3, including techniques for efficiently handling the special purpose predicates. Section 4 presents experimental results when using the new method. Finally, we give some concluding remarks in section 5.

2 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. 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 [Sch99], 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 weight of each example is readjusted, based on the correctness of the class assigned to the example by the base classifier. The final result is obtained by weighted votes of the base classifiers.

Inspired by the good results of works using ensembles of very simple classifiers [Sch99], sometimes named *stumps*, we have studied base classifiers consisting of clauses with only one literal in the body.

Multiclass problems There are several methods of extending AdaBoost to the multiclass case [Sch99]. We have used AdaBoost.OC [Sch97] 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. 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 base learner searches for a rule with the head:

class(Example, [class_1, ... class_k])

This predicate means that the Example is of one of the classes in the list. Figure 1 shows a fragment of a classifier with rules on this form.

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 the weights of all the labels in the list are increased by the weight of the rule, if it is false the weights of the labels out of the list are incremented. Finally, the label that has been given the highest weight is assigned to the example.

3 Base Classifiers

3.1 Predicates

Figure 2 shows a classification of the predicates. Point based predicates use only one point of the series:

- point_region(Example, Variable, Region, Point) it is true if, for the Example, the value of the Variable at the Point is in the Region.

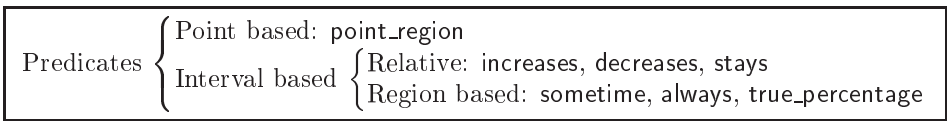

Figure 2: Classification of the predicates.

Note that a learner which only uses this predicate is equivalent to an attribute-value learning algorithm. This predicate is introduced to test the results obtained with boosting without using interval based predicates.

Two kinds of interval predicates are used: relative and region based. Relative predicates consider the differences between the values in the interval. Region based predicates are based on the presence of the values of a variable in a region during an interval.

3.1.1 Relative Predicates

A natural way of describing series is to indicate when they increase, decrease or stay. These predicates deal with these concepts:

- increases(Example, Variable, Beginning, End, Value). It is true, for the Example, if the difference between the values of the Variable for End and Beginning is greater or equal than Value.
- decreases(Example, Variable, Beginning, End, Value).
- stays(Example, Variable, Beginning, End, Value). It is true, for the Example, if the range of values of the Variable in the interval is less or equal than Value.

Frequently, series are noisy and, hence, a strict definition of increases and decreases in an interval i.e., the relation holds for all the points in the interval, is not useful. It is possible to filter the series prior to the learning process, but we believe that a system for time series classification must not rely on the assumption that the data is clean. For these two predicates we consider what happens only in the extremes of the interval. The parameter value is necessary for indicating the amount of change.

For the predicate stays it is neither useful to use a strict definition. In this case all the points in the interval are considered. The parameter Value is used to indicate the maximum allowed difference between the values in the interval.

3.1.2 Region Based Predicates

The selection and definition of these predicates is based in the ones used in a visual rule language for dynamic systems [AGRD99]. These predicates are:

- 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).
- 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 is decided to work with temporal intervals, the use and definition of the predicates always and sometime is natural, due to the fact that they are the extension of the conjunction and disjunction to intervals. Since one appears too demanding and the other 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).

Regions. The regions that appear in the previous predicates are intervals in the domain of values of the variable. In some cases the definitions of these regions can be obtained from an expert, as background knowledge. 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 $1 \ldots i$ (less or equal tests).

The reasons for fixing the regions before the classifier induction, instead of obtaining them while inducing, are efficiency and comprehensibility. The literals are easier to understand if the regions are few, fixed and not overlapped.

3.2 Searching Literals

The weak learner receives a set of examples, labeled as positive or negative. Its mission is to find the best body literal for a clause that discriminates positive from negative examples. Then it is necessary to search over the space of literals. For each body literal considered it is necessary to find out which positive and negative examples are covered by the corresponding clause.

The possible number of intervals, if each series has n points, is $(n^2 - n)/2$. With the objective of reducing the search space, not all the intervals are explored. Only those that are of size power of 2 are considered. The number of these intervals, for $k = \lfloor \lg n \rfloor$, is

$$\sum_{i=1}^{k} (n - 2^{i-1}) = kn - 2^k - 1 \in O(n \lg n)$$

Given an interval and an example, the predicates increases and decreases can be evaluated in $O(1)$, because only the extremes of the interval are considered. For the other interval predicates this time is $O(w)$, where w is the number of points in the interval.

It is necessary, when searching for the best literal of a given predicate, to calculate how many examples of each class are covered for each considered interval. A simple method for this would be to consider all the intervals and for each interval and example to evaluate the literal.

A better method is possible, if the relationships among intervals are considered. When a literal is evaluated, some information is saved for posterior use. Then, the evaluation of a literal with an interval of width $2w$ is obtained from the previous evaluation of two literals whose intervals are consecutive and with widths w.

Figure 3 describes this process in an abstract way. Some details, such as how to handle multivariate series or regions (for region based predicates) are left out in the description, but their inclusion are straightforward. In a first step (initialization), all intervals between two consecutive points are considered. For each example and interval, some info is initialized. Then this interval is evaluated, according to the number of examples of each class that are true. In the second step (combination), the info calculated for two consecutive intervals is combined for getting the info of the union interval. There are two nested loops: the first one (variable j) considers the sizes of the intervals and the second one (variable i) considers the begining of the intervals.

The evaluation of an interval, figure 4, counts how many examples of each class are true for the literal considered. The array covered keeps this information. Note that dealing with weigthed examples is as simple as substituting the unitarian increment by the addition of $e.\mathsf{weight}$. The procedure EVALUATE_LITERAL uses the information of the array covered to decide if the actual literal is the best found until now.

Since there are additional arguments for the predicates (e.g., the parameter value in relative predicates), the truth value of the literals depends also on the values of these parameters. In figure 4 this is represented using dots $(\ldots)$ to indicate that additional parameters could be necessary.

The details for the different predicates considered are shown in table 1. For instance, consider the predicate true_percentage. When evaluating it, two values are necessary for each region r: the width of the interval and the sum of the lengths of the sub-intervals in the interval where the value is in the region. These values are calculated for intervals between consecutive points in INITIALIZE. COMBINE only adds the attributes width and sum of two consecutive intervals. The function COVERED depends on an additional parameter, the percentage p.

Note that all the operations of table 1 are independent on the length of the intervals, they are $O(1)$. Another interesting fact is that the same array info[_, _] is used when considering different

```
LITERAL_SELECTION
for i ← 1...n do
    for e ∈ Examples do
        INITIALIZE(info, e, i)
    EVALUATE_INTERVAL(info, i, 1)
j ← 2
while j ≤ n do
    i ← 1
    while i + j ≤ n + 1 do
        for e ∈ Examples do
            COMBINE(info, e, i, j/2)
        EVALUATE_INTERVAL(info, i, j)
        i ← i + 1
    j ← 2j
```

Figure 3: Selection of literals.

```
EVALUATE_INTERVAL(info, i, j)
covered[...] ← 0
for e ∈ Examples do
    if COVERED(e, info, i, j, ...) then
        covered[e.class, ...] ← covered[e.class, ...] + 1
EVALUATE_LITERAL(covered)
```

Figure 4: Evaluation of intervals.

increases / **decreases** (for a value v)	
INITIALIZE(info, e, i)	-
COMBINE(info, e, i, j)	-
COVERED(e, info, i, j, v)	$e[j-1] - e[i] \geq v$ (for increases) $e[i] - e[j-1] \geq v$ (for decreases)
stays (for a value v)	
INITIALIZE(info, e, i)	info$[e,i]$.min ← info$[e,i]$.max ← $e[i]$
COMBINE(info, e, i, j)	info$[e,i]$.min ← MIN(info$[e,i]$.min, info$[e,i+j]$.min) info$[e,i]$.max ← MAX(info$[e,i]$.max, info$[e,i+j]$.max)
COVERED(e, info, i, j, v)	info$[e,i]$.max − info$[e,i]$.min $\leq v$
always / **sometime** (for a region r)	
INITIALIZE(info, e, i)	info$[e,i] \leftarrow e[i] \in r$
COMBINE(info, e, i, j)	info$[e,i] \leftarrow$ info$[e,i] \vee$ info$[e,i+j]$ (for sometime) info$[e,i] \leftarrow$ info$[e,i] \wedge$ info$[e,i+j]$ (for always)
COVERED(e, info, i, j)	info$[e,i]$
true_percentage (for a region r, a percentage p)	
INITIALIZE(info, e, i)	info$[e,i]$.width ← WIDTH$(i, i+1)$ info$[e,i]$.sum ← WIDTH$(i, i+1)$ **if** $e[i] \in r$ **else** 0
COMBINE(info, e, i, j)	info$[e,i]$.width ← info$[e,i]$.width + info$[e,i+j]$.width info$[e,i]$.sum ← info$[e,i]$.sum + info$[e,i+j]$.sum
COVERED(e, info, i, j, p)	100 info$[e,i]$.sum / info$[e,i]$.width $\geq p$

Table 1: Definition of the procedures for the literals.

	Classes	Examples	Points	Variables
Waveform	3	900	21	1
Wave + noise	3	900	40	1
Shifted Wave	2	600	40	1
CBF	3	798	128	1
Control charts	6	600	60	1
Sonar	2	208	60	1
Iono-1	2	351	34	1
Iono-2	2	351	17	2
Auslan	10	200	20	8

Table 2: Characteristics of the data sets

	1	2	3	4	5
point_region	•		○	○	○
increases		•			•
decreases		•			•
stays		•			•
always			•	○	○
sometime			•	○	○
true_percentage				•	•

Table 3: Predicates used in each experimental setting. The symbol '•' indicates that the predicate is used in the experiment, and '○' indicates that the predicate is not used but there is another one that can express all its conditions.

interval lengths. Hence, the memory needed is in the order of the number of points, not in the number of intervals.

The time necessary for the selection of the best literal is linear in the number of examples, the number of variables, the number of regions (for region based predicates) and the number of intervals (which is $O(n \lg n)$). There are also additional costs for selecting the best additional parameters for some predicates (i.e., the parameter Value for relative predicates and the parameter percentage for true_percentage). If the possible values for this parameters are fixed, then the selection of one of them is linear in the number of values allowed.

4 Experimental Validation

The characteristics of the data sets are summarized in table 2. Note that data sets for classification of time series are not easy to find [Kad99]. For each data set, 5 settings were considered. The predicates used in each setting are shown in table 3. The values considered for the parameter Value of relative predicates were multiples of the range of the variable divided by 20. For region based predicates, 6 regions were considered. The percentages considered for the predicate true_percentage were 5, 15, 30, 50, 70, 85 and 95.

The error rates for each data set and setting were obtained using 10-fold stratified cross-validation. Table 4 compares the results of settings 1 and 5, for different number of iterations, considering for how many datasets the results for one setting is better than the results for the other one. Significant results were obtained using the binomial test [Sal97]. This table shows a clear advantage of interval based predicates over point based ones.

Table 5 summarises the results for each data set. It shows the results using 100 iterations with settings 1 and 5. For all the data sets, the results using setting 5 are better than using setting 1. Significance results were obtained using McNemar's test [Die98], because it is non-parametric, and, hence, no assumption, e.g. the test sets are independent, is made. For table 5 the differences are significant at the 0.05 level on 5 cases and at the 0.01 level in 3 cases.

Graphs for the results on these two settings are shown, for each data set, in figure 5. For 5 of the 9 data sets, the results are always better in setting 5 than in setting 1. These graphs show that

Iter.	10	20	30	40	50	60	70	80	90	100
Win-Loss	7-2	8-1	8-1	8-1	9-0	8-1	8-1	8-1	8-0	9-0
Signific.	0.180	▷0.039	▷0.039	▷0.039	▷0.004	▷0.039	▷0.039	▷0.039	▷0.008	▷0.004

Table 4: Interval based vs. point based results. *Win* indicates the number of datasets such as the error is smaller for setting 5 than for setting 1. The symbol '▷' marks values ≤ 0.05.

	Error set. 1	Error set. 5	*Significance 1 vs 5*	Decision Tree	*Nodes DT*	Boost 10 DT	Boost 100 DT
Waveform	15.67	**14.78**	*0.461*	23.89	*125*	19.33	15.67
Wave + noise	16.44	15.78	*0.624*	23.78	*137*	18.67	**15.67**
Shifted Wave	42.50	**35.00**	▷*0.002*	46.33	*83*	44.50	37.50
CBF	2.11	**0.62**	▷*0.001*	9.27	*49*	3.38	2.38
Control charts	4.50	**0.00**	▷*1e-08*	8.50	*35*	3.17	1.00
Sonar	17.46	15.98	*0.711*	22.12	*35*	22.12	**12.98**
Iono-1	7.69	6.85	*0.648*	11.11	*35*	6.84	**6.27**
Iono-2	9.72	6.29	▷*0.036*	11.11	*35*	6.84	**6.27**
Auslan	7.5	**3.00**	▷*0.012*	20.50	*31*	11.00	6.00

Table 5: Results for all the data sets, settings 1 and 5 with 100 iterations, decision trees, and boosted decision trees

the gain obtained by using interval based predicates depends greatly on the data set.

Table 5 also includes results for decision trees and boosting decision trees. They were obtained using the WEKA library [WF99]. The decision tree method, J48, is based on C4.5 and the boosting variant used is AdaBoost.M1 (AdaBoost.OC is currently not included in this library). Each base learner in M1 discriminates between all the classes, and in OC discriminates only between two groups of clases. Hence, it seems that OC will need more iterations than M1 for obtaining comparable results and using the same number of iterations gives adavantage to boosting decision trees over boosting interval literals.

The results for decision trees are not directly comparable with the results of boosting interval literals because the folds used in the 10-fold cross validation process are not the same. The best error results are rather evenly distributed (5–4) between our setting 5 and boosting 100 decision trees. This is specially compelling when considering that each decision tree is far more complicated than one interval literal. As an indication of the size of the trees, table 5 includes the size of the decision tree obtained using all the examples of each data set.

The rest of this section contains a detailed discussion for each data set, including its description, the results for the five settings and different number of iterations, and a comparison of the results for settings 2–5 (combinations of interval literals) against the results for setting 1 (point based literals), using the McNemar's test.

4.1 Waveform

This data set was introduced by [BFOS93]. The purpose is to distinguish between three classes, defined by the evaluation for $i = 1, 2 \ldots 21$, of the following functions:

$$\begin{aligned} x_1(i) &= uh_1(i) + (1-u)h_2(i) + \epsilon(i) \\ x_2(i) &= uh_1(i) + (1-u)h_3(i) + \epsilon(i) \\ x_3(i) &= uh_2(i) + (1-u)h_3(i) + \epsilon(i) \end{aligned}$$

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 6 shows two examples of each class. We used the version from the UCI ML Repository [BM98]. The results for this data set are shown in table 6.

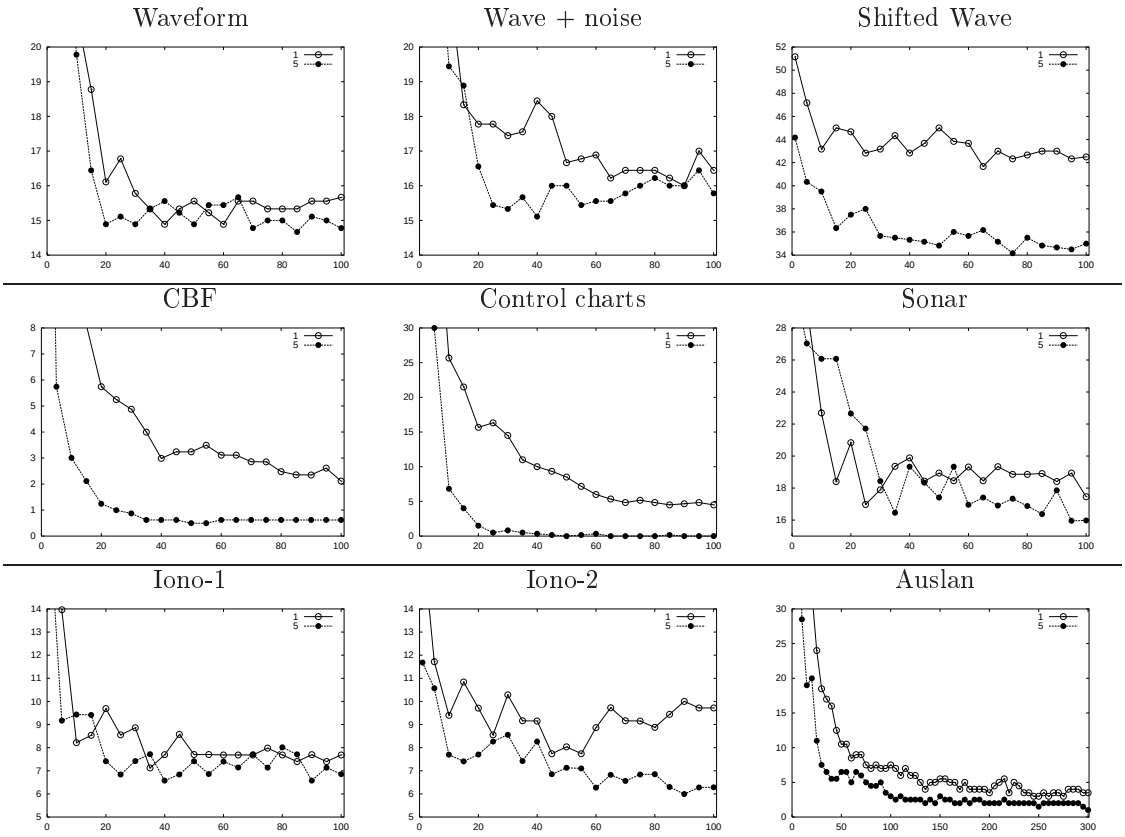

Figure 5: Graphs of the error rates for all the data sets, settings 1 and 5. Note that the scales used are different.

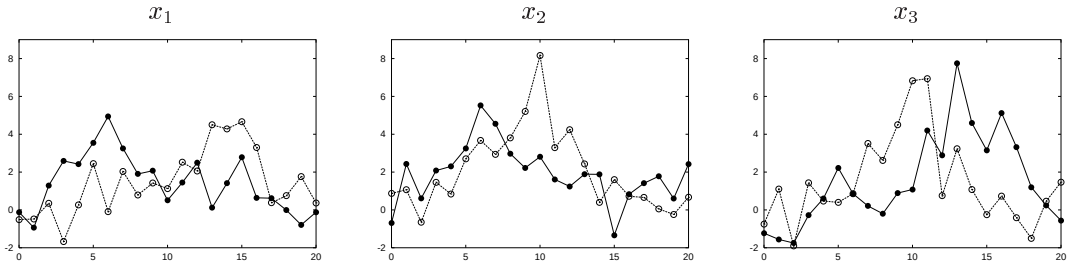

Figure 6: Examples of the Waveform data set. Two examples of the same class are shown in each graph.

Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	21.00	16.11	15.78	14.89	15.56	14.89	15.56	**15.33**	15.56	15.67
	2	18.67	16.44	15.78	15.78	**14.67**	**14.67**	14.89	14.78	**14.67**	**14.67**
	3	19.00	15.33	15.56	**14.67**	15.00	15.33	14.89	15.22	16.33	16.56
	4	19.00	16.67	14.78	15.00	14.89	**14.44**	14.56	15.11	15.33	15.33
	5	19.78	14.89	14.89	15.56	14.89	15.45	**14.78**	15.00	15.11	14.78
Signific.	2	0.110	•0.863	1.000	•0.528	0.512	0.923	0.634	0.709	0.501	0.444
	3	0.179	0.597	0.923	0.920	0.682	•0.749	0.594	1.000	•0.494	•0.434
	4	0.187	•0.712	0.422	•1.000	0.581	0.734	0.368	0.909	0.908	0.820
	5	0.431	0.355	0.497	•0.624	0.598	•0.672	0.530	0.838	0.749	0.461

Table 6: Results for the Waveform data set. In boldface, the best result for each setting. The symbol '•' indicates that the result is better for setting 1 than for the other setting.

The error of a Bayes optimal classifier on this data set, obtained analytically from the functions that generate the examples, is approximately 14 [BFOS93]. There are several works that use this data set with boosting. The best results, we know, from all of them is 15.21 reported in [Die99] That result was obtained using boosting, with decision trees as base classifiers, which are much more complex than our base classifiers (clauses with one literal in the body). Recently, a best result of 14.30 is reported in [TD00]. This result was obtained using meta decision trees, combining models of two decision trees learners, a rule learner, a nearest neighbor algorithm and a naive Bayes algorithm. Our best result is 14.44 for the setting 4 using 60 iterations, and several values are smaller than 15.

4.2 Wave + noise

This data set is generated in the same way than the previous one, but 19 random points are added at the end of each example, with mean 0 and variance 1. Again, we used the data set from the UCI ML Repository, and the error of a Bayes optimal classifier is 14. Our results are shown in table 7 This data set was tested with bagging, boosting and variants over MC4 (similar to C4.5) [BK99], 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. Our result for setting 5 with 100 iterations is 15.78.

4.3 Shifted wave

The results on the previous data sets do not show clear improvements using interval predicates. Our conjecture is that interval based predicates are advantageous over point predicates whenever there are shifts, expansions or compressions among the examples of the same class. To check this conjecture we generated this data set from the previous one, introducing shifts.

Each example of the first and second classes of the previous data set was shifted to the right a random number of positions, between 0 and 39. In each shift, every value is substituted for the

Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	21.89	17.78	17.44	18.44	16.67	16.89	16.44	16.44	**16.00**	16.44
	2	18.67	16.44	15.78	16.67	16.33	16.33	**15.33**	16.11	16.00	**15.33**
	3	20.56	17.33	16.89	17.00	17.11	**16.67**	17.00	16.89	16.67	16.78
	4	20.44	17.89	16.44	**15.67**	16.89	16.89	17.78	17.11	17.44	17.11
	5	19.44	16.56	15.33	**15.11**	16.00	15.56	15.78	16.22	16.00	15.78
Signific.	2	▷0.038	0.356	0.221	0.188	0.863	0.721	0.415	0.859	1.000	0.407
	3	0.407	0.794	0.704	0.255	•0.775	0.921	•0.679	•0.757	•0.598	•0.834
	4	0.375	•1.000	0.471	▷0.026	•0.924	1.000	•0.290	•0.617	•0.228	•0.617
	5	0.107	0.363	0.104	▷0.006	0.631	0.290	0.640	0.923	1.000	0.624

Table 7: Results for the Wave + Noise data set

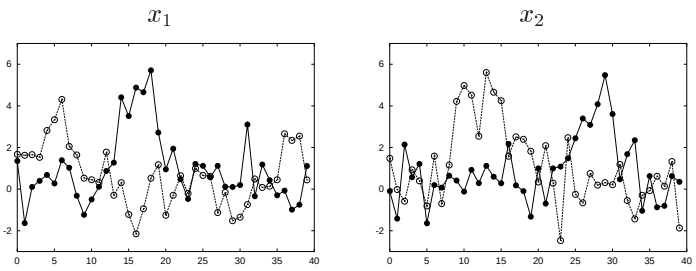

Figure 7: Examples of the Shifted Wave data set.

value at its left. The value in the last position is moved to the first position. The examples of the third class were not used because the formula which generates its examples is the same that the one used for generating the second one, shifted 4 positions ($x_3(i) = x_2(i+4)$). Figure 7 shows two examples of each class.

The results for this data set are shown in table 8. They show that this is a very difficult problem. For settings 4–5 and all the iterations considered the results are better than for the setting 1. For setting 5 and all iterations considered, except the first one, these differences are significant.

4.4 Cylinder, Bell and Funnel (CBF).

This is an artificial problem, introduced in [Sai94]. The learning task is to distinguish between three classes: cylinder (c), bell (b) or funnel (f). Examples are generated using the following functions:

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

Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	43.17	44.67	43.17	42.83	45.00	43.67	43.00	42.67	43.00	**42.50**
	2	**42.17**	43.33	42.33	42.83	43.17	43.83	44.50	44.67	43.83	43.33
	3	44.17	41.17	40.83	39.17	40.50	38.83	**38.17**	38.33	39.00	39.83
	4	42.00	40.33	42.33	41.67	37.83	40.00	39.00	**38.00**	39.67	39.50
	5	39.50	37.50	35.67	35.33	34.83	35.67	35.17	35.50	**34.67**	35.00
Signific.	2	0.756	0.671	0.806	1.000	0.514	•1.000	•0.606	•0.478	•0.798	•0.801
	3	•0.736	0.150	0.346	0.107	▷0.041	▷0.018	▷0.021	▷0.028	▷0.048	0.214
	4	0.704	0.084	0.781	0.676	▷0.002	0.135	0.086	▷0.039	0.147	0.192
	5	0.186	▷0.005	▷0.003	▷0.002	▷4e-05	▷0.001	▷0.001	▷0.002	▷3e-04	▷0.002

Table 8: Results for the Shifted Wave data set

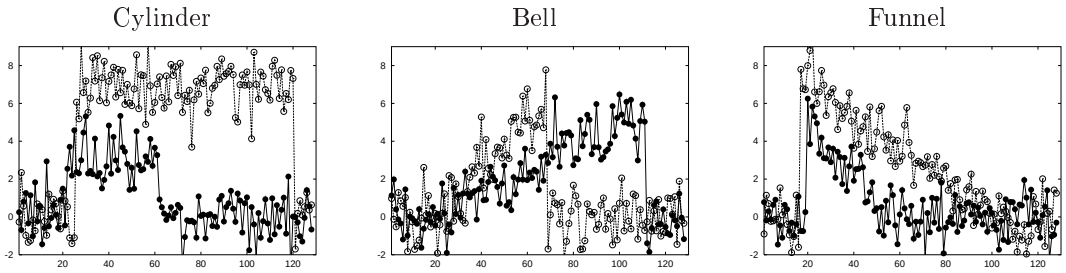

Figure 8: Examples of the CBF data set.

Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	9.63	5.74	4.88	2.99	3.24	3.11	2.86	2.48	2.35	**2.11**
	2	6.36	4.87	3.99	3.23	2.74	2.36	1.86	1.99	**1.74**	1.98
	3	3.62	2.11	1.62	1.61	1.99	1.62	1.49	1.62	1.50	**1.00**
	4	2.63	1.38	1.12	0.74	0.87	0.87	0.99	0.87	0.74	**0.62**
	5	3.01	1.24	0.87	0.62	**0.49**	0.62	0.62	0.62	0.62	0.62
Signific.	2	▷0.004	0.382	0.360	•0.864	0.618	0.362	0.152	0.523	0.383	1.000
	3	▷1e-08	▷1e-06	▷3e-06	▷0.019	0.064	▷0.012	▷0.019	0.143	0.118	▷0.022
	4	▷6e-12	▷6e-10	▷2e-08	▷4e-05	▷2e-05	▷4e-05	▷3e-04	▷0.001	▷0.001	▷0.002
	5	▷3e-11	▷1e-09	▷7e-08	▷2e-05	▷5e-07	▷2e-06	▷8e-06	▷6e-05	▷1e-04	▷0.002

Table 9: Results for the CBF data set

where

$$\chi_{[a,b]}(t) \quad = \quad \begin{cases} 0 & \text{if} \quad t < a \vee t > b \\ 1 & \text{if} \quad a \leq t \leq b \end{cases}$$

and 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 $t = 1, 2 \ldots 128$. Figure 8 shows some examples of this data set.

The results obtained for this data set are shown in table 9. The error reported in [Kad99] is 1.9, using event extraction, event clustering and decision trees. The results obtained with region based predicates (settings 3–5) are better than this value. Moreover, using true_percentage, this value is improved with only 20 iterations. For settings 4 and 5 the tests are always significant.

4.5 Control Charts

In this data set there are six different classes of control charts, synthetically generated by the process in [AM99]. Each time series is of length n, and it is defined by $y(t)$, with $1 \leq t \leq n$:

1. Normal: $y(t) = m + rs$. Where $m = 30$, $s = 2$ and r is a random number in $[-3, 3]$.
2. Cyclic: $y(t) = m + rs + a\sin(2\pi t/T)$. a and T are in $[10, 15]$.
3. Increasing: $y(t) = m + rs + gt$. g is in $[0.2, 0.5]$.
4. Decreasing: $y(t) = m + rs - gt$.
5. Upward: $y(t) = m + rs + kx$. x is in $[7.5, 20]$ and $k = 0$ before time t_3 and 1 after this time. t_3 is in $[n/3, 2n/3]$.
6. Downward: $y(t) = m + rs - kx$.

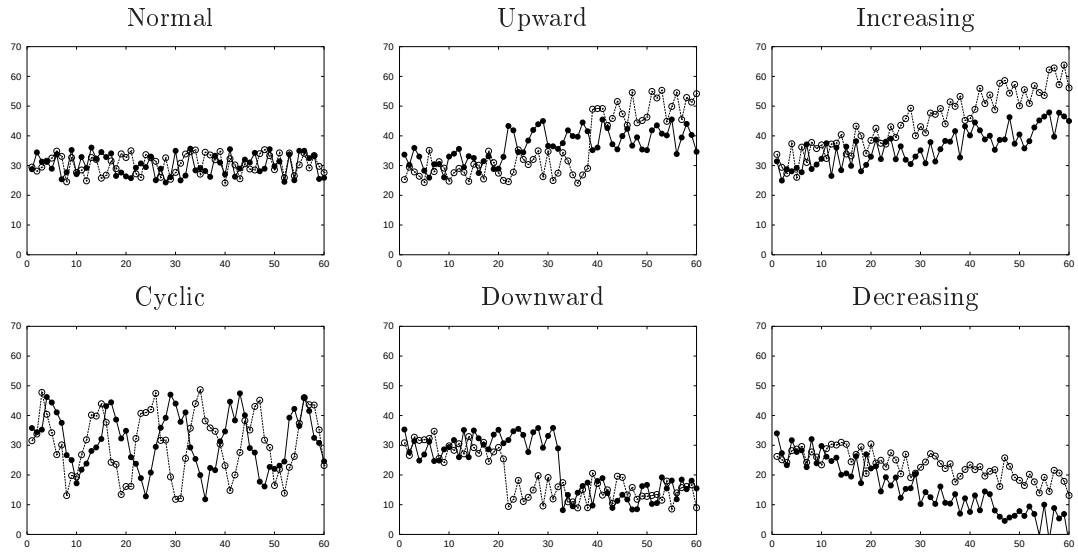

Figure 9: Some examples of the Control data set.

Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	25.67	15.67	14.50	10.00	8.50	6.00	4.83	4.83	4.67	**4.50**
	2	27.50	8.83	4.33	2.67	1.67	**0.50**	0.67	0.67	0.83	**0.50**
	3	6.33	1.67	1.67	1.00	1.00	1.00	0.83	**0.50**	0.83	**0.50**
	4	7.17	2.67	1.83	1.17	1.67	1.50	1.17	1.33	**1.00**	1.17
	5	6.83	1.50	0.83	0.33	**0.00**	0.33	**0.00**	**0.00**	**0.00**	**0.00**
Signific.	2	•0.514	▷3e-04	▷4e-10	▷6e-08	▷3e-08	▷2e-09	▷5e-06	▷5e-06	▷3e-05	▷3e-05
	3	▷2e-20	▷3e-20	▷7e-18	▷6e-14	▷2e-11	▷2e-07	▷8e-06	▷9e-07	▷6e-06	▷3e-06
	4	▷1e-19	▷4e-17	▷1e-18	▷2e-14	▷2e-09	▷7e-06	▷3e-05	▷2e-04	▷6e-05	▷2e-04
	5	▷2e-19	▷3e-21	▷1e-22	▷1e-16	▷9e-16	▷1e-09	▷4e-09	▷4e-09	▷7e-09	▷1e-08

Table 10: Results for the Control data set

Figure 9 shows two examples of each class. The data used was obtained from the UCI KDD Archive [Bay99]. We are not aware of any result for this data set in a supervised classification setting from other authors. The results are shown in table 10. All the differences considered are significant, with only one exception.

4.6 Sonar

This data set was introduced in [GS88] and it is available at the UCI ML Repository [Bay99]. The task is to discriminate between sonar signals bounced off a metal cylinder and those bounced off a roughly cylindrical rock. Two examples of each class appear in figure 10.

In this data set the examples are not time series, instead each pattern is a set of 60 numbers in the range 0.0 to 1.0. Each number represents the energy within a particular frequency band, integrated over a certain period of time.

In this data set there is a specified partition of examples in training and testing. In this partition the training and testing sets were carefully controlled to ensure that each set contained cases from each aspect angle (this parameter does not appear explicitly in the data set, but the examples appear ordered by its value) in appropriate proportions.

Our results, using 10-fold cross validation and the specified partition are shown in table 11. The results reported in [GS88], using neural networks, are a best error of 15.3, with 13-fold cross-validation and 9.6 with the specified partition. Nevertheless, in [DCK93] these results could not be replicated. This data set has been frequently used, and there are even papers devoted to this

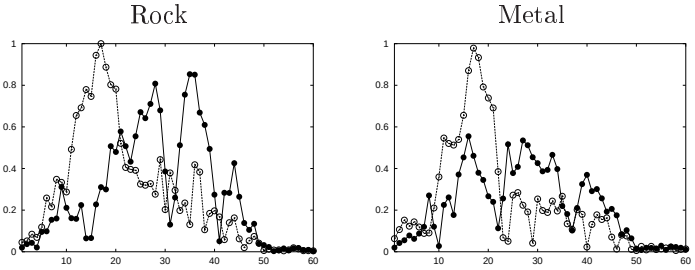

Figure 10: Some examples of the Sonar data set.

10-fold cross-validation											
Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	22.70	20.84	17.89	19.88	18.93	19.32	19.34	18.86	18.41	**17.46**
	2	25.65	18.77	18.81	18.36	17.38	16.90	16.43	16.43	**15.48**	16.43
	3	22.73	21.32	20.77	18.44	17.98	18.86	16.44	16.46	15.51	**13.55**
	4	20.31	17.93	16.93	16.95	16.93	17.41	**13.50**	14.53	14.45	14.46
	5	26.08	22.65	18.43	19.34	17.41	16.95	16.91	16.88	17.86	**15.98**
Signific.	2	•0.497	0.659	•0.878	0.749	0.755	0.551	0.418	0.522	0.418	0.864
	3	1.000	•1.000	•0.362	0.690	0.839	1.000	0.362	0.458	0.327	0.169
	4	0.522	0.405	0.875	0.377	0.608	0.597	▷0.042	0.108	0.200	0.327
	5	•0.382	•0.636	•1.000	1.000	0.743	0.500	0.458	0.541	1.000	0.711
Specified partition											
Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	26.92	18.27	23.08	**14.42**	21.15	20.19	19.23	23.08	24.04	23.08
	2	17.31	18.27	17.31	17.31	18.27	15.38	15.38	**13.46**	15.38	16.35
	3	28.85	21.15	18.27	16.35	20.19	19.23	16.35	**15.38**	19.23	17.31
	4	16.35	15.38	15.38	16.35	15.38	16.35	18.27	17.31	18.27	**13.46**
	5	17.31	13.46	13.46	14.42	14.42	13.46	**10.58**	**10.58**	13.46	11.54
Signific.	2	0.076	1.000	0.286	•0.664	0.678	0.383	0.481	0.053	0.078	0.167
	3	•0.839	•0.664	0.383	•0.815	1.000	1.000	0.648	0.077	0.302	0.210
	4	0.071	0.664	0.134	•0.791	0.180	0.388	1.000	0.180	0.146	▷0.006
	5	0.064	0.332	▷0.041	1.000	0.167	0.167	▷0.049	▷0.004	▷0.013	▷0.008

Table 11: Results for the Sonar data set

problem only [TMG98, HR99]. The somewhat surprising fact is that this data set is linearly separable was discovered recently. The best error reported in [HR99] is an error of 9.96, for the specified partition. Our results for the specified partition, setting 5, 100 iterations is an error of 11.54, our best result is 10.58.

4.7 Ionosphere

This data set, also from the ML UCI Repository, contains information collected by a radar system [SWHB89]. The targets were free electrons in the ionosphere. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. For this data set also there exists a specified partition: 200 instances are used for training, which were carefully split almost 50% positive and 50% negative. The test set is formed by the rest of examples, and the distribution of examples in this set is rather uneven (124 vs. 27).

The examples of this data set neither are time series:

> "Received signals were processed using an autocorrelation function whose arguments are the time of a pulse and the pulse number. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse

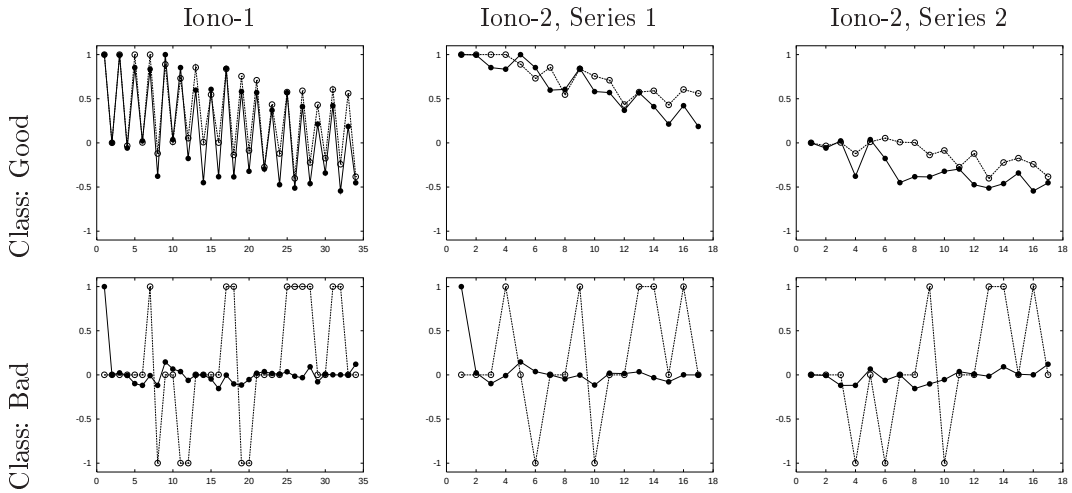

Figure 11: Some examples of the Ionosphere data set.

number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal."

We consider two versions of this data set, in the first one series for each example was used and in the second one, two series for example was used (one series for the real part and another for the imaginary part of the complex numbers). Two examples of each class are shown in figure 11.

Our results are shown in tables 12 and 13. For this data set the differences between using 10 or 100 iterations is, in general, very small; and even in several occasions the results are better for 10 than for 100.

For the two variants, it is remarkable that the results using 10-fold cross validation are nearly always better for settings 4–5 than for setting 1, while for the specified partition a lot of times the results are better for setting 1 than for the rest of settings.

With respect to the relationship between the results of the two variants, the first clear point is that there is not a clear advantage of using two series instead of one. Another interesting issue is the results for setting 1. The results for variant 1 seem better than the results for variant 2. If we use the point based literal, the fact that there are one or two series apparently is unimportant. Nevertheless there is one difference: the discretization process is applied as many times as series. The fact that the results are better using only one discretization suggest that this process must be further studied.

The best result reported in [SWHB89] is an error of 4, using backpropagation, and in [AK89], using instance based learning, is an error of 3.3. Our result for setting 5, 100 iterations, is an error of 4.64. Nevertheless, for setting 4 the error is 1.99.

4.8 Auslan

Auslan is the Australian sign language, the language of the Australian deaf community. Instances of the signs were collected using an instrumented glove [Kad99]. Each example is composed by 8 series: x, y and z position, wrist roll, thumb, fore, middle and ring finger bend. There are 10 classes and 20 examples of each class. The number of points in each example is variable and currently the system does not support variable length series, so they were reduced to 20 points (the series were divided in 20 segments along the time axis and the means of each segment were used as the values for the reduced series).

This is the data set with the highest number of classes (10) and also is the only one with more than one variable (8). Hence, we incremented the number of iterations for this data set, allowing up to 300 iterations. The results are shown in table 14. The results reported in [Kad99] is an error of 2.50, using event extraction, event clustering and Naïve Bayes Classifiers. Our result for setting 5, 300 iterations, is an error of 1.00.

10-fold cross-validation											
Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	8.22	9.69	8.86	**7.69**	7.70	**7.69**	**7.69**	**7.69**	**7.69**	**7.69**
	2	8.81	7.95	7.38	7.68	7.40	7.42	**6.85**	7.42	7.14	7.42
	3	**5.96**	6.84	7.40	7.71	7.12	6.84	8.57	7.12	7.43	6.84
	4	7.42	7.14	7.71	6.85	**5.99**	6.57	6.86	6.85	6.86	7.99
	5	9.43	7.42	7.42	**6.57**	7.41	7.40	7.71	8.02	**6.57**	6.85
Signific.	2	•0.860	0.405	0.473	1.000	1.000	1.000	0.664	1.000	0.832	1.000
	3	0.115	0.052	0.359	1.000	0.815	0.664	•0.678	0.832	1.000	0.678
	4	0.701	0.108	0.541	0.648	0.286	0.481	0.678	0.678	0.664	•1.000
	5	•0.608	0.169	0.424	0.503	1.000	1.000	1.000	•1.000	0.503	0.648
Specified partition											
Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	5.96	13.91	5.30	3.97	**3.31**	**3.31**	**3.31**	5.30	5.30	5.30
	2	8.61	6.62	4.64	4.64	5.96	5.30	4.64	**3.31**	**3.31**	**3.31**
	3	9.27	12.58	8.61	5.96	5.96	**5.30**	6.62	5.96	**5.30**	5.96
	4	16.56	4.64	5.96	3.97	3.97	3.97	3.97	3.31	**1.99**	**1.99**
	5	8.61	4.64	3.97	3.31	3.31	**2.65**	3.31	**2.65**	5.96	4.64
Signific.	2	•0.481	▷0.043	1.000	•1.000	•0.344	•0.453	•0.727	0.453	0.453	0.453
	3	•0.227	0.803	•0.227	•0.508	•0.344	•0.508	•0.227	•1.000	1.000	•1.000
	4	▷•0.001	▷0.001	•1.000	1.000	•1.000	•1.000	•1.000	0.453	0.125	0.125
	5	•0.340	▷0.001	0.688	1.000	1.000	1.000	1.000	0.219	•1.000	1.000

Table 12: Results for the Ionosphere-1 data set.

10-fold cross-validation											
Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	9.40	9.71	10.28	9.15	**8.04**	8.87	9.16	8.88	10.00	9.72
	2	7.95	9.11	6.83	6.82	7.39	7.68	7.97	7.39	7.13	**6.25**
	3	7.98	6.86	**5.71**	5.99	6.28	6.29	6.29	6.85	6.57	6.28
	4	6.85	5.42	5.70	5.70	**5.40**	5.71	5.99	6.28	6.28	5.99
	5	7.69	7.70	8.56	8.27	7.13	6.27	6.56	6.85	**5.99**	6.29
Signific.	2	0.442	0.860	0.043	0.152	0.845	0.572	0.585	0.458	0.076	▷0.029
	3	0.383	0.076	▷0.002	0.052	0.327	0.108	0.076	0.210	▷0.017	▷0.017
	4	0.078	▷0.001	▷1e-04	▷0.017	0.093	▷0.027	▷0.035	0.093	▷0.011	▷0.015
	5	0.238	0.189	0.263	0.678	0.664	0.108	0.108	0.210	▷0.007	▷0.036
Specified partition											
Iter.:		10	20	30	40	50	60	70	80	90	100
Error	1	5.30	5.96	**4.64**	**4.64**	5.96	5.30	6.62	7.28	7.28	7.28
	2	6.62	7.95	5.30	5.96	5.30	5.30	**3.97**	**3.97**	**3.97**	4.64
	3	**6.62**	10.60	7.28	7.95	7.28	**6.62**	7.28	**6.62**	**6.62**	7.95
	4	3.97	**3.31**	3.97	4.64	4.64	4.64	5.96	5.30	5.96	6.62
	5	5.96	6.62	5.96	5.96	5.96	5.96	**4.64**	5.30	5.96	5.96
Signific.	2	•0.774	•0.607	•1.000	•0.754	1.000	1.000	0.289	0.180	0.180	0.289
	3	•0.754	•0.118	•0.289	•0.125	•0.727	•0.688	•1.000	1.000	1.000	•1.000
	4	0.754	0.344	1.000	1.000	0.727	1.000	1.000	0.549	0.754	1.000
	5	•1.000	•1.000	•0.727	•0.727	1.000	•1.000	0.453	0.453	0.688	0.688

Table 13: Results for the Ionosphere-2 data set.

Iter.:		30	60	90	120	150	180	210	240	270	300
Error	1	18.50	8.50	7.00	6.00	5.50	4.00	5.00	**3.50**	**3.50**	**3.50**
	2	11.00	5.00	3.50	4.50	**3.00**	4.00	4.50	4.50	4.00	4.00
	3	11.00	5.00	4.00	4.50	**2.00**	2.50	2.50	3.00	2.50	2.50
	4	8.00	4.00	3.00	3.50	**2.00**	**2.00**	2.50	2.50	2.50	2.50
	5	7.50	5.00	5.00	2.50	3.00	2.00	2.00	2.00	2.00	**1.00**
Signific.	2	▷0.044	0.118	0.092	0.581	0.125	1.000	1.000	•0.688	•1.000	•1.000
	3	▷0.017	0.092	▷0.031	0.507	▷0.039	0.375	0.125	1.000	0.625	0.500
	4	▷0.001	▷0.035	▷0.039	0.125	▷0.039	0.289	0.125	0.625	0.688	0.688
	5	▷1e-04	0.143	0.424	▷0.039	0.125	0.289	▷0.031	0.250	0.375	0.063

Table 14: Results for the Auslan data set

Although the results for settings 3–5 are always better than the results for setting 1, these differences are significant only in few cases: the number of examples is small and the error rates for all the settings are low. Using 300 iterations, only 7 examples are missclassified for setting 1 and only two examples for setting 5.

5 Conclusions

A time series classification system has been developed. It is based on boosting very simple classifiers. The individual classifiers are formed by clauses with only one literal in the body. The predicates used are based on intervals. Two kinds of interval predicates are used: relative and region based. Relative predicates consider the differences between the values in the interval, while region based predicates consider the presence of the values of a variable in a region during an interval.

Experiments on several different data sets show that the proposed method is highly competitive with previous approaches. On several data sets, the proposed method achieves better than all previously reported results we are aware of. Moreover, although the strength of the method is based on boosting, the experimental results using point based predicates shows that the incorporation of interval predicates can improve significantly the obtained classifiers, especially when using less iterations.

Another interesting feature of the method is its simplicity. From a user point of view, the method has only one free parameter, the number of iterations. Moreover, the classifiers obtained with a number of iterations are included in the ones obtained with more iterations. Hence, it is possible i) to select only an initial fragment of the obtained classifier and ii) to continue adding base classifiers to a previously obtained classifier. Although less important, from the programmer point of view the method is also rather simple. The implementation of boosting of stumps is one of the easiest among classification methods.

One of the current limitations of the proposed method is the requirement that all the series are of the same length. We can consider two approaches, the first one is to preprocess the examples for normalizing the lengths and use the current method. This normalization can be as simple as the reduction used for the auslan data set, which gave good results, or more complex approachs, such as time warping methods [OFC99]. The second one is to alter the method for dealing with variable length time series. A possibility would be to use similar methods to the ones used for dealing with missing values in classical ML algorithms.

Boosting binary stumps produces good results, but the effect of using more complex base learners, such as decision trees or rules (of interval literals), must be studied. Currently the base learners only return the classification of the example. The use of confidence-rated predictions [SS98] may however improve the method.

Acknowledgements

To the maintainers of the ML [BM98] and KDD [Bay99] UCI Repositories, and to all the donators of the used data sets.

References

[AGRD99] 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.

[AK89] D. Aha and D. Kibler. Noise tolerant instance based learning algorithms. In *11^{th} International Joint Conference on Artificial Intelligence*, pages 794–799. Morgan Kaufmann, 1989.

[AM99] 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.

[Bay99] Stephen D. Bay. The UCI KDD archive, 1999. `http://kdd.ics.uci.edu/`.

[BC96] 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.

[BFOS93] 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.

[BK99] Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36(1/2):105–139, 1999.

[BM98] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[DCK93] R. A. Dunne, N. A. Campbell, and H. T. Kiiveri. Classifying high dimensional spectral data by neural networks. In *4^{th} Aust. Conf. on Neural Networks*, Sidney, 1993.

[Die98] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.

[Die99] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, 1999.

[GS88] R. P. Gorman and T. J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[HR99] Martina Hasenjäger and Helge Ritter. Perceptron learning revisited: The sonar targets problem. *Neural Processing Letters*, 10:1–8, 1999.

[Kad99] Mohammed W. Kadous. Learning comprehensible descriptions of multivariate time series. In Ivan Bratko and Saso Dzeroski, editors, *16^{th} International Conference of Machine Learning (ICML-99)*. Morgan Kaufmann, 1999.

[KKP98] 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.

[OFC99] Tim Oates, Laura Firoiu, and Paul R Cohen. Clustering time series with hidden markov models and dynamic time warping. In *IJCAI-99 Workshop on Neural, Symbolic and Reinforcement Learning Methods for Sequence Learning*, 1999.

[RAB00] Juan J. Rodríguez, Carlos J. Alonso, and Henrik Boström. Learning first order logic time series classifiers: Rules and boosting. In Zighed et al. [ZKZ00], pages 299–308.

[Sai94] Naoki Saito. *Local Feature Extraction and Its Applications Using a Library of Bases.* PhD thesis, Department of Mathematics, Yale University, 1994.

[Sal97] Steven Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328, 1997.

[Sch97] Robert E. Schapire. Using output codes to boost multiclass learning problems. In *14^{th} International Conference on Machine Learning (ICML-97)*, pages 313–321, 1997.

[Sch99] Robert E. Schapire. A brief introduction to boosting. In Thomas Dean, editor, *16^{th} International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 1401–1406. Morgan Kaufmann, 1999.

[SS98] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *11^{th} Annual Conference on Computational Learning Theory (COLT-98)*, pages 80–91. ACM, 1998.

[SWHB89] V. G. Sigillito, S. P. Wing, L. V. Hutton, and K. B. Baker. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262–266, 1989.

[TD00] Ljupčo Todorovski and Sašo Džeroski. Combining multiple models with meta decision trees. In Zighed et al. [ZKZ00], pages 54–64.

[TMG98] Juan M. Torres Moreno and Mirta B. Gordon. Perceptron learning revisited: The sonar targets problem. *Neural Processing Letters*, 7:1–4, 1998.

[WF99] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 1999.

[ZKZ00] Djamel A. Zighed, Jan Komorowski, and Jan Żytkow, editors. *Principles of Data Mining and Knowledge Discovery: 4^{th} European Conference; PKDD 2000*, volume 1910 of *Lecture Notes in Artificial Intelligence*, Lyon, France, September 2000. Springer.