
Machine Learning, 58, 179–216, 2005

Classification of Multivariate Time Series and Structured Data Using Constructive Induction

MOHAMMED WALEED KADOUS waleed@cse.unsw.edu.au
CLAUDE SAMMUT claude@cse.unsw.edu.au
School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

Editor: Eamonn Keogh

Abstract. We present a method of constructive induction aimed at learning tasks involving multivariate time series data. Using metafeatures, the scope of attribute-value learning is expanded to domains with instances that have some kind of recurring substructure, such as strokes in handwriting recognition, or local maxima in time series data. The types of substructures are defined by the user, but are extracted automatically and are used to construct attributes.

Metafeatures are applied to two real domains: sign language recognition and ECG classification. Using metafeatures we are able to generate classifiers that are either comprehensible or accurate, producing results that are comparable to hand-crafted preprocessing and comparable to human experts.

Keywords: time series, constructive induction, propositionalisation, substructure

1. Introduction

There are many domains that do not easily fit into the static attribute-value model so common in machine learning. These include multivariate time series, optical character recognition, sequence recognition, basket analysis and web logs. Consequently, researchers hoping to apply attribute-value learners to these domains have few choices: apply hand-crafted preprocessing, write a learner specifically designed for the domain, or use a learner with a more powerful representation, such as relational learning or graph-based induction.

However, each of these has problems. Hand-crafted preprocessing is frequently used, but is time-consuming and requires in-depth domain knowledge. Writing a custom learner is possible, but is labour-intensive. Relational learning techniques tend to be very sensitive to noise and to the particular clausal representation selected. They are typically unable to process large data sets in a reasonable time frame, and/or require the user to set limits on the search such as refinement rules (Cohen, 1995).

In this paper, we use a generic constructive induction technique to allow for domains where instances exhibit recurring substructures. For example, with Chinese character recognition, the recurring substructure is a stroke. The user defines the recurring substructures (termed metafeatures), but subsequent steps are automated. Further, our experimental results show that a small set of generic metafeatures may be applicable to many temporal domains. These substructures are extracted, and a novel clustering algorithm is used to

construct synthetic attributes based on the presence or absence of certain substructures. Standard learners can then be applied.

The learnt concepts are expressed using the same substructures identified by the user. Since these substructures are frequently the same concepts humans use themselves in classifying instances, this results in readable descriptions. To our knowledge, there are very few other systems that build classifiers for multivariate time series that are comprehensible.

There are several novel aspects to the approach. Recurring substructures are processed to construct a set of features using a specially designed clustering technique. Temporal events, global properties of the time series and specified attributes can all be combined within the well-understood propositional framework. The results of propositional learning are post-processed to generate more human-readable descriptions. The net effect of employing these techniques is a system that can easily and simply be applied to new temporal classification problem domains with little or no modifications.

This paper begins by motivating work in this field, and providing two examples, before giving an overview of both related fields and directly related work. A theoretical and practical definition of the problem, is then presented. An overview of the approach taken using a pedagogical domain is given, followed by a discussion of the implementation of *TClass*, our temporal classification learner, with several extensions to the basic idea. Experimental results using *TClass* on several domains are presented, followed by a conclusion and some suggestions for future work.

2. Motivation

2.1. Prevalence and importance of temporal classification domains

There are many real domains that are temporal in nature. An examination of the UCI repository (Blake & Merz, 1998) reveals that there are at least six domains that were originally temporal classification tasks but were "propositionalised" to make it possible to use attribute-value learners.[1]

Examples of other problems that are temporal classification tasks include: gesture recognition, printed character recognition, speaker identification and/or authentication, classification of medical time series such as electrocardiographs and electroencephalograms, robot sensor data analysis and more.

Given the importance and prevalence of these temporal classification problems, this work builds a tool that could build classifiers for these kinds of domains and apply it to them "out-of-the-box" in much the same way that a toolkit like Weka (Witten & Frank, 1999) is applied to propositional problems.

2.2. Practical examples

Consider two application domains of classification of multivariate time series (we will term this temporal classification for convenience). These two examples will provide us with some insights into the nature of the problem.

Table 1. The training set for the Tech Support domain.

	Volume level over time																						
	0										1										2		
Call	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0		Outcome
1	L	L	L	H	H	H	L	L	L	L	L	L											Happy
2	L	L	L	H	L	L	H	L	L	H	H	H	H										Angry
3	L	L	H	L	L	H	L	L	L	L	L	L	H	H	H								Angry
4	L	L	L	L	H	H	H	H	L	L	L	L	L										Happy
5	L	L	L	H	H	H	L	L	L	L													Happy
6	L	L	H	H	L	L	H	L	L	H	H	H											Angry

2.2.1. Tech support. This pedagogical domain is meant as an extremely simple example of temporal classification. A computer company called SoftCorp provides telephone technical support. These phone calls are recorded for analysis. SoftCorp discovers that the handling of these phone calls has a huge impact on future buying patterns of its customers, so based on the recordings of tech support responses, they hope to find the critical difference between happy and angry customers.

An engineer suggests that the volume level of the conversation is an indication of frustration level. SoftCorp divides each phone call into 30-second segments; and work out the average volume in each segment. If it is a high-volume conversation, it is marked as "H", while if it is at a reasonable volume level, it is labelled as "L". On some subset of their data they determine whether the tech support calls resulted in happy or angry customers by independent means. Note that conversations are not of fixed length; some conversations are short, others take a bit longer.

Six examples of recorded phone conversations are show in Table 1. SoftCorp would like to employ machine learning to find rules to predict whether, at the end of a conversation, a customer is likely to be happy or angry.

2.2.2. Recognition of signs from a sign language. Consider the task of recognising signs from a sign language[2] using instrumented gloves. The glove provides the information shown in Table 2.

Each instance is labelled with its class, and all of the values are sampled approximately 23 times a second. Each training instance consists of a sequence of measurements. Training instances differ in the number of samples; depending on the duration of the sign itself—some like `my` are short, while others like `computer` are longer.

An example training instance is shown in figure 1. This is a recording of the Auslan[3] sign `come`. The horizontal axis represents the time and the other axes represent normalised values between -1 and 1 for x, y, and z and between 0 and 1 for the remaining values.

The objective in this domain is to learn to classify unlabelled instances, given labelled recordings of different signs used for training.

Table 2. Information provided by instrumented gloves.

Channel	Description
x	Hand's position along horizontal axis (parallel with shoulders).
y	Hand's position along vertical axis (parallel with sides of body).
z	Hand's position along lateral axis (towards or away from body).
Roll	Hand's orientation along arm axis (i.e. palm facing up/down).
Thumb	Thumb bend (0 = straight, 1 = completely bent)
Fore	Forefinger bend (0 = straight, 1 = completely bent)
Middle	Middle finger bend (0 = straight, 1 = completely bent)
Ring	Ring finger bend (0 = straight, 1 = completely bent)

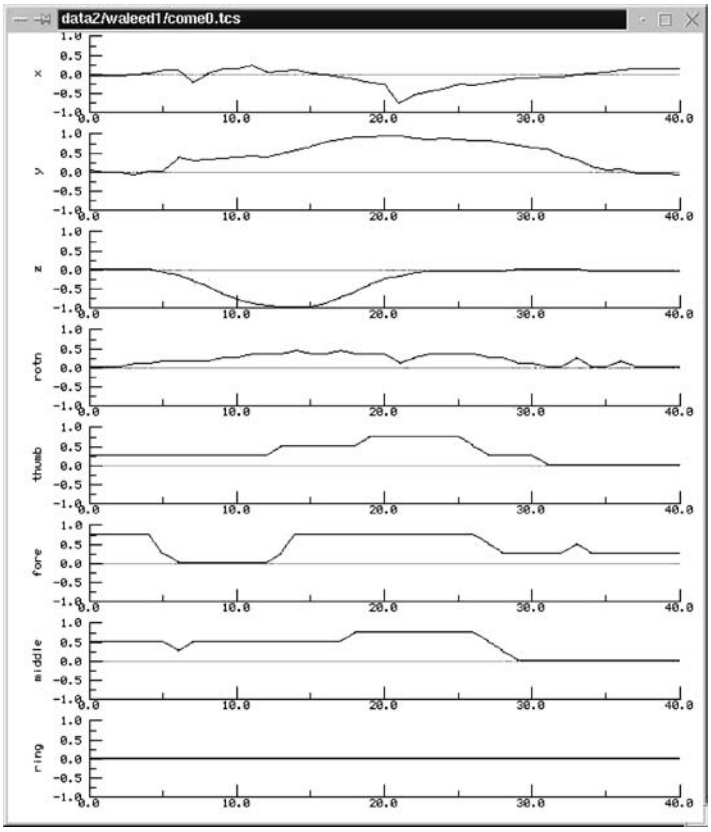

Figure 1. An example of an instance from the sign recognition domain with the label `come`.

3. Prior and related work

3.1. Relationship to other fields

While temporal classification may be relatively underexplored, time series have been studied extensively in different fields, including statistics, signal processing and control theory.

The study of time series in statistics has a long history (Box & Jenkins, 1976). The two main goals of time series analysis in statistics are (a) to characterise, describe and interpret time series; (b) to forecast future time series behaviour (Statsoft, 2002). The focus in most of the time series analysis work is on long-duration historical data (months/years), because it typically models time series from fields such as biology, economics and demographics. Approaches typically tried to model time series as an underlying "trend"—a long term pattern—together with a seasonality—short term patterns. The most popular techniques in this type of analysis are Autoregressive Integrated Moving Averages (ARIMA) and Autocorrelation.

The field of signal processing has also explored time series. The objective of signal processing is to characterise time series in such a manner as to allow transformations and modifications of them for particular purposes; e.g. optimal transmission over a phone line. One of the oldest techniques for time series analysis is the Fourier transform (Bracewell, 1965). The Fourier transform converts from a time-series representation to a frequency representation. Fourier's Theory states that any periodic time series can be represented as a sum of sinusoidal waves of different frequencies and phases. Converting from a time series representation to a frequency representation allowed certain patterns to be observed more easily. However, the limitation of periodicity is quite significant, and so recently much work has focused on wavelet analysis (Mallat, 1999), where any time series can be represented not as a sum of sinusoidal waves, but "wavelets"—non-periodic signals that approach zero in the positive and negative limits. Signal processing has seen applications in the design of electrical circuits, building of audio amplifiers, the design of telecommunications equipment and more.

Control theory is another field that has explored time series. Control theory studies systems that produce output values at regular time intervals on the basis of certain inputs. However, the relationship between outputs and inputs depends on their previous values. Control theory observes patterns in the past inputs and outputs to try to set new inputs so as to achieve a desired output. Typical applications of control theory are to operating industrial equipment such as steel mills and food production processes. Recent work has seen a move towards "adaptive control" (Goodwin, Ramage, & Caines, 1980)—approaches that modify the control theory on the basis of past observations.

All of these fields relate closely to the current work in their study of time series; however, our work differs in that its objective is not to predict future values or to modify behaviour, but rather to classify new time series based on past observations of time series, rather than analysing a single time series' patterns.

3.2. Related work

This work has a relation to the divide-and-conquer framework proposed by Dietterich (2000). Rather than the simple propositional form where instances are presented in the

form of $\langle x, y \rangle$, where x is a fixed size vector and y is a class label, Dietterich's framework allows both x and y to be complex. In contrast, the approach used in this paper allows for a complex x (such as a time series), but retains the conventional representation for y—i.e., a single class label.

This work also closely relates to the areas of feature extraction and construction (Liu & Motoda, 1998), although the formal definitions of extraction and construction assume attribute-vector representation (Liu & Motoda, 1998, p. 4). Liu and Motoda do point to automated preprocessing (or data categorization) as future work (Liu & Motoda, 1998, p. 9). They also point to the importance of comprehensibility. This work also closely relates Michalski's work (Michalski, Mitchell, & Carbonell, 1983) on constructive induction, but again the work assumes that examples are described in attribute-value format.

There are some general techniques that can be applied to temporal and structured domains. The best developed technique for temporal classification is the hidden Markov model (Rabiner, 1989). HMMs have proved themselves to be very useful for speech recognition, and are the basis of most commercial systems. However, they do suffer some serious drawbacks for general use. Firstly, the structure of the HMM—similar to that of a finite state machine—needs to be specified *a priori*. The structure selected can have a critical effect on performance. Secondly, extracting comprehensible rules from HMMs is not at all easy. Thirdly, even making some very strong assumptions about the probability model, there are frequently hundreds or thousands of parameters per HMM. As a result, many training instances are required to learn effectively. Recurrent neural networks and Kohonen maps have also been used to solve temporal and sequence problems (Bengio, 1996), but suffer similar problems.

Another approach that has a long history of use in temporal classification tasks is dynamic time warping (Myers & Rabiner, 1981). This approach can be thought of as a generalisation of nearest neighbour, but with a refined distance metric specially suited to streams (based on matching the time in one stream to the other one).

Keogh and Pazzani (2001) have worked on improving and extending dynamic time warping to temporal domains by representing the time series hierarchically. Oates, Schmill, and Cohen (2000) also use dynamic time warping to cluster experiences of mobile robots from real world data.

A more traditional artificial intelligence approach has also been taken to some of these problems. Lee and Kim (1995) take a syntactic approach to time series recognition. Based on knowledge of the financial markets, they develop a grammar for events. Mannila, Toivonen, and Verkamo (1995) have also been looking at temporal classification problems, in particular applying it to network traffic analysis. In their model, streams are a sequence of time-labelled events rather than regularly sampled channels. Learning is accomplished by trying to find sequences of events and the relevant time constraints that fit the data. Das et al. (1998) also worked on ways of extracting rules from a single channel of data by trying to extract rules of the form "A is followed by B", where A and B are events in terms of changes in value. Rosenstein and Cohen (1998) used delay portraits (a representation where the state at time $t - n$ is compared to its state at time t). These delay portraits are then clustered to create new representations, but they tend to be sensitive to variations in delay and speed. Keogh and Pazzani, in addition to the previously mentioned work on dynamic time warping, have been

working on time series classification techniques for use with large databases of time series (Keogh et al., 2001), using a representation of time series as a tree structure, with the root being the mean of the time series, the two children being the mean of the first half and second half respectively and so on. This allows very fast computations to be done on time series.

Recent interest has also arisen in applying ILP to temporal classification problems. (Rodríguez, Alonso, & Boström, 2000) is an interesting example of this, using a tailored search algorithm designed to cope with temporal constraints, although the scalability of this approach is an issue. Geurts (2001) also presents a system that uses ILP for temporal classification.

4. Formal definition of the problem

4.1. Terminology

In order to simplify the presentation some terms are defined here.

4.1.1. Channel. Consider the sign language domain discussed in Section 2.2.2. There are different sources of information, each of which we sample at each time interval—such as the x position, the thumb bend and so on. Each of these "sources" of information we term a **channel**. In figure 1, each of the sub-graphs represents a channel. The Tech Support domain contains a single channel: volume.

Formally, a channel can be defined as a function c which maps from a set of timestamps T to a set of values, V, i.e.

$$c : T \rightarrow V$$

T can be thought of as the set of times for which the value of the channel c is defined. For simplicity in this thesis, we assume that:

$$T = [0, 1, .., t_{\max}]$$

In the Tech Support domains, $t_{\max}$ varies between 9 and 14. Looking at figure 1 from the sign language domain, we can see that $t_{\max} = 40$.

The set of values V obviously depends on the classification task. In the Tech Support domain, $V = \{L, H\}$. In the sign language domain, V for the fingers is $[0, 1]$, but for the x, y and z position may be $[-1, 1]$.

4.1.2. Stream. Consider the sign language domain once again. Each training instance consists of the individual channels. It is convenient to talk of all of the channels collectively. For example, figure 1 shows a number of channels, but we want to talk about all of the channels at once. This is the definition of a **stream**.

A stream is a sequence of channels $\mathbf{s}$, such that the domain of each channel in is the same, i.e.

$$\mathbf{s} = [c_1, c_2, \ldots, c_n] \text{ s.t. } \text{domain}(c_1) = \text{domain}(c_2) = \cdots = \text{domain}(c_n)$$

where n is the number of channels in the stream **s**. Each channel has the same domain—this corresponds to the timepoints where samples of each of the channels exist.

A stream allows us to collect all these various measurements and treat them, in some ways, as part of the same object. The Tech Support domain is very simple: each training stream has one channel. However, the Auslan domain has eight.

4.1.3. Frame. It is also convenient at times to refer to the values of all channels at a given point in time. Formally, a frame **fr** can be defined as a function on a stream **s** and a time t as:

$$\mathbf{fr}(\mathbf{s}, t) = [c_1(t), c_2(t), \ldots, c_n(t)]$$

In figure 1, this can be thought of a "vertical slice" of the time series.

4.2. Statement of the problem

4.2.1. Goal. Let S be the set of all possible streams in a domain, and let L be a set of class labels for the domain. Further, let d be a function that maps from a given stream to a class label; i.e. $d : S \rightarrow L$. Given a subset of d, say d_T; the goal is to produce a function d_P that is as similar to d as possible.

Similarity in this case, is defined as follows: let $P(\mathbf{s})$ be the probability that a stream **s** occurs in a particular problem domain. Furthermore define a function $cost(i, j)$.

$$\begin{aligned} cost(i, j) &= 0, \quad \text{if } i = j \\ &= 1, \quad \text{otherwise} \end{aligned}$$

Then the temporal classification task can be defined as finding the d_P that minimises:

$$\sum_{\mathbf{s} \in S} cost(d(\mathbf{s}), d_P(\mathbf{s}))\ P(\mathbf{s})$$

Informally, we have several other important goals. The most important of these is comprehensibility. It would be advantageous in many domains if d_P was presented in a way that was easily understood. Another important goal is that d_P should be produced in a timely and space efficient manner.

4.2.2. Strong temporal classification. An obvious generalisation of temporal classification outlined above is to domains where a single stream is not associated with one class label, but with a sequence of class labels; i.e. instead of $d : S \rightarrow L$, $d : S \rightarrow L^+$ (where L^+ is the set of all non-null sequences of elements of L). This complicates the classification task significantly, and in the current work, we consider only the weaker form of temporal classification; with strong temporal classification examined in future work.

5. Approach

The approach we take in this work is to take advantage of the recurring substructure that many temporal classification problems exhibit.

5.1. Outline

Let us revisit the Tech Support domain discussed in Section 2.2.1. One expert advises that "runs" of high volume conversation—continuous periods where the conversation runs at a high volume level—are important for classification. Runs of loud volume could be represented as a tuple (t, d) consisting of:

– The time at which the conversation becomes loud (t).
– How long it remains loud (d).

This is our first **metafeature**, called LoudRun.

Each instance can now be characterised as having a set of LoudRun events—the LoudRun events are the recurrent substructures appropriate for this domain. These can be extracted by looking for sequences of high-volume conversation. For example, call 2 has one run of highs starting at time 3 lasting for 1 timestep, a high run starting at time 6 lasting for one timestep and a high run starting at time 9 for 4 timesteps. Hence the LoudRuns extracted from call 2 are {(3, 1), (6, 1), (9, 4)}.

To take advantage of attribute-value learners, these sets must be converted into propositional form. A good hypothesis language for such domains consists of rules that check for combinations of particular kinds of events that are critical for classification. For example, in the sign language domain, an upwards motion early in the sign *and* a movement of the hand forward later in the sign *without* a closed hand means `thank`. Thus we break the learning into two stages: the first to pick out the prototypical instances of an event—in this case: the upwards motion, the movement of the hand forward and the closed hand; and the second to create rules using the prototypical instances. To accomplish the first task, we use a clustering technique similar to prototype learning, and for the second we use an attribute-value learner.

To complete the first stage, the events extracted above can be plotted in the two-dimensional space shown in figure 2. This is the **parameter space**. This two-dimensional space consists of one axis for the start time and another for the duration.

Once the points are in parameter space, "typical examples" of LoudRuns can be selected. In this case, the points labelled A, B and C have been selected,[4] as shown in figure 3. These are termed **synthetic events**. They may or may not be the same as an observed event—so for example, point A actually corresponds to a real event (the event (3, 3) actually was observed in the data), whereas B and C do not.

These synthetic events can be used to segment the parameter space into different regions by computing the Voronoi tiling: for each point in the parameter space, the nearest synthetic event is found. The set of points associated with each synthetic event form a region and the boundaries of each region are calculated. These are shown as dotted lines in figure 3.

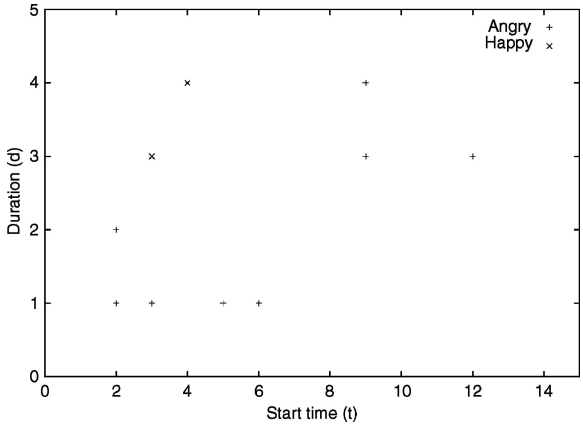

Figure 2. Parameter space for the LoudRun metafeature in the Tech Support domain.

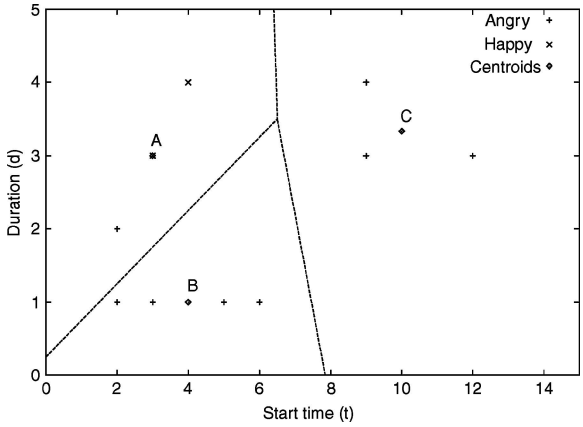

Figure 3. Three synthetic events and the regions around them for LoudRuns in the Tech Support domain.

To accomplish the second stage, each region constructed in the first stage is used as a test: if a training instance has an event in a region, it "passes" the test. Each training instance is labelled with the test results, each test result effectively becomes an attribute. In this case, three tests A, B and C are constructed and the results shown in Table 4. To construct this table, Table 3 is examined, and for each region if there is an event that lies within it, a **synthetic attribute** corresponding to the point is marked as a "yes".

As a result of the clustering stages above, the initial time series has been converted into a table suitable for learning. In fact, if it is fed it to C4.5, the simple tree[5] in figure 4 results.

This tree says that if the training instance has an event that lies within in region C (i.e. a run of high values that starts around time $t = 10$ and goes for approximately 3.33 timesteps),

Table 3. Observed LoudRun events for the Tech Support domain.

Call	Observed events
1	{(3, 3)}
2	{(3, 1), (6, 1), (9, 4)}
3	{(2, 1), (5, 1), (12, 3)}
4	{(4, 4)}
5	{(3, 3)}
6	{(2, 2), (6, 1), (9, 3)}

```
rgnC = yes: Angry (3.0)
rgnC = no: Happy (3.0)
```

Figure 4. Rule for telling happy and angry customers apart.

then its class is Angry. In other words, as long as there is not a long high-volume run towards the end of the conversation, the customer is likely to be happy.

6. Implementation

While the idea of metafeatures is simple, there are several possible enhancements and features that need to be considered when implementing a practical system. The *TClass* algorithm is shown in figure 5. Explanation of each of the components follows.

6.1. Input format

The input is specified as a set of streams S_{train}. As previously mentioned, each stream represents a single training instance. For example, a single sign is one stream. In addition, for training purposes we are given a set of class labels.

6.2. Specified global attributes and global attribute calculators

In addition to temporal characteristics, many domains have non-temporal attributes that are important for classification. For example, consider diagnosing ECGs, as discussed in Section 7.5. Obviously there are important temporal characteristics of the ECG, but there are also other attributes which are important to classification, such as the patient's age, height, weight and gender. *TClass* allows the integration of conventional features, aggregate global features and temporal features, while other systems (such as dynamic time warping and hidden Markov models) do not.

Specified global attributes may therefore be provided to *TClass a priori*, with one vector of attributes for each training instance.

```
Inputs:
    S_train = [s_1, ..., s_n] /* Training streams */
    G_train = [g_1, ..., g_n] /* Specified global attributes */
    L_train = [l_1, ..., l_n] /* Class labels of training streams */
    f = [f_1, ..., f_m] /* Metafeature extraction functions */
    w = [w_1, ..., w_p] /* Global attribute calculators */
Outputs:
    E = [e_1, e_2, ..., e_k] /* Synthetic events */
    d_P /* Learnt classifier */
    h /* Human-readable description of learnt concept */

Temporary:
    O_train = [o_11, ..., o_nm]
        /* Observed events*/
    A_train = [a_1, ..., a_n]
        /* Values that result from global attribute calculators */
    I_train = [i_1, ..., i_n] /* Synthetic attributes */
    B_train = [b_1, ..., b_n]
        /* Combination of all attributes from all sources */

procedure Train
    O_train := ExtractObservedEvents(S_train, f)
    E := SelectSyntheticEvents(O_train, L_train)
    I := EvaluateSyntheticAttributes(O_train, E)
    A := CalculateGlobalAttributes(S_train, w)
    B_train := CombineAttributes(I_train, A_train, G_train)
    d_P := Learn(B_train, L)
    h := ProduceComprehensibleDescription(d_P, E, O_train)
End
```

Figure 5. *TClass* training algorithm.

It is also useful in temporal classification tasks to examine aggregate values of signals and use them as propositional attributes. For example, for continuous channels, the global maximum and minimum value of a channel, or the mean of each channel may be important attributes. Such attributes are examples of **aggregate global attributes**—they measure some property of each training instance as a whole, rather than looking at the temporal structure. Such features can be surprisingly useful for classification. For example (Kadous, 1995) shows that in the sign language domain almost 30 per cent accuracy can be obtained in classifying 95 signs based solely on the maxima and minima of the x, y, and z positions of the right hand.

To support the use of such aggregate features, *TClass* supports the use of global attribute calculators. For example, one very commonly used global attribute calculator is the mean of a given channel. Another commonly used global attribute calculator is duration, since duration is a good indicator of class on many problems.

In *TClass* the global attribute calculators are provided a vector of functions, each function acting on each training stream. The return value of each calculator is a single attribute which can be appended to the feature vector.

6.3. *Metafeatures*

Metafeatures form the most critical component of *TClass*. The formal definition of a metafeature is very simple. It consists of:

- A tuple of parameters $p = (p_1, \ldots, p_k)$, which represents a particular *event*. Let P_1 represent the set of possible values of the parameter p_1. Let P be the space of possible parameter values, i.e. $P = P_1 \times P_2 \times \ldots \times P_k$. P is termed the *parameter space*.
- An *extraction function* $f : S \rightarrow \mathbb{P}(P)$ which takes a stream **s** and returns a set of observed events from P.

For example, each of the entries in Table 3 is an event for the metafeature LoudRun, and figure 2 shows the parameter space. A simplified version of the extraction function is:

$$\begin{aligned} f(\mathbf{s}) = \{(t, d) \mid & v(t-1) = L \\ & \wedge v(t) \ldots v(t+d-1) = H \\ & \wedge v(t+d) = L\} \end{aligned}$$

In this way, metafeatures capture a type of recurring temporal substructure, which can then be used as the basis of constructing a propositional feature learner.

6.3.1. Good metafeature design. Designing metafeatures is not trivial. For example, consider designing a metafeature for periods of increase in a single channel. There are several possible representations of such an event. For example: (start time, end time, start value, end value). However, another representation is possible: (midtime, duration, average value, gradient). Both of these representations give us information about the underlying features, and it is easy to convert from one to the other. What then, is the appropriate representation? The following characteristics are desirable in a metafeature:

- *Robust to noise*. In the above example, the start value and the end value are both sensitive to noise, as compared to, say average and gradient, both of which are aggregate measures of the data points belonging to the period of increase. Hence it would be better to represent an increasing event using the latter two parameters rather than the former. In general, an aggregate of the signal will be more robust than a single measurement of the signal, so average value and gradient are more robust than start value and end value.
- *Temporal variations are mapped to proximal parameter values.* Careful consideration reveals that metafeatures should capture the temporal variation as parameter values, and in so doing, allow temporal variation to be handled. For instance, if an Increasing event occurred slightly later than expected, then the difference is mapped into a difference of the midtime parameter; and the greater the difference in the midtime, the greater the difference in that parameter's value. Hence, the temporal variation between the two cases is mapped into proximal parameter values, where the usual distance metrics still apply. In terms of the parameter space, this means that two observed events that occur at

approximately the same time or last for approximately the same duration should be close in parameter space.
- *Orthogonality*. If we were to represent an increasing event by start time, end time and duration, then this would be redundant. Any two of these would be sufficient to uniquely represent such an event. Fewer parameters reduce processing time of the observed events; furthermore, some learning and clustering algorithms applied assume that attributes are independent. This may be an appropriate approximation at times, but if there is a known and clearly defined dependence (e.g., in this case $duration = endtime - starttime$) it might affect the performance of the algorithm.
- *Captures the important characteristics of the event.* For example, the gradient of the increasing interval is an important characteristic for sign language: the speed of movement upwards indicates rapidity of movement—e.g. the distinction between shutting a window (a slow movement of one forearm downwards against the other forearm) and slamming a window (a much more rapid movement). On the other hand, the variance of the signal may not be important to us, as it may be, for example, the result of noise in our measurement system, rather than some important underlying property of the data.
- *Matches the "human" description of the event.* Humans would probably, if observing a signer move his hand up and down, be interested in the general location of the movement, the speed of the movement, the duration of the movement and how it temporally relates to other events. This will allow us to "reconstruct" the learnt concept in a manner more readable by a person.

6.3.2. A universal set of metafeatures for temporal domains. It may be possible to find a generic set of metafeatures that are applicable across a wide variety of temporal domains. This work therefore proposes such a generic set of metafeatures, designed to be general enough for many domains.

6.3.2.1. Increasing and decreasing. The Increasing metafeature detects when a continuous signal is increasing. It operates on continuous values only. The output from Increasing is a list of tuples consisting of four parameters:

- *midTime*. The middle point temporally of the event.
- *average*. The average value of the event. This value is expressed in terms of the data type of the channel.
- *gradient*. This is the rate of change of the channel. To be exact, a line of best fit is fitted to the increasing interval, and the gradient of the line is calculated.
- *duration*. This is the length of the increasing interval.

Increasing operates on a single channel. It accepts a number of settings that modify its robustness to noise, whether time is expressed in absolute or relative terms and which channel to extract the events from.

Decreasing is identical to Increasing in almost every way, except it tries to isolate intervals of decrease rather than increase.

6.3.2.2. Plateau. The Plateau metafeature detects when a continuous signal is not changing, neglecting noise. The output from Plateau is a list of observed consisting of three parameters, basically identical to the Increasing metafeature but without a gradient.

6.3.2.3. LocalMax and LocalMin. The LocalMax metafeature detects when a signal has a local maximum. It should not be confused with the global maximum and minimum functions that operate on the whole channel. It produces observed events consisting of the following parameters:

– *time*. This is as for the **midTime** parameter for the Increasing metafeature.
– *value*. The height of the maximum.

LocalMin is identical in most respects to LocalMax, except that it detects local minima.

6.3.2.4. Other metafeatures. Additional metafeatures can be easily implemented, usually in less than 200 lines of code. To add metafeatures, they must implement the interface of the Java class `MetafeatureI`, which has a total of 8 methods. Most of these are housekeeping functions (e.g. `String name()` to return a metafeature's name). The most important is the `ObservedEventVec extractEvents(Stream s)` that applies the metafeature extraction function to a training stream and returns a vector of observed events.

6.4. Extracting observed events

Once the metafeatures have been selected, the extraction of observed events is accomplished by applying each metafeature to each training instance. For each metafeature and training instance, a list of the observed events is stored for the the later synthetic event selection. It is also retained for use in attribution.

6.5. Selection of synthetic events

Now that a list of all events and their observed events have been constructed, the synthetic events must be selected. The key insight to selecting these synthetic events is to assume the distribution of observed events in the parameter space is probably not uniform.

An initial approach might be to use normal clustering in the parameter space, i.e., clustering that groups points in such a way that the distance between points within the same cluster is small, and distances between points in different clusters is large. However, this is merely one approach to answering the real question that we are interested in: "Which are the observed events whose presence or absence indicate that the instances belong to a particular class?" The clustering approach can certainly be used to answer this question—the theory being that the different clusters represent different types of observed events. This is an example of *undirected segmentation*.

However, a more refined approach is possible: one that takes into account the *class* of the observed events. One way to do this is to augment the parameter space with the class as an extra dimension. However, this would likely not work either, as the distance metric between

classes is not obvious; and the weight given to the class label relative to other features in the parameter space is not easy to determine. Likely, such an algorithm would be caught between ignoring class completely, leading to traditional clustering, and paying too much attention to the class, leading to too many clusters.

Another approach is to re-pose the question as: "Are there regions in the parameter space where the class distribution is significantly different to what is statistically expected?" This question is very similar to typical problems of supervised classification. In particular, if we pose the question as "Are there attribute-value pairs for which the class distribution is significantly different to what we would expect?", then this is the same question that is the basis of many decision tree and rule induction systems. This is an example of *directed segmentation*: segmentation directed towards creating good features for learning.

The solution used in this work is as follows: We have to select a group of synthetic events to query on, and we have to divide the space into regions. Hence, one approach is to select a number of observed events. Let the set of all observed events be O. Let the set of points we select be $E = \{e_1, e_2, \ldots, c_k\}$, where there are k points selected. Define the set of regions $R = \{R_1, \ldots, R_k\}$ as:

$$R_i = \{x \in P \mid closest(x, E) = e_i\}$$

and

$$\text{closest}(x, E) = \underset{e \in E}{\text{argmin}}\, dist(x, e)$$

where *dist* is the distance metric we are using for the parameter space.

In other words, each region R_i is defined as the set of points in the parameter space P for which c_i is the closest point (using the distance metric *dist*) in the set E. This is the definition of a Voronoi diagram. A good review of Voronoi diagrams and their (many) applications can be found in Aurenhammer and Klein (2000).

If these regions were selected randomly, it would be expected that the class distribution in each region would be similar to the global class distribution. However, if the distribution of observed events differed significantly from the global class distribution, then this would be interesting and useful for a machine learning algorithm. Asking whether a training stream has an observed event within a region would be informative of the class of the training stream that the observed event came from.

In general, the objective is to find the subset of the observed events that would give us the most "different" class distribution for each region. This would then give something for the learner in the subsequent stage to use for the basis for learning. However, one might then suggest that the solution to this problem is simple: make $E = O$—in other words, make one synthetic event for each observed event. Each region will then have exactly one observed event in it, with a very non-random class distribution. Hence we need a measure that "trades off" the number of regions for size. We will term this measure the *disparity measure*. The disparity measure measures the difference between the expected and observed class distribution. For now, let us assume that we have such a measure, say, called *DispMeas*.

Then we are looking to find:

$$R = \underset{E \in \mathbb{P}(O)}{\operatorname{argmax}} \, DispMeas(E)$$

In other words, we are looking to find the subset of O (the set of all observed events), E, for which the disparity measure is the greatest.

The difficulty in the above equation is the $\mathbb{P}(O)$. In general, if $n = \|O\|$, then there are 2^n elements in $\mathbb{P}(O)$. If we wanted to do an exhaustive search, we would therefore have to look at a number of subsets that was exponential in the number of observed events. For example, if there are only 100 elements in I, we would still have to check more than 10^{30} possible sets. Even if we were to constrain it to a reasonable size, say no more than 10 regions, E is still combinatorially large in O.

However, it might be possible to search part of the feature space and come up with a reasonable set E that is effective in constructing features.

6.5.1. Disparity measures. Disparity measures are a recurring theme in supervised learning. For example, in decision-tree building algorithms like C4.5 (Quinlan, 1993), the algorithm proceeds by dividing the instances into two or more regions, based on attribute values. The disparity measure used by C4.5 is the gain ratio, however there are several other possible disparity measures.

In fact there is a whole area of study of different disparity measures. White and Liu (1994) provide an extensive survey of disparity measures. There are many similarities between decision tree induction (which is built on segmentation based on attribute values) and the creation of regions around centroids (which is built on segmentation based on a Euclidean distance measure).

Another approach, this one from statistics, is the chi-square test. The chi-square test measures the difference between the expected values in each of the cells in the contingency table. By comparing this against a chi-square statistic, a measure of the probability that the distribution of observed events we see in the contingency table is random. The smaller that this probability is, the less likely it is that the distribution is due to chance. This probability is called the *power* of the test.

The χ^2 statistic can be computed as:

$$\chi^2 = \sum_i \sum_j \frac{(Exp_{ij} - Obs_{ij})^2}{Exp_{ij}}$$

where Obs_{ij} is the observed number of observed events belonging to region R_j in class L_i and Exp_{ij} is the number of observed events that we would expect if the region was independent of the class.

Once we have computed the χ^2 statistic, we can compute the probability from the definition of the χ^2 distribution, the probability this particular contingency table was the result of random chance. The smaller that this probability is, the more confident we can be that the distribution is likely to be useful for discriminative purposes.

```
Inputs:
        O = {o_1, ..., o_k} /* Observed events */
        L = {l_1, ..., l_k} /* Class labels of observed events */
        N = {n_min, ..., n_max} /* Number of synthetic events */
        DispMeas(SynthEvents, Instances, Labels) /* Disparity measure */
        numTrials /* number of trials to attempt */
Outputs:
        E = {e_1, ..., e_n} /* Synthetic events */

procedure RandSearch
        bestMeasure := 0
        bestSynthEvents := {}
        for i := 1 to numTrials
                r := randBetween(n_min, n_max)
                currentE := randSubset(r, O)
                currentMetric := DispMeas(currentE, O, L)
                if(currentMetric > bestMetric)
                        bestMetric := currentMetric
                        bestSynthEvents := currentE
                End
        End
E := bestEvents
End
```

Figure 6. Random search algorithm.

Unlike information gain, the χ^2 distribution does not suffer from the same issues of bias towards more regions, at least theoretically. However, it is more difficult and time-consuming to calculate than the information gain or gain ratio (since to compute the probability mentioned above requires computing the integral of the probability density function of the of χ^2 function). In *TClass*, the χ^2 is used by default.

However, we still do not have a search algorithm to find the set of points.

A "random search" can be employed to solve the problem, as shown in figure 6. While it may first seem that using a random search algorithm is not productive, work by Ho, Sreenivas, and Vakili (1992) in the field of ordinal optimisation shows that random search is an effective means of finding near-optimal solution. This was also used by Srinivarsan (2000) in his *Aleph* ILP system where it was shown to perform well even compared to complex search methods.

Ordinal optimisation can be applied to our problem as follows. Let us assume that our objective is to find a subset of points that is in the top k per cent of possible subsets (i.e. whose disparity measure ranks it in the top k per cent). If we test n possible subsets randomly, then the probability P of getting an instance in the top k per cent can be shown to be:

$$\begin{aligned} P(\text{1 or more subsets in top } k\%) &= 1 - P(\text{0 subsets in top } k\ \%) \\ &= 1 - (1-k)^n \end{aligned}$$

For example, if $n = 1000$ and $k = 1\%$ then the probability of finding an instance in the top 1 per cent of possible cases is $1 - (1 - 0.01)^{1000} = .999957$. In other words, if we

try 1000 instances, then there is a better than 99.995 per cent chance that we will get one instance in the top 1 percent. Hence by softening the goal (trying to find *one of the best* rather than *the best* subset of points) we can be highly confident of getting a solution in the top percentile.

An algorithm for random search is shown in figure 6. It assumes the following auxiliary functions:

- randBetween(min, max) returns a random number between *min* and *max* inclusive.
- randSubset(num, set) returns a random subset of *set* with *num* elements in it, but with no repetitions.

Also note that *DispMeas* takes a set of synthetic events, a set of points and their class labels. Note that the random search algorithm is highly parallelisable. The main `for` loop can be run in parallel across many different processors and/or computers, provided that the final `if` statement that updates the best instance seen so far is done atomically. This is a significant advantage over the K-Means algorithm, which is not easy to parallelise at all, as it is inherently iterative.

6.6. *Evaluation of synthetic attributes*

We now have a set of synthetic events for each metafeature. Each synthetic event from each metafeature is converted into a synthetic attribute. Each attribute is a test based on whether a given training instance has an observed event that lies within the region around that particular synthetic event.

In Table 4, the attributes generated are binary—i.e., we are checking for the presence of particular events. However, this gives a very "crisp" decision boundary. It may be more useful to have a richer measure of membership. We use the measure $D = \log_2(\frac{d_2}{d_1})$, where d_1 is the distance to the nearest centroid and d_2 is the distance to the second nearest centroid. This measure has useful properties; for instance, a point on the boundary between two regions has $D = 0$, whereas the centroid has a measure of $D = \infty$. This expands the hypothesis language significantly and makes the classification more robust.

Table 4. Attribution of synthetic attributes for the Tech Support domain.

		Synth Attrib		
Stream	Class	A	B	C
1	Happy	Yes	Yes	No
2	Angry	No	Yes	Yes
3	Angry	No	Yes	Yes
4	Happy	Yes	Yes	No
5	Happy	Yes	Yes	No
6	Angry	Yes	Yes	Yes

6.7. Combining attributes and learning

Attributes are then combined for each instance from the three sources: synthetic attributes, specified global attributes, and calculated global attributes.

With this in place, an attribute-value learner can be applied. *TClass* employs Weka (Witten & Frank, 1999) as a backend learner, and almost any learner can be used. Furthermore, both bagging (Breiman, 1996) and boosting (Schapire, 1999) can also be used.

There is a further possibility resulting from the random nature of the synthetic event selection algorithm. Each time the synthetic event selection algorithm runs, it results in a different set of synthetic attributes. Multiple runs of synthetic event selection and the subsequent evaluation and learning stages can be used to create an ensemble, resulting in improved accuracy, but at the cost of incomprehensibility.

6.8. Producing comprehensible descriptions

If the learner used at the backend produces descriptions of the form `rgnC = yes`, then these can be used to create comprehensible descriptions by substituting the synthetic event in place of the attribute name. Hence figure 4 can be easily transformed into figure 7 by substituting the parameters of C.

However, figure 7 is lacking in one regard: what does "approximately" mean? Does timestep 10 mean between 9 and 11 or between 5 and 15? We can give an approximation of the bounds on these values by drawing a bounding box in the original parameter space of all the instances belonging to region C. Looking at figure 3, we see that all the points in region C lie within the bounding box $d = [3, 4], t = [9, 12]$. Hence the rule in figure 7 can be rewritten more clearly as shown in figure 8. Note that this is not the same concept that the classifier uses on unseen instances, but it is still useful as an approximation. An obvious modification of this approach allows it to be used with relative membership.

```
Does have a Loud run
   approx starting from timestep 10 AND
   approx lasting for 3.33 timesteps: Angry (3.0)
Otherwise: Happy (3.0)
```

Figure 7. Human readable form of rules in figure 4.

```
Does have a Loud run
   starting between time 9 and 12 AND
   lasting between 3 and 4 timesteps: Angry (3.0)
Otherwise: Happy (3.0)
```

Figure 8. Adding bounds to figure 7.

Inputs:
$S_{test} = [\mathbf{s_1}, ..., \mathbf{s_n}]$ /* Test streams */
$G_{test} = [\mathbf{g_1}, ..., \mathbf{g_n}]$ /* Specified global attributes */
$\mathbf{f} = [f_1, ..., f_m]$ /* Metafeature extraction functions */
$\mathbf{w} = [w_1, ..., w_p]$ /* Global attribute calculators */
$E = [e_1, e_2, ..., e_k]$ /* Synthetic events from training */
d_P /* Learnt classifier from training */

Outputs:
$L_{test} = [l_1, ..., l_n]$ /* Test set labels */

Temporary:
$O_{test} = [\mathbf{o_{11}}, ..., \mathbf{o_{nm}}]$ /* Observed events */
$A_{test} = [\mathbf{a_1}, ..., \mathbf{a_n}]$ /* Global attribute calculated */
$I_{test} = [\mathbf{i_1}, ..., \mathbf{i_n}]$ /* Synthetic attributes */
$B_{test} = [\mathbf{b_1}, ..., \mathbf{b_n}]$ /* Combination of all attributes from all sources */

procedure **Test**
$O_{test} :=$ ExtractObservedEvents$(S_{test}, \mathbf{f})$
$I :=$ EvaluateSyntheticAttributes(O_{test}, E)
$A :=$ ExtractGlobalAttributes$(S_{test}, \mathbf{w})$
$B_{test} :=$ CombineAttributes$(I_{test}, A_{test}, G_{test})$
$L_{test} :=$ Classify(B_{test}, d_P)
End

Figure 9. *TClass* testing algorithm.

6.9. Testing

The testing algorithm employed by *TClass* is shown in figure 9. As can be seen it reuses many of the same components used from training. Note that the global attribute calculators and metafeature extraction functions must be the same for both training and testing.

As before, the observed events are extracted, but the synthetic events selected in the training stage are used to create the synthetic attributes. Once the attributes are combined, the classifier built in the training stage can be employed to give a classification.

7. Experimental results

TClass was tested on two artificial and two real-world domains. In order to assess error rates and similar details, we first outline the nature of the tests we performed.

To measure accuracy, single runs of cross validation were used (either 5-fold or 10-fold depending on the data set). Stratified cross-validation was considered, however, the quantity of the data and the processing time are such that this would have resulted in overly long computation times.

TClass relies on a propositional learner at the backend. We consider four propositional learners: J48 (the Weka equivalent of `c4.5`), PART (the Weka equivalent of `c4.5rules`), IB1 (nearest neighbour algorithm) and Naive Bayes (with the Laplace adjustment to prevent zero probabilities and Fayyad and Irani (1993) based discretisation).

AdaBoost and Bagging can be incorporated as part of the propositional learner—that is within the *TClass* framework. Both AdaBoost and Bagging were applied, using J48 as the base learner.

7.1. Baseline algorithms

In order to evaluate the effectiveness of our learners, it is useful to compare the performance of *TClass* against other temporal learners. We consider:

7.1.1. Naive segmentation. One approach that has been previously explored and employed (e.g., Kadous, 1995) is naive segment-based feature extraction. This can be thought of as downsampling to a fixed size. Each channel is divided into n segments temporally. For example, if $n = 5$ and the stream is 35 frames long, then the first 7 frames of each channel will be "binned" in first segment, the second 7 in the next segment and so on. The mean of each segment is calculated, and this becomes an attribute.

Hence, if there are c channels, this will generate nc features. These can be used as input to an attribute-value learner.[6] Four possible values of n: 3, 5, 10, 20 will be considered. In the following experiments, we also consider several different learners for the segmented attributes: J48, PART, IB1, AdaBoost with J48 and bagging with J48. Unless otherwise indicated the *best* result amongst the different learner-segment (there are 20 possible) combinations are presented. This does place a bias in the results towards the segmented learner, since it has more opportunities to fit the test set.

7.1.2. Hidden Markov models. We employed the implementation of hidden Markov models provided by HTK (Young et al., 1998). Each class is modelled using the same topology. The following number of states are considered: 3, 5, 10, 20; and the following topologies: left-right, left-right with 1 skip allowed and ergodic. Each state in a left-right HMM allows transitions only to itself or the next state—hence each state must be visited. to A 1-skip model is a left-right model that also allows the transition from state i to state $i+2$, effectively allowing to skip certain states. Ergodic HMMs allow all possible transitions. HMMs that include the raw data as well as both the raw data and the first derivative are considered. As with the naive segmenter, unless otherwise indicated the *best* results, over all the topologies, states and both raw data and raw data with first derivative (a total of 32 possibilities) were reported. It should be noted this is biased in favour of hidden Markov models.

7.2. Artificial datasets

We examined the performance of *TClass* on two artificial datasets. These are useful because:

- Evaluation is simplified, particularly in relation to comprehensibility. For example, the induced concept can be compared with the correct concept. It can also be used to explore factors of interest such as noise, etc.
- It allows us compare our results with other researchers.

7.2.1. Cylinder-bell-funnel. The artificial cylinder-bell-funnel task was originally proposed by Saito (1994), and further worked on by Manganaris (1997). The task is to classify a stream as one of three classes, cylinder (c), bell (b) or funnel (f). Samples are generated as follows:

$$
\begin{aligned}
c(t) &= (6+\eta)\cdot\chi_{[a,b]}(t)+\epsilon(t)\\
b(t) &= (6+\eta)\cdot\chi_{[a,b]}(t)\cdot(t-a)/(b-a)+\epsilon(t)\\
f(t) &= (6+\eta)\cdot\chi_{[a,b]}(t)\cdot(b-t)/(b-a)+\epsilon(t)\\
\chi_{[a,b]} &= \begin{cases} 0 & t<a,\\ 1 & a\le t\le b\\ 0 & t>b \end{cases}
\end{aligned}
$$

where η and $\epsilon(t)$ are drawn from a standard normal distribution $N(0, 1)$, a is an integer drawn uniformly from [16, 32] and $b-a$ is an integer drawn uniformly from [32, 96]. The cylinder class is characterised by a plateau from time a to b, the bell class by a gradual increase from a to b followed by a sudden decline and the funnel class by a sudden increase at a and a gradual decrease until b. Although univariate (i.e., only has one channel) and of a fixed length (128 frames), the CBF task attempts to characterise some of the typical properties of temporal domains. Firstly, there is random amplitude variation as a result of the η in the equation. Secondly, there is random noise (represented by the $\epsilon(t)$). Thirdly, there is significant temporal variation in both the start of events (since a can vary from 16 to 32) and the duration of events (since $b-a$ can vary from 32 to 96).

266 instances of each class were generated. In all tables, the bounds represent the standard error of the mean (the standard deviation divided by the number of of folds).

The results, shown in Table 5, were very surprising. One surprise was the performance of the baseline algorithms: nowhere else in the literature has their performance been compared. It is surprising to see them perform amongst the best and even perform better than previously published results.

Table 5. Error rates using the current version of *TClass* on the CBF domain.

Approach	Error
TClass with J48	2.28 ± 0.69
TClass with PART	4.56 ± 0.82
TClass with IBL	56 ± 1.24
TClass with Bagging/J48	1.9 ± 0.48
TClass with AdaBoost/J48	1.39 ± 0.33
TClass with Naive Bayes	3.16 ± 0.70
Voted TClass (3 runs) with AdaBoost/J48	$\mathbf{0 \pm 0}$
Naive segmentation	$\mathbf{0 \pm 0}$
Hidden markov model	$\mathbf{0 \pm 0}$

Thus, this may be too simple to test the capabilities of our system. This makes it difficult to evaluate accuracy and other phenomena. Hence we need to construct a dataset that is (a) more difficult and (b) more typical of the domains that we are interested in.

Let us suggest why the two controls performed so well—they capture the natural aggregate nature of the data. There is only one real event in each of the three classes; and all of these events are of extended duration—at a bare minimum the one event characterises itself over 25 per cent of the data (32 frames) and possibly as much as 75 per cent of the data (96 frames). On average, the one event therefore occupies 50 per cent of the channel. Although the underlying signal is masked by the noise, the noise has a Gaussian distribution, exactly the type of noise that aggregate approaches (such as naive segmentation and HMMs) are designed to exclude. Further investigation revealed an interesting result in the HMMs: the self-transition probability for each state was almost always approximately 0.8. This means the hidden Markov model spends approximately the same time in each state. Some simple mathematics show that if $a_{ii} = 0.8$ the HMM stays in each state for about 7 time steps, which is very close to the size of a segment used by the naive segmenter (6.25 timesteps). It seems that the HMM is defaulting to a naive segmentation approach: equal size regions, with a mean and an average for each region functioning as a probabilistic model.

7.2.2. Comprehensibility. Are the results produced using *TClass* comprehensible? In this particular case, we can compare the induced classifier to the real definitions.

Figure 10 shows a typical ruleset created by *TClass* with PART. The first rule for instance checks for a high local minimum around time 44, which should only be possible if it is a cylinder or a funnel; and failing this, looks for a nice gentle decreasing signal that is

```
IF c HAS LocalMin: time = 44.0 val = 5.76 AND
IF c HAS NO Decreasing: midTime = 59.0 avg = 1.98 m = -0.09 d = 53.0
 THEN cyl (239.0/1.0)

IF c HAS NO Increasing: midTime = 23.5 avg = 1.95 m = 0.29 d = 18.0 AND
IF c HAS NO Decreasing: midTime = 59.0 avg = 1.98 m = -0.09 d = 53.0
 THEN bell (233.0)

IF c HAS NO LocalMax: time = 72.0 val = 6.14 AND
IF c HAS NO LocalMax: time = 65.0 val = -0.36
 THEN funnel (228.0)

IF c HAS NO Increasing: midTime = 59.0 avg = 3.93 m = 0.21 d = 11.0 AND
IF c HAS LocalMax: time = 27.0 val = 4.82
 THEN funnel (9.0)

IF c HAS NO Increasing: midTime = 59.0 avg = 3.93 m = 0.21 d = 11.0
 THEN cyl (7.0)

: bell (3.0)
```

Figure 10. A ruleset produced by *TClass* using PART as the learner (bounds have been omitted to save space).

characteristic of the funnel. If this is not present it is a cylinder. If it is not a cylinder, then it looks for a sudden increase early on in the signal (typical of cylinders or funnels) or a gradual decrease (typical of funnel). If it does not have either of these, then it is a bell. To finally ensure it is a funnel, it checks to see if there is either a middle time high maximum (more characteristic of a cylinder) or a middle time low maximum (only possible with a bell). If it has neither of these it must be a funnel. Between them, these three rules cover 88 percent of instances.

7.2.3. Conclusions. The CBF dataset turns out to be largely trivial as a learning problem. All three approaches, *TClass*, naive segmentation and hidden Markov models attain 100 per cent accuracy.

This makes the CBF domain a poor one for evaluation of some of the other issues we are interested in since it does not give good guidance to the real-world performance of the classifier.

We claim subjectively that the *TClass* definitions are easy to understand, as they are expressed in terms that a human looking at the data might use: the height of maxima, gentle changes in gradient, and sudden changes in gradient, rather than averages of particular temporal segments of the data.

7.3. TTest

In order to better understand the behaviour of the algorithm a new artificial dataset was created to overcome the difficulties of the CBF domain. This artificial dataset tries to capture some of the variations that occur in real-world data. *TTest* is a problem domain with three classes and three channels. It has a total of five parameters controlling various characteristics.

There are three classes, imaginatively called A, B and C. The three channels, likewise, are imaginatively termed alpha, beta and gamma. For each of the three classes, there is a prototype, as shown in figures 11. All the prototypes are all exactly a hundred units of time long.

One form of variation is temporal stretching. In general, temporal stretching is non-linear; in other words, temporal stretching need not be of the kind where the signal is uniformly slowed down or sped up; rather, some parts can be sped up while others can be slowed down. However, for simplicity, we will assume linear temporal stretching. The amount of temporal (linear) stretching for the *TTest* dataset is controlled by the parameter d (short for overall duration). That is to say, d specifies the percentage variation of the duration. For example, if $d = 0.1$, it means that duration in frames would vary from 90 to 110.

Another form of variation is random noise. For this problem, we will assume that such noise is Gaussian. This is not an unreasonable assumption, as Gaussian noise is typical of many types of sensors and measurements that are used in temporal data. For the *TTest* dataset, the amount of noise is controlled by the parameter g, the noise on all channels is taken by multiplying the parameter g by the random Gaussian noise function:

$$noise() = g * \epsilon()$$

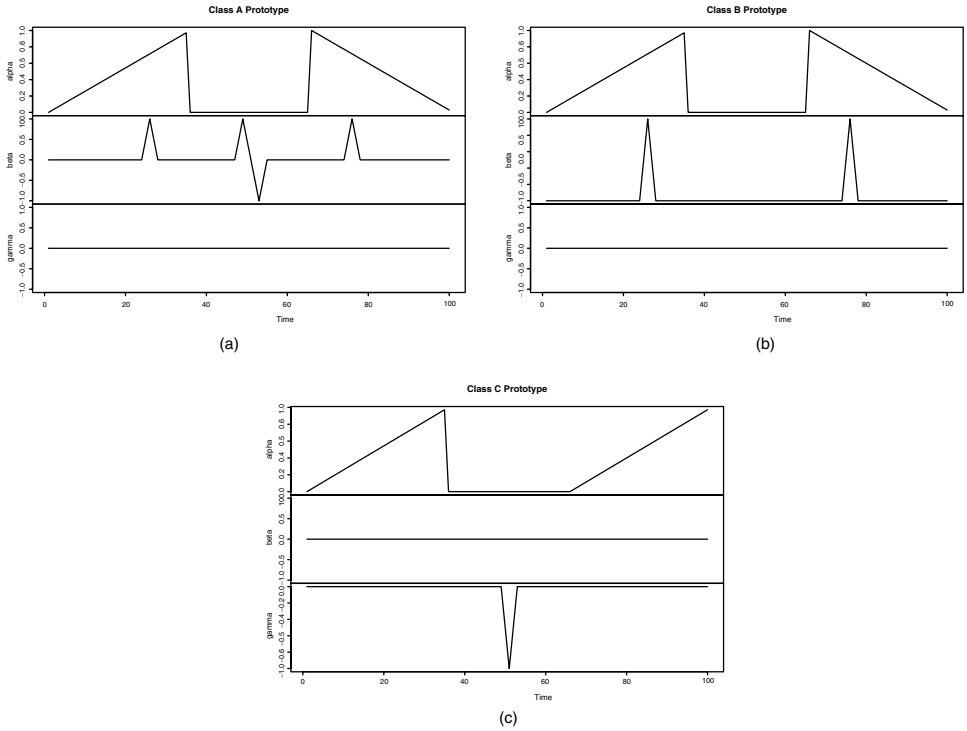

Figure 11. Prototype for classes A, B and C.

As mentioned before, temporal stretching is only a linear approximation of a non-linear process. Hence, we can also modify the times of events within each instance relative to one another. In *TTest*, this is controlled by the parameter c. Within the dataset, the startpoints and endpoints of increases and decreases, as well as the timing of local maxima and minima, are all randomly offset in time. The amount of variation of these temporal events is modified by a random value c.

Also, as with real datasets, sometimes the amplitudes of the various events vary. In *TTest* the parameter h determines the amount of variation in the amplitude of the local maxima of the signal.

Finally, one problem that often occurs in real datasets is that there is some data on a channel that looks useful, but in fact is irrelevant. For this reason, for classes A and B, the gamma channel was replaced with something that looks plausible as a signal: a sequence of between 2 and 9 random line segments whose endpoints are randomly generated. For class C, the beta channel is replaced with a similarly generated sequence of random line segments. Note that this is meant to explore a different issue to that of the Gaussian noise above. It is meant to test the learner's ability to cope with data that is irrelevant.

This dataset has some useful properties. Firstly, it is truly multivariate. Secondly, it has variation in the length of streams and the onset and duration of sub-events. Thirdly, it has

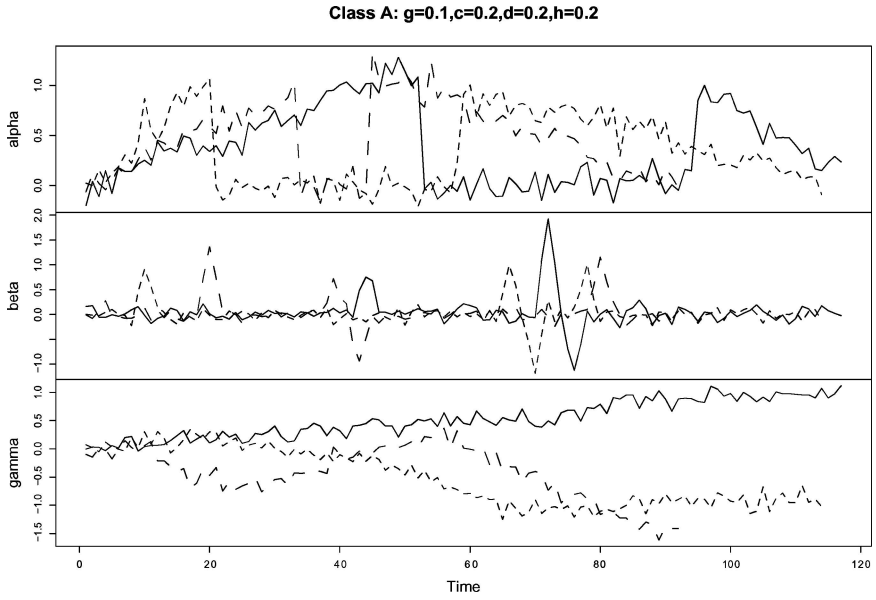

Figure 12. Examples of class A with default parameters.

variation in the amplitudes of events as well as Gaussian noise (as does CBF). Fourthly, it has several sub-events and a complicated relation between sub-events which is typical of real-world domains. CBF only had one of these properties.

7.3.1. Experimental results. As a starting dataset, we generated 333 instances of each class, using the following parameters: $g = 0.1, d = 0.2, c = 0.2, h = 0.2$ with irrelevant features switched on. A complete definition can be found in Kadous (2002). Figure 12 shows several examples from class A.

The results are shown in Table 6. We see here the success of the voting algorithm in classifying instances.

7.3.2. Comprehensibility. To check that *TClass* worked, we switched off Gaussian noise, but otherwise left the parameters the same. The resulting definition is a good approximation of the original concept.

The definitions arrived at for the no-noise case shown in figure 13 are exactly correct. It looks for the increasing alpha value distinctive of the C channel, as well as the high gradient decreasing event in the middle of the A class.

As we add noise, the definitions become slightly more complex, but are still understandable. The first rule looks for the beta channel with no significant local minima and at least one local maximum, indicating a B class. Similarly, it looks for the big local minimum of the C class in the gamma channel.

Table 6. Error rates on the *TTest* domain with default settings.

Approach	Error
TClass with J48	3.3 ± 0.9
TClass with PART	2.3 ± 0.3
TClass with IBL	68.1 ± 1.5
TClass with Bagging/J48	2.5 ± 0.4
TClass with AdaBoost/J48	1.0 ± 0.3
TClass with Naive Bayes	9.8 ± 1.5
TClass with voting	$\mathbf{0.5 \pm 0.2}$
Naive segmentation	7.2 ± 0.7
Hidden Markov model	4.4 ± 1.5

```
IF alpha HAS Increasing: midTime=98.5 avg=0.47 m=0.03 d=30.0
THEN C (299.0)
OTHERWISE
|   IF beta HAS Decreasing: midTime=59.5 avg=-0.28 m=-0.57 d=4.0
THEN A (277.0)
|   OTHERWISE THEN B (323.0/24.0)

Number of Leaves  :     3

Size of the tree :      5
```

Figure 13. A decision tree produced by *TClass* on *TTest* with no Gaussian noise, but with other temporal parameters set to default values. It is a perfect answer; ignoring gamma altogether as a noisy channel and having the absolute minimum number of nodes.

```
IF beta HAS LocalMax: time = 48.0 val = 0.07 (*1) AND
IF beta HAS NO LocalMin: time = 64.0 val = -0.72 (*2) AND
IF beta HAS NO LocalMin: time = 63.0 val = 0.31 (*3) THEN B (299.0/3.0)

IF gamma HAS LocalMin: time = 59.0 val = -0.00 (*4) AND
IF gamma HAS NO LocalMin: time = 56.0 val = 0.46 (*5) AND
IF gamma HAS LocalMax: time = 26.0 val = 0.03 (*6) THEN C (301.0/3.0)

: A (299.0/2.0)

Number of Rules  :  3
```

Figure 14. A decision list produced by *TClass* on *TTest* with 10 per cent noise.

Table 7. Error rates for the Flock sign language data.

Approach	Error
TClass with J48	14.5 ± 0.4
TClass with PART	16.7 ± 0.9
TClass with IB1	60.3 ± 1.1
TClass with Naive Bayes	29.2 ± 0.8
TClass with Bag	9.4 ± 0.8
TClass with AB	6.4 ± 0.4
TClass with voting (11 voters)	$\mathbf{2.1 \pm 0.2}$
Naive segmentation	5.5 ± 0.5
Hidden Markov model	12.9 ± 0.6

.

7.3.3. Conclusions. *TClass* is able to achieve high accuracy and comprehensible description of the *TTest* dataset. It is competitive in its generation of correct and accurate descriptions—especially in situations where there is little or no noise. If one is not interested in comprehensibility of the learnt concepts, voting methods can be employed to great effect, obtaining accuracies far in excess of those of the baseline learners. On this dataset, it results in an error rate half of that of other learners.

7.4. Auslan

Auslan is the language used by the Australian Deaf and non-vocal communities. Auslan is a dialect of British Sign Language. It is dissimilar to American Sign Language, although some signs have been adopted.

Auslan also has about 4000 "dictionary" signs. For this application, we selected 95 signs, based on coverage of different handshapes, types and so on. Samples from a single signer (a native Auslan signer)[7] were collected over a period of nine weeks. In total, 27 samples per sign, and a total of 2565 signs were collected. The average length of each sign was approximately 57 frames.

The equipment used consisted of two instrumented gloves with magnetic position trackers. Each hand therefore generated a total of 11 features: 3 for orientation (roll, pitch, yaw), 3 for position (x, y, z) and 5 for finger bends. The gloves have excellent accuracy; within degrees for orientation, centimetres for position, and 64 usable levels of finger bend, all updated at 100 frames per second.

The results of classification tests are shown in Table 7. *TClass* on this data set performed very well. On average, the tree constructed by J48 in the above has 132.6 leaf nodes. Given the 95 classes we have, this is an excellent result as it means there are about 1.4 leaf nodes per class. Similarly, PART generates 108.6 rules on average.

7.4.1. Comprehensibility. Comprehending a decision tree with 132 nodes is difficult. In order to consider comprehensibility and simplify comparison, *TClass* was used to learn a binary classification task of one sign against all other signs.

91
n (end dir, redu) **THANKS**. *v (end dir, redu)* **THANK**, *SE* **THANK**. Opaque sign. 'Thank you' is sometimes produced as a blend of THANK + YOU (i.e., as THANK ending in the G handshape directed at one's interlocutor). *A one-handed sign in which the B handshape, palm towards, hand up, is placed on the chin and moved away with stress.*

Figure 15. The gloss for the Auslan sign `thank`.

```
PART decision list
------------------
IF rroll HAS NO LocalMin: time = 0.19 val = -0.28 AND
IF rroll HAS NO LocalMin: time = 0.21 val = -0.26
THEN not-thank (2367.0)

IF rx HAS NO Increasing: midTime = 0.11 avg = 0.0 m = 0.18 d = 0.23 AND
IF rz HAS NO Decreasing: midTime = 0.14 avg = 0.04 m = -1.00 d = 0.17 AND
IF lroll HAS NO Increasing: midTime = 0.72 avg = 0.00 m = 0.01 d = 0.5
THEN not-thank (154.0)

rring-mean <= 0.01 AND
IF lx HAS NO Increasing: midTime = 0.32 avg = -0.02 m = 0.35 d = 0.20
THEN thank (27.0)

: not-thank (17.0)

Number of Rules  :      4
```

Figure 16. A decision list produced by *TClass* on the Flock sign data for the sign `thank`.

The tree produced for the sign `thank` is shown in figure 16. The dictionary "gloss" for the sign `thank` is shown in figure 15 from Johnston (1989).

The rule can be interpreted in the following manner. The first rule says: if the right hand does not roll to a palm up position early in the sign, then it can not be the sign `thank`.[8] This is a logical first-cut: there are very few signs that begin in a palm-up position and it eliminates about 92 per cent of the training instances.

The second rule checks to see if the hand is moved horizontally and laterally towards the body. If it has either of these characteristics, then it can not be the sign `thank`.

The third rule checks the ring finger (a good indication of the state of the palm). If the ring finger is, on average, very close to unbent (as would be the case for an open-palm sign like `thank`) and the left hand doesn't move horizontally at all, then it is the sign for `thank`. The "at all" comes from looking at the event index.

7.4.2. Conclusions on auslan. Using *TClass* we have created a learner that can produce low error rates on classification when voted. When not voted, it can be used to create comprehensible descriptions that are easily understood in this context. *TClass* produces definitions that can be compared (favourably) against definitions in the Auslan dictionary.

7.5. ECG

Electrocardiograms (commonly abbreviated as ECGs) are a non-invasive technique for measuring the heart's electrical activity. By "listening in" on these signals, we can diagnose problems within two main areas: firstly, the electrical system itself (these are called *Type B* diagnostic statements); and secondly, tissue damage to the heart, such as valve blockages, muscle misfires and other physiological issues that manifest themselves as alterations to the heart's electrical system (these are referred to as *Type A* diagnostic statements). Type A problems are more difficult to diagnose, since we are observing the effect of physiological phenomena on the electrical system, rather than the electrical system itself.

ECGs are captured by attaching a number of electrodes around the body, most of which are near the heart, but some of which are at the extremities. Once the electrodes are connected; voltages, and differences between the voltages at different electrodes, are recorded as time series.

7.5.1. Previous work. There has been a lot of work on analysing ECGs within the artificial intelligence and machine learning communities. Perhaps the most significant of these was Bratko et al's work on the KARDIO (Ivan Bratko, 1989) methodology and model for the heart. However, it is interesting to note that in discussing its application in practice, Bratko said:

> In respect to clinical application of KARDIO, the cardiologists felt that a significant limitation is that KARDIO accepts as input symbolic ECG descriptions rather than the actual ECG signal. Thus the user is at present required to translate the patient's ECG waveform into the corresponding symbolic descriptions ... In presently available ECG analysers, these difficulties [in extracting symbolic descriptions] lead to unreliable recognition of some of the ECG features.

TClass provides a mechanism for overcoming this problem: it ECGs as input. Further, KARDIO was developed to diagnose Type B problems; whereas in this research we tackle the harder Type A classification problem.

Our dataset was a random selection of 500 recordings from the full CSE diagnostic database (Willems et al., 1990). This dataset was also explored by de Chazal (1998). The records come from men and women with an average age 52 ± 13 years. The sample rate was 500 Hz. There were seven possible classes: normal (NOR), left ventricular hypertrophy (LVH), right ventricular hypertrophy (RVH), biventricular hypertrophy (BVH), acute myocardial infarction (AMI), inferior myocardial infarction and (IMI) and combined myocardial infarction (MIX). The classes are not evenly distributed, with the most common having 155 examples, and the least having 21 examples. The class labels were determined

independently through medical means (e.g. surgery after the ECGs were recorded), so the class labels can be assumed to be free of noise.

Each recording consists of 15 channels. These include the three Frank leads that provide a 3D representation of the heart: X, Y and Z, the raw sensors V1 through to V6 as well as aVF, aVR and aVL.

The focus of de Chazal's thesis was the manual construction of a set of features for classification of Frank lead electrocardiogram. A filter was applied to the ECG to remove "baseline wander" due to the patient's respiration at about 0.5 Hz, and mains electricity at 50 Hz. We also used the de Chazal's segmentation of the ECG recordings into individual beats. A typical ECG recording consists of several beats. These heartbeats must be segmented into individual heartbeats so that learning can take place. For our data, the ECGs are segmented according to the Q wave onset time.

In de Chazal's work, a variety of learners were applied. The two main families examined were voted softmax neural networks and C5.0. He also employed a number of feature selection techniques.

The lowest error rate achieved was 28.7 per cent, obtained by using 100 voted softmax-neural networks, each with different initial conditions applied to all features. These results compare extremely well with median error rate for human cardiologists of 29.7 per cent (a human panel obtained approximately 25.2 per cent).

However, there are some limitations of de Chazal's work. Firstly, because neural networks were used (and in particular, 100 voted neural networks) the results are not very comprehensible. Clearly it would be desirable, especially from a medico-legal standpoint, for explanations of classifications of ECGs to be given. Secondly, de Chazal—building on decades of research—spent at least 3 years developing software for extracting features. In other domains, such background knowledge may not be available, or may be very costly.

7.5.2. Experimental results. In his thesis, de Chazal took each of the ECGs with its heartbeats, and selected what is termed the "dominant" heartbeat. The dominant heartbeat is the one that is the most free of noise. Each ECG consisted of between 4 and 16 heartbeats, with an average of approximately 8. Typically, the dominant heartbeat was the 6th heartbeat; however, domain experts analysed each example to see if they were "typical" examples of the beat. This, of course, requires domain knowledge.

For comparison with de Chazal's, we used the same dominant beats for our first group of experiments.

It is interesting to note that these data files are much larger than a typical instance in a propositional problem: each training instances was between 20 kilobytes and 84 kilobytes, with the average being 47 kilobytes. Even considering only dominant beats, it is still approximately 24 megabytes of raw data.

Each dominant beat was extracted and labelled by the class. Using 10-fold cross validation (as de Chazal did), we used *TClass* to learn all 7 classes. The results of the experiments are shown in Table 8. For *TClass* we used both relative time and height because this makes more sense in this domain: the amplitudes depend greatly on the peculiarities of the ECG electrodes; and the beats of the heart vary greatly in duration.

Table 8. Error rates for the ECG data (dominant beats only).

Approach	Error
TClass with J48	51.4 ± 2.0
TClass with PART	45.2 ± 3.0
TClass with IB1	52.4 ± 3.1
TClass with NB	36.6 ± 2.2
TClass with Bag	39.0 ± 2.6
TClass with AB	35.4 ± 3.8
TClass wit voting (11 voters)	$\mathbf{32.6 \pm 2.8}$
Naive segmentation	34.2 ± 2.7
Hidden Markov model	34.4 ± 2.0

The results in Table 8 seems to be very close between *TClass* and the baseline learners. However, it does not compare well with de Chazal's work, or even human experts.

However, there is another way to improve accuracy: As mentioned before, each ECG recording consists of multiple beats, with an average of 8 beats per recording. Unfortunately, these other beats do not represent "good" data for the following reasons:

- They are likely to very strongly correlated with other beats from the same recording—so it is not really new data in some sense. The amount of data is also increased by a factor of 8, which will consequently have an impact on performance and time consumed.
- Some instances have more examples than others, so we may be "biasing" the learning to people with faster heartbeats (since they will have more heartbeats recorded).
- The data has not been "vetted" by domain experts as typical heartbeats.

Hence our algorithm will have to be robust to noise and able to cope with biases in the input data, as well as able to cope with large amounts of data. In fact, using this approach will use approximately 200 megabytes of input data.

TClass was able to cope with this amount of data, even on a computer with only 512 megabytes of RAM. It necessitated setting some of the parameters for the feature extractors to not produce superfluous events—e.g. tiny maxima and minima of no importance to classification.

One other observation is that the results can easily be voted across all the heartbeats in the test set. Also note that this is applicable not just to the *TClass* methods but to naive segmentation and hidden Markov models as well.

Table 9 shows how this worked in practice.

7.5.3. Comprehensibility. We used the binary classification rules approach as used on the Auslan datasets to see if there were any intelligent concepts that could be deduced. The results are shown in figure 17.

Table 9. Error rates for the ECG data with all beats used. (*: Only trained on dominant beats).

Approach	Error
TClass with J48	45.5 ± 1.7
TClass with PART	41.9 ± 2.1
TClass with IB1	45.3 ± 1.3
TClass with Bag	35.1 ± 2.6
TClass with AB	32.9 ± 2.4
TClass with AB and voting	$\mathbf{28.0 \pm 1.8}$
Naive segmentation	28.5 ± 2.6
Hidden Markov model	33.5 ± 1.7
de Chazal*	28.6 ± 2.4
Human expert	29.7
Human panel	25.2

```
PART decision list
------------------

IF aVL HAS LocalMin: time = 0.49 val = -43.53 AND
IF x HAS NO LocalMin: time = 0.11 val = -1103.50 AND
IF V1 HAS NO Plateau: midTime = 0.55 avg = -2.38 d = 0.07
THEN not-RVH (3727.0/32.0)

aVL-max > 554.61: not-RVH (92.0)

x-min <= -212.28 AND
IF aVL HAS LocalMax: time = 0.54 val = 45.15 AND
IF z HAS LocalMax: time = 0.42 val = 269.06
THEN RVH (82.0/3.0)

x-min <= -221.53 AND
V1-max <= 317.57 AND
IF z HAS LocalMin: time = 0.45 val = -24.93
THEN RVH (67.0/31.0)

V1-max <= 352.09: not-RVH (108.0/4.0)

: RVH (62.0/14.0)

Number of Rules  :      6
```

Figure 17. A two way classifier for the `RVH` class in the ECG domain.

To gain some insight as to whether this was a useful rule, it was compared with the rules used by a commercial ECG classifier based on an expert system (Schiller Medical, 1997).[9] The definitions produced show the same characteristics as the definitions produced in this manner for the Auslan domains: the first few rules provide a "first cut" exclusion. The third rule looks for a very low minimum on the X value. This is because patients with RVH have a depression in the S wave (Schiller Medical, 1997), leading to a very large minimum x value. In de Chazal's work (de Chazal, 1998) (page 179), he found that the minimum x value is the most discriminant feature based on rank-correlation analysis. The other criteria look for local maxima in the aVL and Z channels that are unique to right ventricular hypertrophy cases: the T wave is biphasic, i.e. rather than having a single maximum, it has two maxima; while most normal heartbeats will have one maximum occurring slightly later (around time 0.7) (Schiller Medical, 1997).

7.5.4. ECG conclusions. Our accuracy results are competitive with a median human cardiologist; and also competitive to a learner trained on the same dataset, but with the application of copious quantities of background knowledge. Given that we did not use any background knowledge, the results are good.

8. Conclusions and future work

Each of the novel contributions has had an impact on the results. By using metafeatures, the temporal properties are converted into propositional attributes. The directed segmentation process captures the temporal properties well, and hence classification is performed with high accuracy. It also allows us to combine temporal properties with the aggregate properties of the time series and specified attributes to leverage the success of propositional learners. Further, the metafeature approach, combined with the postprocessing of learnt classifiers allows the construction of comprehensible classifiers that in our tests showed a strong correspondence with understood concepts in the problem domains.

Metafeatures have been applied to diverse domains that exhibit difficult properties: it has been tested on domains with up to 22 channels, 110 metafeatures, 200 megabytes of data, 95 classes, and highly skewed class distributions. They have been shown capable of producing high-accuracy classifiers, in fact, classifiers that match hand-crafted preprocessing techniques. Although the user must define the metafeatures, we have shown that a generic family of metafeatures work well for diverse temporal domains. Furthermore, they produce comprehensible descriptions.

However, results show that it may have difficulty generating rules that are simultaneously accurate *and* comprehensible. This suggests one avenue for future work. The marked difference between voted and unvoted results points to the weakness of the random search for a good segmentation. Several solutions are being explored, including the following: using a decision tree builder with heavy pruning to do the directed segmentation, putting the learner into the loop for selecting a good segmentation, using a hill-climbing approach.

One surprising result from the experimental stage was the success of the "naive segmentation" algorithm—although it did not create comprehensible results and despite its simplicity, it produced some surprisingly good results.

There are several other issues to explore: automatic selection and generation of metafeatures, and letting the class label be more complex (for example, in temporal domains allowing y to be a sequence of class labels). We also plan several practical steps: applying metafeatures to new domains, for example robot vision.

Notes

1. These datasets are: arrythmia, audiology, bach chorales, echocardiogram, mobile robots and waveform.
2. Note that recognising sign language as a whole is extremely difficult—to mention four serious difficulties: the use of spatial pronouns (a certain area of space is set to represent an entity); the use of *classifiers*—signs that are physically descriptive but are not rigidly defined; the importance of facial gestures for many signs and finally improvised signs.
3. Auslan is the name used for AUstralian Sign LANguage, and it is the language used by the Deaf community in Australia.
4. Typically, these synthetic events would *not* be chosen manually, but by an automated process.
5. Most commonly, the trees would be more complex, testing for the absence or presence of elements in multiple regions.
6. We have not found references to this as a general technique. However it seemed so obvious that it was not considered worthy of publication. Other researchers such as Geurts (2001) have since cited us for the algorithm.
7. My deepest gratitude extends to Todd Wright for his help on this project.
8. There are two events tested against here; this is an artifact of the random segmentation process.
9. Unfortunately, companies are unwilling to disclose the exact performance of their systems for direct comparison.

References

Aurenhammer F., & Klein, R. (2000). Voronoi diagrams. In J. Sack & G. Urruita (Eds.), *Handbook of Computational Geometry*. Elsevier Science.

Bengio, Y. (1996). *Neural Networks for Speech and Sequence Recognition*. International Thomson Publishing Inc.

Blake, C. L., & Merz, C. (1998). UCI Repository of machine learning databases.

Box, G. E. P., & Jenkins, G. M. (1976). *Time Sereis Analysis: Forecasting and Control*. Holden Day.

Bracewell, R. N. (1965). *The Fourier Transform and Its Applications*. New York: McGraw-Hill.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*, 123–140.

Cohen, W. W. (1995). Learning to classify English text with ILP methods. In L. D. Raedt (Ed.), *Proceedings of the 5th International Workshop on Inductive Logic Programming* (pp. 3–24) Department of Computer Science, Katholieke Universiteit Leuven.

Das, G., Lin, K.-I., Mannila, H., Renganathan, G., & Smyth, P. (1998). Rule discovery from time series. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*. AAAI Press.

de Chazal, P. (1998). Automatic classification of the Frank lead electrocardiogram. Ph.D. thesis, University of New South Wales.

Dietterich, T. G. (2000). The divide-and-conquer manifesto. In *Proceedings of the Eleventh International Conference on Algorithmic Learning Theory* (pp. 13–26). Springer-Verlag.

Fayyad, U. M., & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *IJCAI-93*: *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence* (pp. 1022–1027). Morgan-Kaufmann.

Geurts, P. (2001). Pattern extraction for time series classification. In L. de Raadt and A. Sieves (Eds.), *Principles of Data Mining and Knowledge Discovery, 5th European Conference, PKDD 2001 Proceedings*. Freiburg, Germany: Springer-Verlag.

Goodwin, G. C., Ramage, P. J., & Caines, P. E. (1980). Discrete time multivariable adaptive control. *IEEE Trans. Automatic Contro, 25*, 449–456.

Ho, Y. C., Sreenivas, R. S., & Vakili, P. (1992). Ordinal Optimization of DEDS. *Discrete Event Dynamic Systems*: *Theory and Applications, 2*(1), 61–88.

Ivan, B., Igor Mozetic, N. L. (1989). *KARDIO*: *A Study in Deep and Qualitative Knowledge for Expert Systems*. MIT Press.

Johnston, T. (1989). *Auslan dictionary: A Dictionary of the Sign Language of the Australian Deaf Community*. Deafness Resources Australia Ltd.

Kadous, M. W. (1995). GRASP: Recognition of Australian sign language using instrumented gloves. Honours Thesis.

Kadous, M. W. (2002). Temporal classification: Extending the classification paradigm to multivariate time series. Ph.D. thesis, School of Computer Science and Engineering, University of New South Wales.

Keogh, E., & Pazzani, M. (2001). Dynamic time warping with higher order features. In *SIAM International Conference on Data Mining, SDM 2001*. SIAM.

Keogh, E. J., Chakrabarti, K., Mehrotra, S., & Pazzani, M. J. (2001). Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD Conference*.

Lee, J. K., & Kim, H. S. (1995). *Intelligent Systems for Finance and Business*. Chapt. 13. John Wiley and Sons Ltd.

Liu, H., & Motoda, H. (Eds.). (1998). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers.

Mallat, S. (1999). *A Wavelet Tour of Signal Processing*. Academic Press.

Manganaris, S. (1997). Supervised classification with temporal data. Ph.D. thesis, Computer Science Department, School of Engineering, Vanderbilt University.

Mannila, H., Toivonen, H., & Verkamo, A. I. (1995). Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)* (pp. 210–215).

Michalski, R. S., Mitchell, J. G., & Carbonell, T. G. (Eds.). (1983). *Machine Learning: An Artificial Intelligence Approach*, Chapt. A Theory and Methodology of Inductive Learning. Tioga Publishers.

Myers, C. S., & Rabiner, L. R. (1981). A comparative study of several dynamic time-warping algorithms for connected word recognition. *The Bell System Technical Journal*, *60*7, 1389–1409.

Oates, T., Schmill, M. D., & Cohen, P. R. (2000). A method for clustering the experiences of a mobile robot that accords with human judgments. In *Proceedings 17th National Conference on Artificial Intelligence* (pp. 846–851). AAAI Press.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE, 77*:2, 257–286.

Rodríguez, J. J., Alonso, C. J., & Boström, H. (2000). Learning first order logic time series classifiers. In J. Cussens, & A. Frisch (Eds.), *Proceedings of ILP2000* (pp. 260–275).

Rosenstein, M. T., & Cohen, P. R. (1998). Concepts from time series. In *AAAI '98: Fifteenth National Conference on Artificial Intelligence* (pp. 739–745). AAAI Press.

Saito, N. (1994). Local feature extraction and its application using a library of bases. Ph.D. thesis, Yale University.

Schapire, R. E. (1999). A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*.

Schiller Medical (1997). *The Schiller ECG Measurement and Interpretation Programs Physicians Guide*.

Srinivarsan, A. (2000). The aleph manual. Technical report, Oxford University.

Statsoft (2002). *Electronic Statistics Textbook* (http://www.statsoft.com/textbook/stathome.html). Tulsa, OK: Statsoft.

White, A. P., & Liu, W. Z. (1994). Bias in information-based measures in decision tree induction. *Machine Learning, 15*, 321–329.

Willems, J. L., Abreu-Lima, C., Arnaud, P., Brohet, C., & Denic, B. (1990). Evaluation of ECG interpretation results obtained by computer and cardiologists. *Methods of Information in Medicine, 29*4, 308–316.

Witten, I. H., & Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.

Young, S., Kershaw, D., Odell, J., Ollason, D., Valtchev, V., & Woodland, P. (1998). *The HTK Book*. Microsoft Corporation.

Received March 30, 2004
Revised October 12, 2004
Accepted October 25, 2004
Final manuscript October 25, 2004