A General Architecture for Supervised Classification of Multivariate Time Series

Mohammed Waleed Kadous
Department of Artificial Intelligence
School of Computer Science & Engineering
University of New South Wales
Sydney NSW 2052, Australia
waleed@cse.unsw.edu.au

UNSW-CSE-TR-9806 – September 1998

Contents

List of Figures

List of Tables

Abstract

Supervised classification has been one of the most active areas of machine learning research. However, the domains where it has been applied are relatively limited. In particular, much of the work has focused on classification in static domains, where the attributes of the training examples are assumed not to change over time. In many domains, attributes are not static; in fact, it is the way they vary temporally that can make classification possible. Examples of such domains include speech recognition, event recognition from sensors in robotics and analysis of electrocardiographs.

So far, researchers tackling these domains have used ad-hoc techniques for converting the problem to a standard classification task. This fails to take into account both the special problems and special heuristics applicable to temporal data.

This paper proposes a general architecture for classification of multivariate time series. Training proceeds in five steps: extraction of events from the data training based on parametrised event primitives; clustering of the events in their parameter space to create synthetic events; event attribution of the training data and finally building a classifier with a conventional learner. Recognition takes two steps: selective event searching for synthetic events within the test instance (usually only a small subset of the synthetic events generated in training need to be searched for), then feeding through the classifier created in the training stage.

An example implementation of this general architecture is presented. Some preliminary results of its application to recognition of signs from Australian Sign Language are also discussed.

Keywords: machine learning, classification, temporal classification, gesture recognition, time series.

Chapter 1

Introduction

Supervised classification has been one of the most active areas of machine learning research. However, in most of the algorithms and techniques presented, the model of the domains has been static; that is, the training and test instances have been described in terms that don't allow for changes over time.

This is surprising, because some of the most useful and interesting domains are dynamic in nature; the description of the instances inherently changes over time. Furthermore, it is only the nature of these changes that makes classification possible. Examples of such domains include recognition of gestures, speech recognition, medical signal analysis, robots detecting temporal events from sensor readings [RC98], data mining in temporal databases and many more.

Consider the classification of spoken words. We can measure what are called the Cepstral coefficients (which try to model the human aural system) of the incoming speech signal. There might be 22 such coefficients, with the values of these coefficients updated every 20 milliseconds. Looking at the value of these coefficients at a single point of time tells us very little about the word that it comes from. Only by looking at how the Cepstral coefficients have changed can the utterance be classified as a particular word. Consider a robot that is watching a door. While the robot might be able, using existing learning techniques, to learn from sensors when the door is open or closed; it would be a great deal harder for the robot to learn from sensors when the door was open*ing* or clos*ing*.

No doubt, these problems can be converted to conventional classification tasks. However, techniques for doing so have generally been ad-hoc, labour-intensive and domain-specific. For example, Kadous [Kad95] does so for the sign language domain. In addition, such techniques do not make good use of the special temporal properties and heuristics that are likely to apply in such domains. Nor do they take into account the special ways in which temporal data can vary.

This work presents a general architecture that can be used for supervised classification in these domains, which can be described as multivariate (in the sense that there is more than one attribute being analysed and learnt) time series.

1.1 Goals

Our goal is to develop a supervised temporal concept learner which operates in the type of domain discussed above. The desirable characteristics of such a learner are discussed below. In some domains, we would prefer some of these goals over others. Nonetheless, it is important to specify the range of possible goals. It should also be noted that it is difficult to achieve all of them simultaneously.

Generality

We would like our classifier to work for as many different domains as possible, without large amounts of arbitrary modification to parameters. We already have some existing techniques which can perform well when the costs of manual adjustments bring enough benefits. The best example of this is hidden Markov models, the dominant means of speech recognition. Hidden Markov models (see section 2.1 and [RJ86]) can recognise speech with high rates of accuracy, but require a great deal of tweaking, not all of which is easily extractable from domain knowledge.

At the same time, we want to provide a mechanism for domain knowledge to be included easily, so that (i) the classification task is made easier (ii) concepts are explained in terms related to the domain itself.

Classification Accuracy

Classification accuracy is obviously a desirable aspect in any learner, despite the complexity and issues of measuring this value.

Meaningful descriptions

It is important in some domains to explain the basis of the classification; so it would be useful if our classifiers, rather than just returning a class, also returned a reason for the classification. This would also be useful in applications such as data mining, where it would enable the discovery of interesting patterns.

Fast learning curve

In some domains, the number of data (especially of some classes) is sometimes limited. Thus it would be best if our algorithm converged to a near-correct answer even from a small amount of data.

Time

For practical purposes, it would be useful if time to train from the data is small. There is also an issue of whether the learner could train incrementally. In many domains, we would also like "on-line" testing to be possible. This is where we classify signals as they arrive. For example, one possible example would be to collect sensor data from a robot, train from the data in batch (off-line) and then to install the classifier produced by training on the robot once again, such that as soon as sensor readings arrive, and as soon as a particular class is observed, a classification is produced.

Chapter 2

Related work

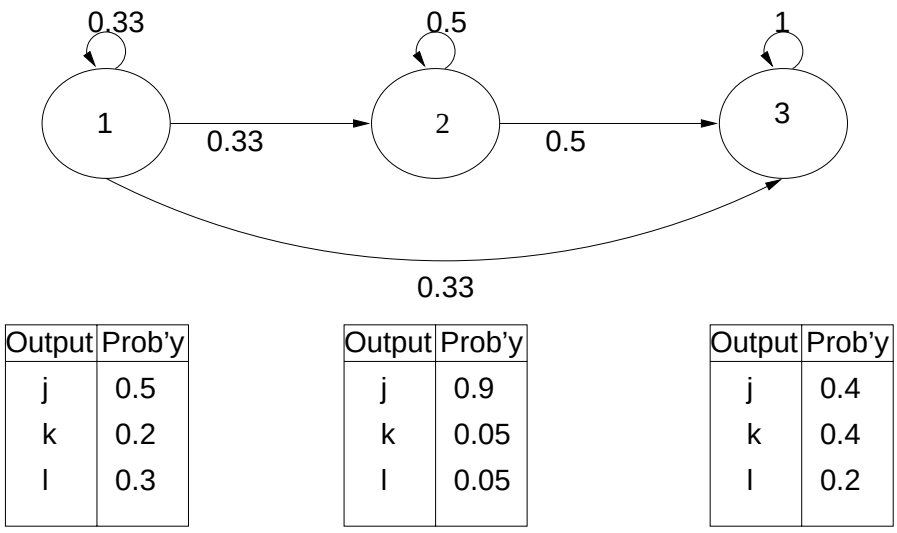
Many techniques have been developed for handling temporal classification in specific domains, but there are few that can be generalised. Many of the techniques come from speech recognition community; which is the most salient temporal classification domain.

2.1 Hidden Markov models

Hidden Markov models (HMMs) are a well-developed technology for classification of multivariate data that have been used extensively in speech recognition, handwriting recognition and even sign recognition. They have been applied for decades. Due to space considerations, an in-depth analysis of HMMs isn't possible. The interested reader may wish to have a look at [RJ86].

HMMs consist of states, possible transitions between states (and the probability of those transitions being taken) and a probability that in a particular state, a particular observation is made. An observation can be anything of interest. For example, it could be a frame in our definition, or it could be a symbol of some kind. HMMs are termed hidden because the state of the HMM can not, in general, be known by looking at the observations. They are Markov in the sense that the probability of observing an output depends only on the current state and not on previous states. By looking at the observations, using an algorithm known as the Viterbi Algorithm, an estimate of the probability that a particular instance or stream observed was generated by that HMM can be computed. We can have one HMM for every class we are interested in, then compute the probability of the particular stream we saw being generated by each of the HMMs, then choose the most probable one.

For example, consider figure 2.1. This is a very simple hidden Markov model, with three states. At each timestep, a j, k, or l is observed. We can work out the probability that the sequence $[j, j, k, l]$ was generated by this HMM



Output	Prob'y
j	0.5
k	0.2
l	0.3

Output	Prob'y
j	0.9
k	0.05
l	0.05

Output	Prob'y
j	0.4
k	0.4
l	0.2

Figure 2.1: A simple HMM where the only observations are j, k, or l

by considering the most probable state transitions that would generate this output, and computing the probability that it would occur (which is the probability of each output times the probability of the transitions). In this case, the most probable sequence of state transitions (given we start in state 1) is: [1,2,3,3]. The probability of this is $0.5 \times 0.33 \times 0.9 \times 0.5 \times 0.4 \times 1 \times 0.2 = 0.00594$. If we had other HMMs, we could compare the probability for each, and choose the HMM that produced the highest probability for the given observed sequence. If we had trained each HMM to have an appropriate set of parameters for a particular class, then by choosing the HMM which generated the observations with the highest probability, we would also know the class.

Note that observations can be very complex, and need not be as simple as a single letter j, k or l. In general, people who use HMMs model multivariate streams by representing each frame as an observation. The probability of a particular frame is estimated by using a Gaussian mixture over the channels.

The Baum-Welch reestimation algorithm provides a way for the probabilities of both transitions and of observations within states to be estimated from training data.

HMMs have proved to be very effective in practice, producing high levels of classification accuracy, but they fail to meet several of the goals discussed in section 1.1, namely:

- The structure of the HMM is not always obvious. Some domain knowledge is required to construct the states and transitions that an HMM

can take. In addition, there is trial and error involved, and there are sometimes complex tradeoffs that have to be made between model complexity and difficulty of learning, which results in inelegant additions to HMMs like "state-tying" (where two states are "tied" by forcing them to have the same output probability distributions).

- They do not produce comprehensible results. In fact, in some cases, considerable effort goes into deciding what the appropriate number of states, transitions and form of probability distributions should be. The distribution and transition probabilities do not provide an intuitive descriptions of what has been learnt.

- They do not have fast learning curves. This is usually because there are a large number of parameters that must be set. For example, consider the HMM in figure 2.1. Even with such a trivial HMM, there are approximately fifteen parameters that need to be set. It is not atypical to see speech recognition systems that have upwards of 200 parameters per state, and twenty states per word.

- They do not automatically cope with multiple variations within the same class. For example, consider the word *either*. Depending on where you are from, you may pronounce that as `ee-thur` or `ai-thur`. Yet both are from the same class. This would have to be explicitly denoted to the HMM classification system, either through an HMM with a complex set of state transitions or two separate HMMs.

2.2 Recurrent neural nets

Another tool that has been used for analysing temporal classification problems is recurrent neural networks. A neural network consists of a set of units, an example of which is shown in figure 2.2. The unit has a weight associated with each input. A function of the weights and inputs (typically, a squashing function applied to the sum of the weight-input products) is then generated as an output.

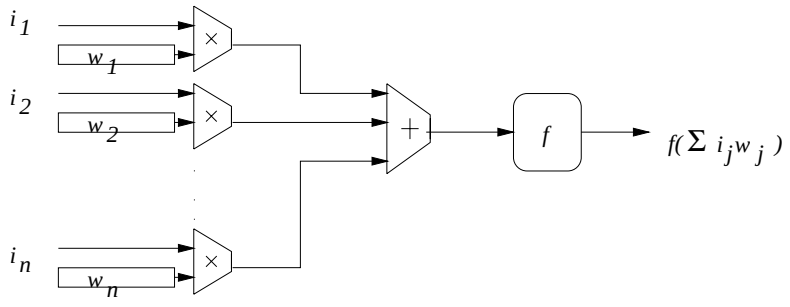


Figure 2.2: A unit

These individual units are connected together as shown in figure 2.3, with an input layer, an output layer and usually a hidden layer. Typically, the input layer consists of one unit per attribute, and the output layer of one unit per class. The number of units in the hidden layer is arbitrary. Through algorithms such as backpropagation, the weights of the neural net can be adjusted so as to produce an output on the appropriate unit when a particular pattern at the input is observed. The interested reader may wish to find out more in [RM86].

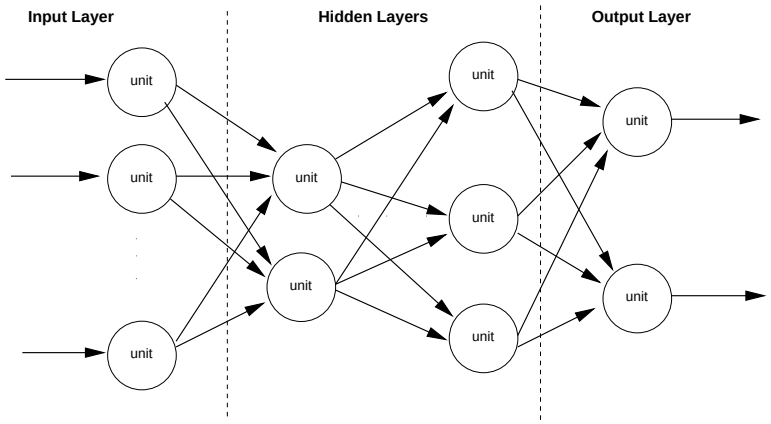


Figure 2.3: A typical feedforward neural network

A recurrent neural network (RNN) is a modification to this architecture to allow for temporal classification, as shown in figure 2.4. In this case, a "context" layer is added to the structure, which retains information between observations. At each timestep, new inputs are fed into the RNN. The previous contents of the hidden layer are passed into the context layer. These then feed back into the hidden layer in the next time step.

In an algorithm similar to the backpropagation algorithm, called back propagation through time (BPTT), the weights of the hidden layers and context layers are set.

To do classification, postprocessing of the outputs from the RNN is performed; so, for example, when a threshold on the output from one of the nodes is observed, we register that a particular class has been observed.

Recurrent neural networks suffer from many of the same problems as hidden Markov models, namely:

- There are many control variables. Many different control variables need to be set in a neural network system, such as the number of units in the hidden layer, the appropriate structure, the correct parameter adjustment algorithm (there are many alternatives to backpropagation), the learning rate, the encoding and more. As of yet, well-formed techniques for deciding appropriate values for these parameters are still developing.

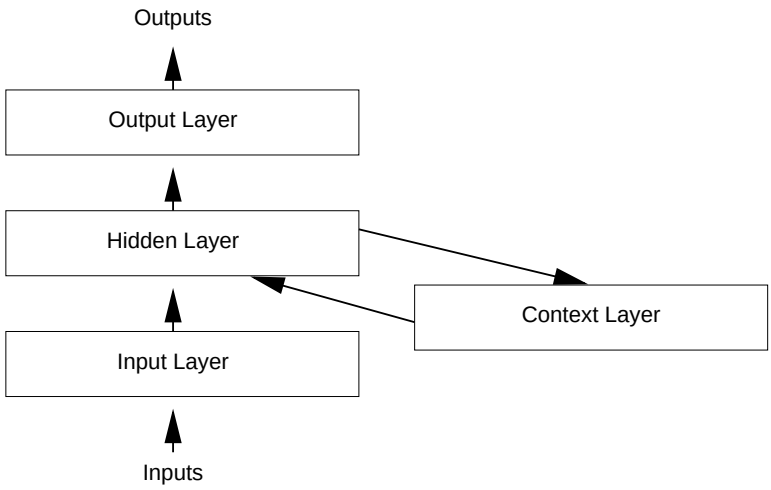


Figure 2.4: Recurrent neural network architecture

- Their efficacy for learning long connected sequences is unproven. Attempts to learn long sequences of events, for example, [Ben96], lasting for hundreds of samples, have not shown good results. They do seem capable of learning short sequences (tens of frames), such as learning individual phonemes, rather than whole words.
- Extracting meaning from a normal feedforward neural networks has proved difficult. Extracting it from recurrent neural networks is even more so.
- They do not have fast learning curves. They usually require many examples before converging, especially if there are many inputs or the hidden layer contains many nodes.

2.3 Dynamic time warping

An older technique that been fallen out of favour since the advent of hidden Markov models is dynamic time warping [MR81]. In some senses, it is the temporal domain equivalent of instance-based learning with a complex distance function. In dynamic time warping, the problem is represented as finding the minimum distance between a set of template streams and the input stream. The class chosen is the "closest" template. However, rather than using a straightforward technique of comparing the value of the input stream at time t to the template stream at time t, an algorithm is used that tries to search the space of mappings from the time sequence of the input stream to that of the template stream, so that the total distance is minimised. This is not always a linear path; for example, we may find that time t_1 in the input stream corresponds to the time $t_1 + 5$ in the template stream, whereas t_2 in the input stream corresponds to $t_2 - 3$ in

the template stream. The search space is constrained to reasonable bounds, such as the mapping function from input time to template time must be monotonically increasing (in other words, the sequence of events between input and template is preserved).

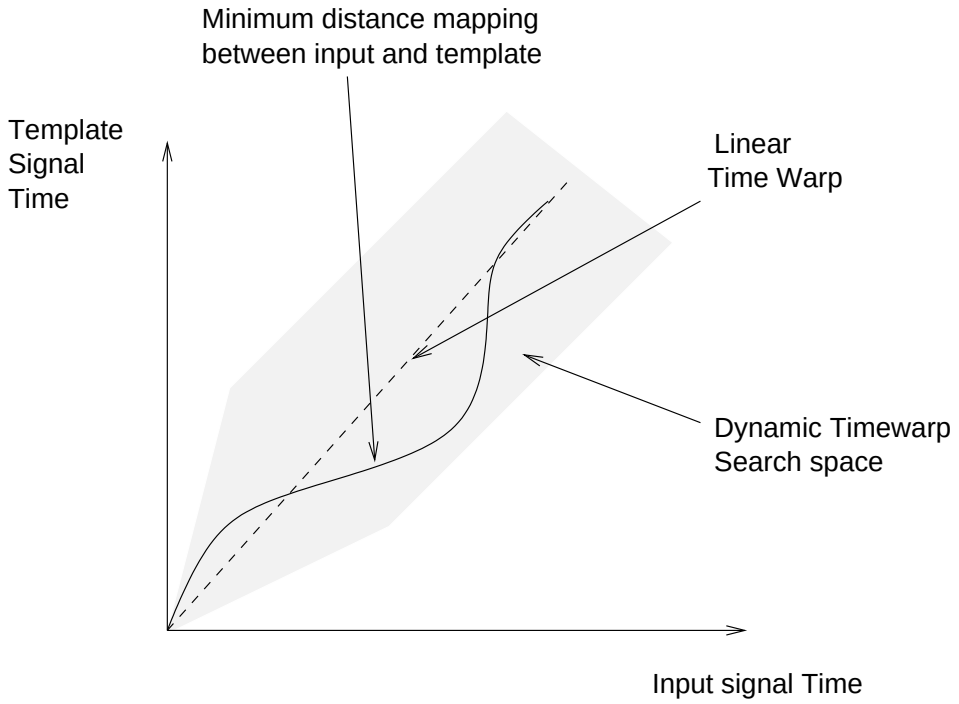


Figure 2.5: Dynamic time warping

This is shown in figure 2.5, where the horizontal axis represents the time sequence of the input stream, and the vertical axis represents the time sequence of the template stream. The path shown results in the minimum distance between the input and template streams. The shaded in area represents the search space for the input time to template time mapping function.

Using dynamic programming techniques, the search for the minimum distance path can be done in polynomial time. However, dynamic time warping has some limitations. Whereas the calculations can be accomplished in polynomial time, this polynomial is approximately $O(N^3V)$ where N is the length of the sequence, and V is the number of templates to be considered. This means that practical applications may be quite slow. In addition, they require the definition of a distance function between the values of the stream at a given time frame. This may be easy if the stream is univariate, but it is more difficult if the stream is, as in our case, a multivariate one. Furthermore, it does not create any meaningful descriptions of the data. In addition, creation of the template streams from data is non-trivial and typically depends on the order of presentation of the training instances, since if we have eight instances from which we are trying to create a template, we

have to warp them to each other in pairs, and repeat the process until we arrive at a single template.

2.4 Recent interest from the AI community

Many other researchers in the area of artificial intelligence have also been looking at areas related to time. These have included work on things such as sequence prediction (e.g. Dietterich and Michalski's work with Eleusis [DM86]), work with temporal logics and their applications to recognising events, for example Kumar's work on temporal event conceptualisation [KM87] and the recent work in context detection and extraction for machine learning applications [Wid96, HHS98]. While some of these areas bear some interesting relations to temporal classification, they differ in several regards. Sequence prediction is not about learning from labelled examples, nor, typically does it deal with multivariate data. The event conceptualisation work focuses on recognition of temporal events, but not learning the events themselves. The work on context detection is about selecting which static classifier to use on a dynamic basis; whereas temporal classification is about classifying the dynamics themselves.

Increasingly, this area has become a popular research topic. For example, a workshop held at AAAI '98 [Dan98] while focusing on temporal prediction, also contained several papers on learning from time series. For example, Keogh and Pazzani [KP98] looks at automated ways of clustering time series from ECG signals and Shuttle information, by using a piecewise model combined with segmentation and agglomerative clustering. In Oates et al. [OJC98], a system is applied to extracting patterns from network failures, by looking at all possible sequences of events and keeping tabs on the frequency of these events.

Shahar [SM95] suggests an expert system architecture for knowledge-based temporal abstraction and also suggests that this system could be used for learning, though he does not actually do so. He then applies the techniques to clinical domains. Paliouras [Pal97] discusses refinement of temporal parameters in an existing temporal expert system. Manganaris [Man97] developed a system for supervised classification of univariate signals using piecewise polynomial modelling combined with a scale-space analysis technique (i.e. a technique that allows the system to cope with the problem that patterns occur at different scales) and applies them to space shuttle data as well as an artificial dataset.

Mannila et al [MTV95] have also been looking at temporal classification problems, in particular applying it to network traffic analysis. In their model, events are modelled not as a set of channels, but as a sequence of time-labelled events. Learning is accomplished by trying to find sequences of events and the relevant time constraints that fit the data.

Das et al [DLM$^+$98] also worked on ways of extracting rule from univariate

data trying to extract rules of the form "A is followed by B".

Rosenstein and Cohen [RC98] used delay coordinates (a representation where the state at time $t-n$ is compared to its state at time t with n varied appropriately to give an appropriate "delay portrait"). These delay portraits are then clustered to create new representations, but they tend to be sensitive to variations in delay and speed.

Another interesting development in recent years is the application of dynamic Bayesian networks to temporal classification tasks. While they are not specifically designed for temporal classification (they are more commonly used for prediction, or for estimating current state given the previous state estimate), Zweig and Russell [ZR98] have applied it to the task of speech recognition. The main problem with using dynamic Bayesian network is that while algorithms for learning the parameters of a Bayes net are well-advanced, learning the structure of Bayes nets has proved more difficult[1]. Friedman et al. [FMR98] are developing techniques for learning the structure of dynamic Bayes networks; it remains to be seen whether these techniques can be applied in temporal classification domains.

[1] In many ways, dynamic Bayes nets suffer the same problems as HMMs. In fact, it can be proved that dynamic Bayes nets and hidden Markov models are isomorphic. Dynamic Bayes nets do allow for far more complex state models and provide a much more intuitive mechanism than HMMs for inclusion of background knowledge.

Chapter 3

Distinctions from traditional classification

A valid and important question is: how do existing classification systems (such as neural networks, decision tree classifiers, rule classifiers, etc) fare in such a domain? Why do we need new tools if existing systems work fine?

Unfortunately, feeding in the raw data to conventional learning algorithms is likely to perform poorly under most circumstances. In some domains it is possible to use extensive domain knowledge to manually create techniques for extracting attributes from the raw data and then feeding these into a classification algorithm. This sometimes works well; but it does not generalise and rarely produces meaningful descriptions of what has been learnt. Also, they may fail to take advantage of heuristics that can be applied.

We first investigate why existing techniques and heuristics for conventional classification are likely to fail in the case of these temporal domains.

3.1 How it is different

In this section, we discuss why an attribute-value learner would not be able to cope with the domains being explored. Consider a simple conversion to a set of features. We could take each channel at each frame and treat it as an attribute. This would not really be effective because:

3.1.1 An indeterminate number of features

Since the number of frames is not fixed; we do not know a priori how many features there are. For example, we might have one instance last for 43 frames and another instance of the same class last for 57 frames. How do we match these? One way is to "pad out" to the maximum number of frames; or to truncate instances to some maximum limit. Neither of these solutions

is really very elegant. We may be losing important data that is critical to classification.

One solution is to use something other than an attribute-value learner; for example a relational learner, where the observation language allows for descriptions not limited to a fixed set of attributes. This may work if the learner is coping with large amounts of data (see next section). Most existing relational learning techniques do not meet this criterion.

3.1.2 Many features, not enough data

If we do choose to use the raw data, even in truncated form, then we still have many features to deal with. For example, assume that there is a domain with 10 classes, 10 channels and an average stream length of 50 frames. The total number of features we would get by "flattening" would be approximately 500 features (10 channels times 50 frames).

Quinlan [personal discussion] points out a rule of thumb which says that as a bare minimum for most learning tasks, there should be at least as many training instances per class as there are features. Thus we would need at least 5000 training streams for the simple example above. This is a simple empirical heuristic; but it highlights the problems that a large number of features introduces.

3.1.3 Strong attribute correlation

In many learning systems, we assume that the attributes are independent in some way; if not independent, then at least measuring different observations of the object we are trying to recognise. In this case, however, we *know* that the attributes are not independent; in fact there is usually a high correlation. For example, consider a position sensor on a robot. We know that due to physical constraints, the position at time $t+1$ is correlated with the position at time t.

In fact, we can take advantage of the fact by making our temporal learning algorithms use the correlations to reduce the data significantly.

3.1.4 Temporal mismatch

Consider a simple classification task with a single continuous channel, such as the three streams shown in figure 3.1. Stream I and II are very similar, except that in stream II the "hump" happens a little later. In many domains (though not all) we want to classify I and II as "similar" in some sense, and III as belonging to a different class. If we simply compare values at the same time, we get a misleading measure of similarity, since on a time-by-time comparison, stream III is closer (using, for example, a least-squares comparison) to stream I than stream II is.

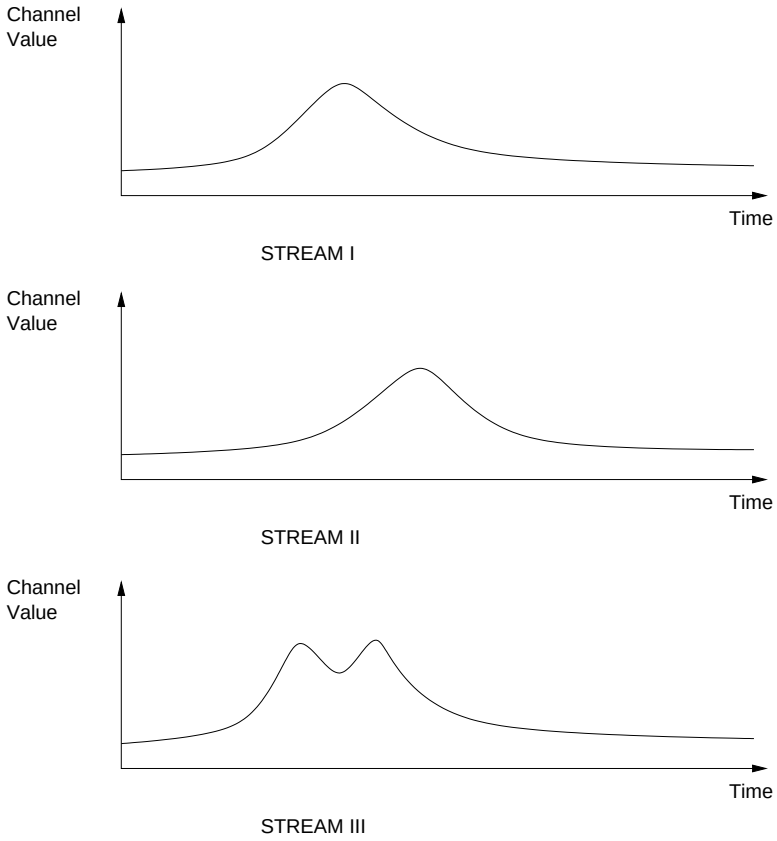


Figure 3.1: Temporal mismatches

This is because we are "mismatching" time in streams I and II. In this particular case, it was caused by delay, but it could have equally been caused by the hump being longer or shorter.

It is certainly possible to try to minimise the mismatch between I and II by stretching. This is very similar in many regards to dynamic time warping as discussed in section 2.3.

3.2 New heuristics

In addition to the reasons why standard attribute-value classifiers are not easy to use for temporal classification, there are also special properties and heuristics that apply in temporal classification domains that do not occur elsewhere. The first thing that one observes when one is looking at datasets of this type is that there appear to be particular "shapes" that occur on given channels that are distinctive. For convenience, we will term a particular shape of some subsequence of a channel as an event.

3.2.1 Temporal Concurrency and Sequences

If events occur concurrently across different channels, and this occurs consistently in our training data, then there is a high probability that this is not a coincidence and can be considered as a defining characteristic of the class. Similarly, if a particular sequence of events occurs consistently for a given class, then our learning algorithm can take advantage of this.

3.2.2 Representing time explicitly

An examination of domains where temporal classification arises indicates that signals that are of the same class tend to vary in the following ways:

- There are random amplitude variations in the channels – what would normally be classified as noise. Also, it may be that a similar-shaped event occurs, but that it has a greater or lesser amplitude as a whole.
- The starting point (often called the "onset") of an event may be different, even though two events may look very similar. An example of this was shown in figure 3.1. Stream I and II would be considered by most to be members of the same class of shapes, even though the "hump" occurs at a different time.
- The duration of an event may be longer or shorter, even though the event may otherwise be the same.

If we can represent the data in a way that explicitly models these events and the variations in their amplitudes, their onset and their durations, then

we may be able to convert the problem into one more palatable to existing classification techniques. This would simultaneously reduce the data we had to deal with, while also making comparisons between streams in a way that is intuitively more appealing.

Chapter 4

Formulation of the problem

4.1 Definition of Terms

4.1.1 Channel

A channel is 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 the remainder of this work, we assume that:

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

The set of values V specifies the values that the channel can take. It may be limited to integers, real numbers, an ordered set, or an unordered set.

How does this representation map to reality? A channel intuitively represents a single attribute's changes in value over time. For example, in sign language recognition, we may be interested in the extent of bending of the forefinger. This channel, of course, varies over time. We may capture information about forefinger bend using some kind of sensor, which could produce a continuous value between 0 (straight) and 1 (fully bent). Thus, in this case, $V = \{i \mid 0 \leq i \leq 1\}$. As time progresses, we could sample the value of the sensor periodically, say once every tenth of a second, and we might sample the signal for 25 seconds, or a total of 250 samples. So, in this case $T = [0, 1, .., 249]$ with a simple mapping from a given element of T to "real time".

4.1.2 Stream

Let $c_1 : T_1 \rightarrow V_1$, $c_2 : T_2 \rightarrow V_2$ and so on. A stream is a sequence of channels S, such that:

$$S = [c_1, c_2, ..., c_n] \text{ s.t. } T_1 = T_2 = ... = T_n$$

n is the number of channels in the stream S. In other words, the channels must define values for the same period of time.

Intuitively, a stream represents monitoring a set of channels, rather than just a single one. To continue the above examples, what if we not only wished to sample forefinger bend, but also wanted to sample the bending of all the other fingers; and even wanted to sample information such as the orientation of the hand? A stream allows us to collect all these various measurements and treat them, in some ways, as part of the same "event". There is also an idea that the above mathematical representation does not make clear: for a given element of the (shared) domain of the channels, the value of the channel somehow represent measurements at the same "time".

4.1.3 Frame

Given the above definition, it is sometimes convenient to look at the values of all the channels of a stream at a given time. This is exactly what a frame is. For a given stream $S = [c_1, c_2, ..., c_n]$, the function fr is defined as:

$$fr : T_1 \rightarrow V_1 \times V_2 \times ... \times V_n$$
$$fr(t) = (c_1(t), c_2(t), ..., c_n(t))$$

Note that in the above, T_1 is used, but any other T_i could just as easily have been used.

Intuitively, a frame represents a "slice" of each of the channels at a given point in time. It represents the values of each of a channel for a given time t.

The connection between channels, frames and streams is illustrated in figure 4.1. In this diagram, we have three channels α, β and γ, with the range of the first two being the real numbers, and the range of the last being $\{r, g, b\}$. The stream consists of these three channels together. The "length" of the stream is 24 time-slices; in other words it consists of 24 frames. Each frame has three channels, and the domain of the function fr in this case is [0..23]. A frame is a "vertical slice" of the stream, for example, frames 13 and 17, each slice indicating the values of the channels at a given time. On the other hand, a channel is a "horizontal slice" of the stream, indicating the variation of some feature over time.

4.1.4 Stream Set

Let SS be a set of streams of the same type. Having the same type can be defined as follows:

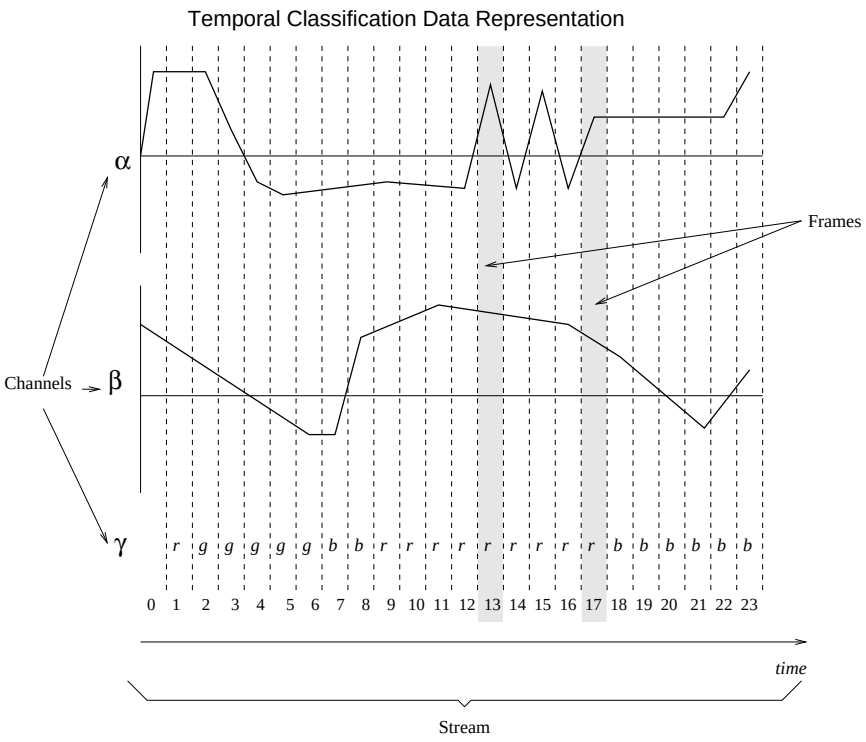


Figure 4.1: The relationship between channels, frames and streams.

Let $S_a = [c_{a_1}, c_{a_2}, ..., c_{a_n}]$ and $S_b = [c_{b_1}, c_{b_2}, ..., c_{b_n}]$. Then

$$SameType(S_a, S_b) \equiv \forall i \text{ s.t. } 1 \leq i \leq n \; V_{a_i} = V_{b_i}$$

So the stream set SS is one with the property that:

$$\forall S_i, S_j \in SS \;\; \Rightarrow \;\; SameType(S_i, S_j)$$

In other words, the range of each channel is the same for each element of SS. Intuitively, the set SS represents streams from a given domain source, for example, it may be the set of possible Auslan signs. Note that not all elements of SS are the same length – i.e. the duration (in frames) isn't necessarily the same.

4.2 Statement of the problem

Given these definitions, we are now in a position to define the temporal classification.

Let SS be a set of streams with the same type. Let CL be a set of labels of some kind, that describes the set of possible classes.

Define a function

$$class(S) : SS \rightarrow CL$$

which takes an element of SS and returns an element of CL.

The goal is given a subset of the function $class$ (say $class_T$) produce a function $class_P$ which is a similar to $class$ as possible[1].

Again, it is hard to represent the notion of "learning" here; and there are several intuitive aspects which the above definition does not cover. Again, the channel type means more than just that the channels have the same range; and SS is more than just a random collection of streams. SS in some ways represents a domain (in the machine learning sense) that we are interested in. It also remains for similarity to be defined. For example, if we are to continue the above example, one SS might be the set of all possible signs. $class(S)$ is not just a random function, but a function that tells us what the class, or type, of a given sign is.

Intuitively, our goal is: given a limited example of streams and their classes, in other words, some subset of the function $class$, say $class_T$, can we make an estimate of $class$, say $class_P$, that is close to $class$? This is similar to the definition for pure inductive inference.

4.3 Assessing success

In defining the above task, we have implied that in some way, the function $class_P$ should "approximate" the function $class$. What exactly do we mean by this, and how do we measure how closely the predicted class functions match the actual class functions?

In general we can't, unless $class_T = SS$. However, we can at least define the a theoretical measure for the accuracy.

On a single element S, one way to measure success would be to say that if $class_P(S) = class(S)$ then it is accurate and inaccurate otherwise.

This works for most cases. In some domains, however, the above is too simplistic – not all inaccuracies are of equal badness. Some errors may be worse than others. For example, consider working on a medical temporal classification application involving a diagnosis, where $CL = \{yes, no\}$, with "yes" indicating they have some condition and "no" indicating they do not. A "false positive" classification (i.e. misclassifying a negative as a positive)

[1] It is possible to imagine a more complex learning situation, which we might term strong temporal classification, where instead of mapping to CL, SS maps to CL^*. In other words, each stream does not map to a single class, but to a sequence of classes. Of course, this makes the learning task much more difficult. It would add the task of segmenting the stream, i.e. deciding which part of the stream is associated with each class label. This is one of the avenues of future research.

may not be as bad as a "false negative" (i.e. misclassifying a positive as a negative). In the sign language domain, misclassifying "bad" as "unwell" may not be as bad as misclassifying "bad" as "good".

To solve this problem, we introduce a function $cost(i, j) : CL \times CL \rightarrow [0..1]$ which tells us what the cost of misclassifying an i as a j. The function need not be i-j symmetric, i.e. $cost(i, j) \neq cost(j, i)$. Typically, of course, $cost(i, i) = 0$.

We can represent the above simple case (where all errors are equally bad) as:

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

Another complication is that sometimes it does not make sense to optimise for the whole space equally over the whole of SS. For example, it would be better to get higher accuracy on frequently occurring signs more than infrequent ones. So to give a more accurate measure of accuracy, this too must be included. We use the function $P_{SS}(S)$ to indicate the probability that a stream S has of occurring in the stream set SS.

Our goal can therefore be defined as finding:

$$\arg\min_{class_P} \sum_{S \in SS} cost(class(S), class_P(S)) P_{SS}(S)$$

In other words, we are trying to find the function $class_P$ which minimises the sum of the cost of misclassification times the probability of occurrence over the whole of SS.

4.4 Examples

4.4.1 Recognition of signs from a sign language

Let us consider one simple example of a temporal classification problem. Consider the task of recognising signs from a sign language[2] using instrumented gloves. The glove provides the information shown in table 4.1.

Each of these measurements represents a channel. A single capture of a sign is a stream. An example of a stream is shown in figure 4.2.

In the sign recognition domain, the set CL would consist of the names of different signs, like *come*, *name*, and so on. The set SS is the set of all possible signs, and we only have available to us a limited subset of SS.

[2]Note that recognising sign language as a whole is extremely difficult – to mention three 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; and improvisation, the creation of new signs based on other signs.

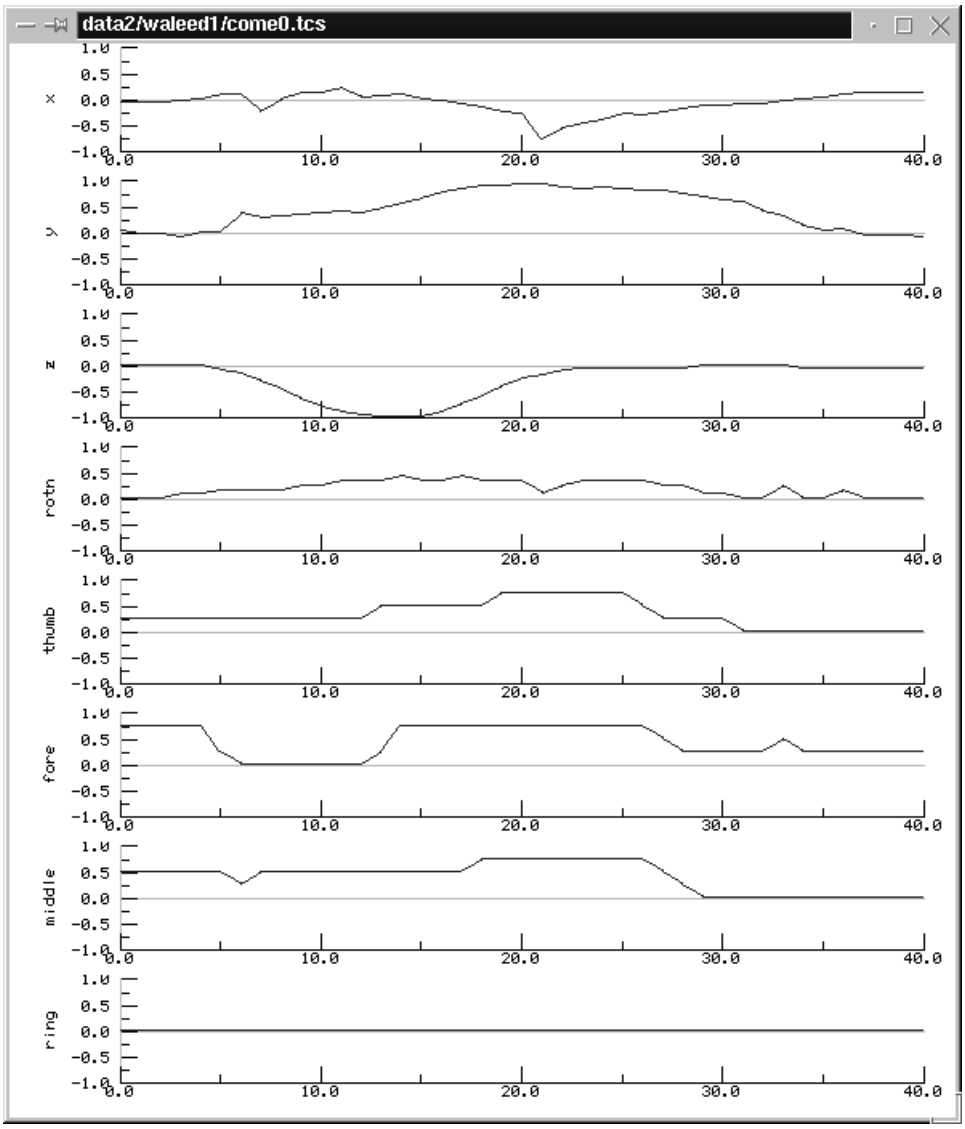


Figure 4.2: An example of a stream from the sign recognition domain with the label *come*.

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)

Table 4.1: Information from gloves

4.4.2 Blues and Reds

This example is used more as a pedagogical example in the rest of the paper. Consider a domain where the following is true:

- There is only one channel, say c.
- The range of the channel (i.e. V) is $\{T, F\}$.
- The set of classes for streams (i.e. CL) is {Blue, Red}.

Say that the subset of SS we are given for this domain is as shown in table 4.2.

Stream	$c(0)...c(t_{max})$ 0 1 2 01234567890123456789	Class
1	FFFTTTFFFFFF	Red
2	FFFTFFTFFTTTT	Blue
3	FFTFFTFFFFFFTTT	Blue
4	FFFFTTTTFFFFF	Red
5	FFFTTTFFFF	Red
6	FFTTFFTFFTTT	Blue

Table 4.2: The dataset for the Blues and Reds learning task

Our task now is to build a classifier that can tell blues and reds apart.

Chapter 5

General architecture

In this chapter, we discuss the proposed architecture for classification in temporal domains. We discuss the training architecture, which takes the examples given and tries to extract rules for describing time series, as well as the testing, or recognition architecture, which can take the outputs from the training architecture and apply them efficiently to unseen streams.

5.1 Overview

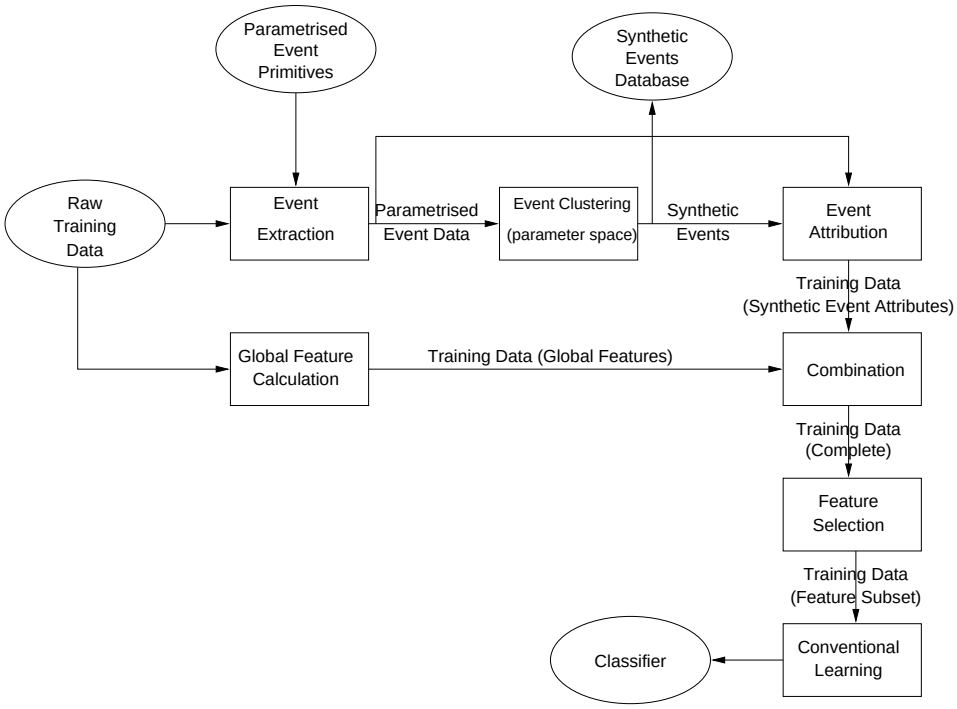


Figure 5.1: General architecture for training

Our input consists of training data consisting of a set of streams, as described in chapter 3. Associated with each stream is a label. We also have a description of the format of the data in terms of which channels there are for the particular domain.

These raw data are then fed into two processes: the global feature calculator and the event extractor. The global feature calculator extracts information about the stream as a whole which may prove useful for classification, but is not associated with a particular part of a stream. This may include measures like the maxima and minima of particular channels, the energy level of a channel and so on. In some domains, these features are very useful for a preliminary classification.

The second is the event extractor. This uses the parametrised event primitives provided to it to search through the stream for anything that fits the requirements of the primitives. The event primitives depend on the domain itself. Each parametrised event primitive is a particular shape or type of event that occurs in the given domain. There is no requirement that there be only one type of event primitive. Different types of events may be extracted simultaneously.

The output from the event extractor is a set of events, each being described in terms of which parametrised event primitive it is an instance of and the parameters obtained from the actual data.

For each of the different event primitives, clustering can be performed in terms of the parameters that describe that primitive's features. This then allows the creation of groups of events that have similar characteristics. We can then consider each of these clusters as a synthetic event, i.e. we can now look at our original training streams and check for the presence or absence of that particular synthetic event.

This is exactly what the event attribution stage does. It checks for the presence or absence of the synthetic events output from the clustering algorithm. Of course, the event attribution can be more complicated, and rather than giving a binary yes/no answer, it could give a confidence measure based on the probability it assesses that the particular training stream has a particular synthetic event. Thus the more confident the event attributor is that the training stream has an event belonging to a given cluster, the higher the score it gets.

The attributes coming from global feature calculation and from the extraction-clustering-attribution process are then combined. What we now have is a set of featuress, that for all intents and purposes, we can treat as a normal set of attributes. We then apply feature selection to select the best set of features (and also possibly to reduce the number of event primitives we are searching for) and apply conventional attribute-value learning techniques.

As a result, we can produce a classifier that we then use in the recognition architecture.

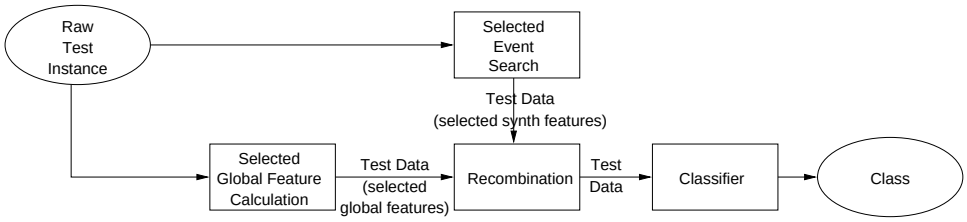


Figure 5.2: General architecture for testing

The recognition architecture uses similar, though not identical, components to those used in training. It uses global feature calculation as in the first case – the only difference is that for recognition, we need only calculate the particular global features that were found to be useful for classification. Similarly, we do a selective event search, i.e. we need only search for events that were found to be useful to the classifier. We recombine the data from both sources as before. We can then use the classifier we constructed by the training architecture, to finally get a single class label.

5.2 Training architecture

The training architecture is used to construct a classifier from examples and background knowledge. In figure 5.1, ovals represent inputs and outputs, and rectangles represent processes on data.

5.2.1 Raw training data

The raw training data are a stream set as defined in section 4.1.4 – that is, they are streams that are of the same type. Recall that the training data is defined as a subset T of SS for which the function *class* is known. For example, in the Auslan training example, the raw training data would consist of several examples of the signs we hope to classify. Each training example would be labelled with its class, i.e the result of applying the function *class* to the stream.

5.2.2 Parametrised event primitives

Parametrised event primitives (PEPs) are a way to incorporate background knowledge into the architecture. They define different "shapes" in the channels we are examining. A PEP consists of the following:

- A set of parameters $p = \{p_1, ...p_k\}$, which describe a particular parametrised event. Let P_1 represent the range of the parameter p_1. Let P be the

space of possible parameter values, i.e. $P = P_1 \times P_2 \times ... \times P_k$. Let P be known as the *parameter space*.

- A *finding function* f which takes a channel c and returns a set of parametrised events E. Each event in E is an element of P.

Each parametrised event primitive (PEP) is described by a set of parameters along with a function for extracting instances of that event primitive. Thus, by applying the finding function, we can find all the events that appear to be an instance of an event primitive we are considering. Example PEPs are discussed below.

Example PEP: A "straight-line" approximation for continuous-valued channels

Manganaris [Man97] and Pednault [Ped89] discuss techniques for converting a sequence of continuous values into a piecewise polynomial model – in other words, you provide a continuous valued function, much like a continuous channel in our representation, and it provides you with a set of polynomials, that together, approximate the channel. Each polynomial is specified in terms of the start time, the end time and the coefficients of the polynomial. The number of polynomials applied is not defined a priori, rather it is decided by the application of an MDL-style heuristic, as is the degree of the polynomial used over a particular range. For simplicity, we will consider only first-degree polynomials, i.e. straight lines.

For example, consider figure 5.3. The input channel, indicated by an unbroken line, can also be converted to a sequence of "straight-line" approximations, as indicated by the dashed lines. In this particular case, the original sequence of 40 data points, is reduced to four straight lines.

Each of these straight lines can be considered as an instance of a general event primitive which is a straight line.

Each straight line can be defined in terms of the following parameters:

- The time when it starts.
- The duration of the event.
- The average value of the line over its duration.
- The gradient of the line.

Note that this is not the only way to describe the line. The line could equally be described by: start value, end value, start time, end time. Also note that time has been explicitly encoded as part of the description of the event. Once we have created a description in terms of these parameters, we can

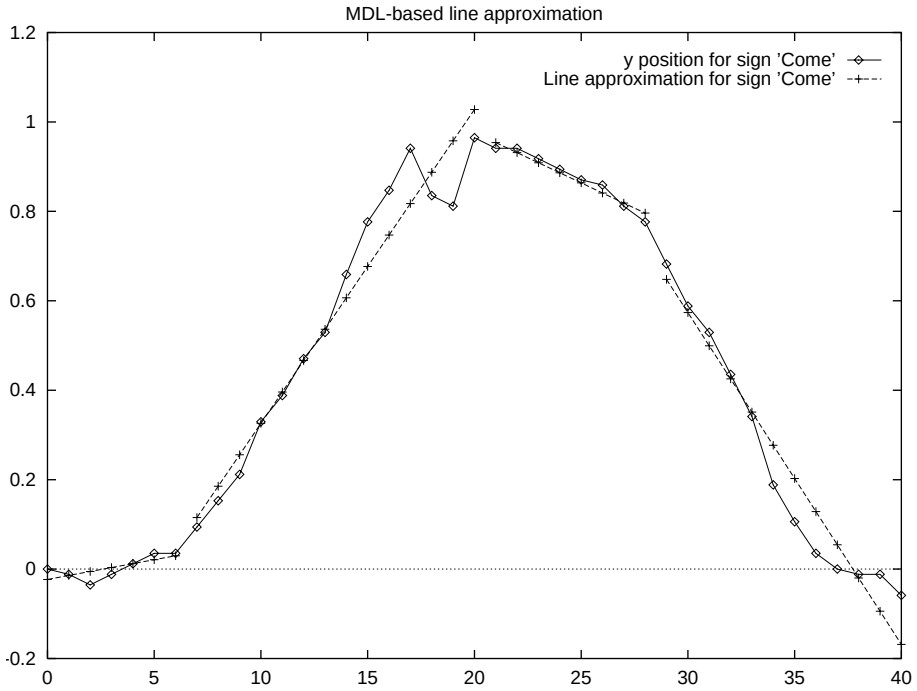


Figure 5.3: An example of a piecewise line approximation to some real data

Start time	Duration	Average Value	Gradient
0	6	0	0.008
7	13	0.57	0.08
21	7	0.89	-0.017
29	11	0.25	-0.07

Table 5.1: Line approximation parameters

make a table of these parameters. For example, the lines shown in figure 5.3 could be described as in table 5.1.

The finding function used above is based on a similar heuristic to that used by Manganaris [Man97]. Giving a closed-form definition of this finding function is difficult.

Example PEP: "Delta" detection for discrete-valued attributes

Consider now that we have a channel which is discrete, and in fact, binary - i.e. the value at each time frame is either T (true) or F (false); for example, the Blues and Reds domain as shown in table 4.2. Also assume that we know from our background knowledge that most of the time, the value is likely to be false – and that times when the value is true are very important. For example, this may be an indication of the failure of some sensor. So the types of events we may be interested in is events where the value goes from F to T.

In this case, the parameters would be:

- The time at which the value becomes true (t).
- How long it remains true (d).

In this case, it is easy to define our finding function

$$\begin{aligned} f(c) = \{(t,d) \quad | \quad & c(t-1) = F \\ \wedge \quad & c(t) = c(t+1) = ... = c(t+d-1) = T \\ \wedge \quad & t, t+1, .., t+d-1 \in \text{domain}(c)\} \end{aligned}$$

(Note the above is a little simplified and does not take into account the possibility of the sequence starting with a T or ending with a T).

So, for example, consider channel c from Stream 2 from table 4.2. Applying the function f to the channel will give us the set $\{(3,1),(6,1),(9,4)\}$.

These examples should serve to illustrate the general concepts of PEPs. It is also important to note that we need not, in a given domain, restrict ourselves to a single PEP, or analyse a channel with one PEP alone. In addition to the above, we could use other means as well. For example, we could apply a wavelet transform and pick the most significant coefficients of the wavelet transform and express them in terms of parameters. We could have a local maximum PEP as well, which is parametrised as the time of the maximum, the value of the maximum and the values and times of the nearest minima either side of the maximum. In fact we could have all of these simultaneously.

```
Inputs:
        T (training set)
        P (set of PEPs)
Outputs:
        E (output events)

procedure EventExtract(T, P, E)
        For each S_i ∈ T
                For each c_j ∈ S
                        For each p_k ∈ P
                                If approp(p_k, c_j)
                                        E = E ∪ (i, j, f_{p_k}(c))
                        End
                End
        End
End
```

Figure 5.4: Event extraction algorithm

5.2.3 Event extraction

The event extraction mechanism takes these PEPs and applies them to the training instances. The event extraction algorithm is shown in figure 5.4.

The algorithm takes each stream, and each channel in each stream, and then applies each appropriate PEP to that channel. The results are then added to the event list E for later use. f_{p_k} is the finding function of the PEP p_k.

Not all PEPs are appropriate for all channels, so we must use our domain knowledge to decide which PEPs should be applied to which channels. Thus approp(p, c) returns true if PEP p can be appropriately applied to channel c. For example, if we know that a channel is highly noisy, it doesn't make sense to apply a local maximum PEP, since it's not likely to pick up salient features of the data so much as random noise. Similarly, it does not make sense to apply a "delta" PEP to a continuous channel, or a "straight line" approximation to a discrete channel.

The result of the operation is that E now holds a set of tuples, each tuple consisting of some identification as to which stream and which channel it belongs to.

If we were to apply this to the Blues and Reds task, we would get the data shown in table 5.2.

5.2.4 Global features

Another technique for extracting information is global feature calculation. This would cover features of a stream that are not localised. For example

Stream	Events found
1	$\{(3,3)\}$
2	$\{(3,1),(6,1),(9,4)\}$
3	$\{(2,1),(5,1),(12,3)\}$
4	$\{(4,4)\}$
5	$\{(3,3)\}$
6	$\{(2,2),(6,1),(9,3)\}$

Table 5.2: Event extraction applied to the Blues and Reds domain

for continuous channels, the global maximum and minimum value that each channel takes; the average value of each channel and so on. They may be more complicated. For example, we could count the number of local maxima and minima, we could measure the total "distance" covered by each channel, or use some measure of energy. For discrete binary channels, it might be something like how many changes there are, the percentage of time for which the value is true and so on.

Note that global features may be associated with either a single channel or more than one channel. For example, in the Auslan domain, we could measure the distance using a Euclidean metric on the x, y and z motions, or on each channel separately. For simplicity below, we consider global feature calculation for a single channel.

Example global attributes

For a given channel, we could define the global maximum 'max' for a given channel as

$$\max(c) = m \text{ s.t. } \forall t\ c(t) \leq m, t \in \text{domain}(c) \wedge \exists t \text{ s.t. } c(t) = m$$

Similarly, we could define a distance measure on a single channel as:

$$dist(c) = \sqrt{\sum_{t=1}^{t_{max}} c(t)^2 - c(t-1)^2}$$

These global features can be useful for classification; in fact, in the past this has been one of the ways of doing temporal classification using existing classification tools. For example, with the sign language recognition domain, approximately 30 per cent accuracy can be obtained by looking at the minima and maxima of the x, y and z axes alone [Kad95]. However, they are usually not sufficient for doing high-accuracy classification. In addition, they are usually insufficient for producing meaningful descriptions.

In the Blues and Reds domain, a simple global variable we could use would be the fraction of the time that the channel c is true.

5.2.5 Parameter clustering

Now we have these descriptions of various events, what can we do with them? Since each PEP is described in terms of a set of parameters, we can now do clustering in the parameter space. For example, if we look at the "delta" PEP, we can do clustering in two dimensions to find groups of event primitives which have similar parameter values, and thus have similar characteristics. For example, if we take the data from table 5.2 and plot them (as shown in figure 5.5) then we can see that we might be able to form several possible clusters in the parameter space to represent different events. Each of these clusters represents a particular type of event, which we will term a synthetic event.

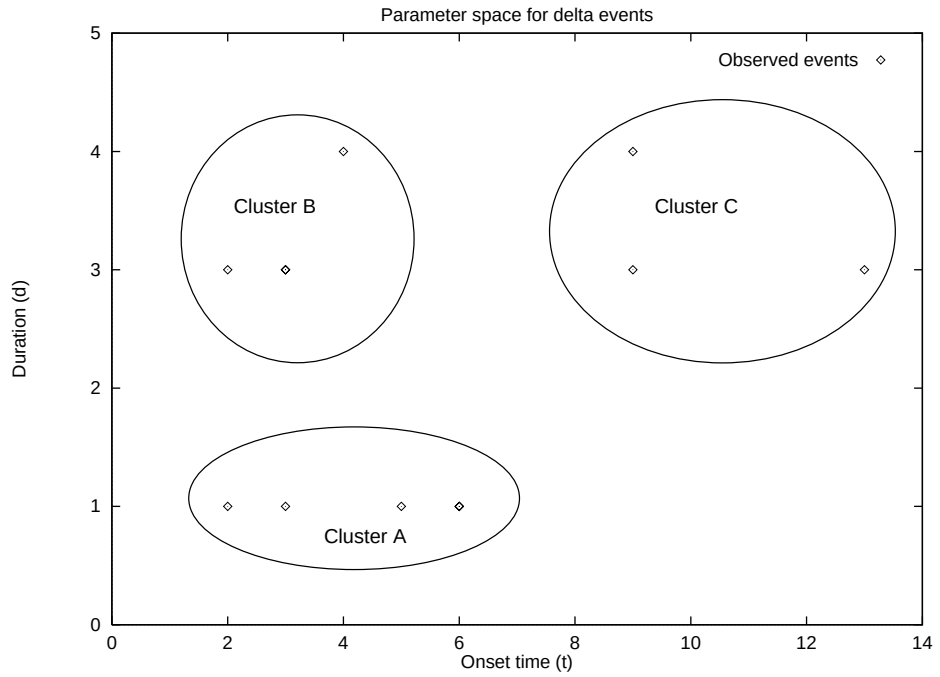


Figure 5.5: Parameter space and clustering for Blues and Reds domain.

Thus what we will get from the clustering is some set of clusters representing different types of events, even though they are of the same type of primitive.

5.2.6 Event attribution

The event attribution stage uses the synthetic events created from the parameter clustering and then looks at each training stream to see if it has an event that is similar to the cluster we are considering.

Consider the Blues and Reds domain. We can now ask whether any of the training streams have events corresponding to clusters A, B and C as shown in figure 5.5. The results are shown in table 5.3.

It is possible to be more general and clever. Depending on the clustering algorithm, we may retain information from the parameter clustering that would provide information about the types of variations we expect. For example, we might be able to deduce that a delta event of (6,2) is still, most probably, a member of the cluster A. Thus, rather than using a simple true/false result, we might instead be able to estimate the confidence that a given stream has an event belonging to cluster A.

Thus, we can treat each of the clusters as a synthetic event, and we can evaluate the confidence for each stream that it has an event corresponding to that attribute.

5.2.7 Recombination

The recombination step is a very simple one. As a result of the event attribution stage, we have a set of attributes, each being a continuous value indicating confidence in the presence or absence of a particular event, and we also have a set of global attributes that we extracted as well. Combining them is simply an issue of combining the different attributes extracted by each process and ensuring they are associated with the correct class label.

Stream	**Globals**	**Event Attributes**		
	Ratio of True	**Cluster A**	**Cluster B**	**Cluster C**
1	0.25	No	Yes	No
2	0.46	Yes	No	Yes
3	0.33	Yes	No	Yes
4	0.30	No	Yes	No
5	0.30	No	Yes	No
6	0.50	Yes	No	Yes

Table 5.3: Global and event attributes for the Blues and Reds domain.

The output of the recombination stage is a training set of attribute-value instances that a conventional attribute-value learner can use. Some of the attributes are global attributes, and the remainder are "event-present" attributes which are the confidence that the training instance has a particular event present in it.

5.2.8 Feature selection

We now have a relatively large set of features to choose from. At this stage, we can apply feature selection techniques to reduce the number of features that the learner must deal with. Of course, it is also possible to integrate the feature selection and learning stages, as for example occurs using "wrapper model" feature selection [JKP94].

```
Rule 1:
        clusterA = Yes
        ->  class blue
Rule 2:
        clusterA = No
        ->  class red

Default class: blue
```

Figure 5.6: Rules for telling blues and reds apart.

5.2.9 Conventional attribute-value learning

This does attribute-value learning once the features have been selected. Almost any conventional learner can be used. For example, we may apply a rule-learning system to the dataset, a decision tree learner or any other attribute-value learner. We applied `c4.5rules`, a simple rule-learning system [Qui93], to the Blues and Reds domain. It came up with the rules shown in figure 5.6.

As can be seen, the rule is very simple. In general, however, this will not always be the case.

5.2.10 Classifier

The classifier consists of a data structure that when given attributes, returns their classes. The attributes it is given consist of both global attributes and some that are the result of the event clustering step.

5.3 Testing architecture

The testing architecture is shown in figure 5.2. It is used to classify unseen test instances.

5.3.1 Raw test instance

We now demonstrate the classification of unseen instances. Consider the test instances in table 5.4 for the Blues and Reds domain. Each of the streams is taken one at a time and its class determined.

5.3.2 Selected event search

Once we have built the classifier as above, and also know which clusters are likely to be important, we can search for events which belong to the

Stream	$c(0)...c(t_{max})$ 0 1 2 01234567890123456789
1	FFFTTTFFFFFF
2	FFFTTFFTTT
3	FFTFFTFFFFTT

Table 5.4: Test instances for Blues and Reds training data

clusters we are interested in. For example, from the rules shown in figure 5.6, we know that we only need to use cluster A to do the classification, thus there is no point looking for potential elements of clusters B or C, as they are not necessary for the classification made by the rule. This allows us to efficiently implement our event search. Of course, if time is not an issue, then you can default to extracting all possible events. However, by using this selective event searching, we can use our expectations to minimise the processing power required to do classifications for more complicated tasks, containing more than one channel and typically more than two classes.

The same issues that arose in event attribution occur here: namely, how do we assess whether a particular event has occurred? We can in fact use the event attribution process in place of the selected event searching process and the system will still function correctly.

5.3.3 Selected global feature calculation

Again, if the classifier uses features that are global, then these too can be calculated. Note that we only need to calculate features used by the classifier. If it doesn't require the features, then we need not calculate them.

In the case of the Blues and Reds domain, it isn't necessary to calculate any global variables.

5.3.4 Recombination

As before, we recombine the data from the global feature calculation and the event search. The results of applying these to test data of table 5.4 are shown in table 5.5, along with the classification results.

5.3.5 Classifier

We can now apply the classifier (figure 5.6) that was produced by the training stage once again, to finally give us a class. For the three instances we were given we can apply the techniques to get the results shown in table 5.5.

Stream	Cluster A	Predicted class
1	No	Red
2	Yes	Blue
3	Yes	Blue

Table 5.5: Results of testing on dataset

5.4 Goal evaluation

In this section, we try to evaluate in a general sense, how well this architecture allows us to meet our goals.

5.4.1 Generality

The architecture appears to be general enough to apply in many domains. We can provide domain-specific knowledge through:

- Selecting appropriate PEPs (parametrised event primitives) for the domain. For example, in some domains we may know that a particular type of event is likely to occur – for example, humps, peaks, changes in level, sudden bursts of noise. We can take advantage of this. Furthermore, we are not constricted to one single PEP, we can use several different ones and choose the ones that perform the best. If we do not have domain knowledge, we have several "fallback" PEPs that may be useful, such as piecewise polynomial models, maxima and minima etc. Preliminary results indicate that these simple PEPs work well as defaults.
- Selecting the clustering algorithm used. We may use domain knowledge about the parameter space that is likely to occur to adjust the parameters of an algorithm, or select an appropriate algorithm[1].
- Selecting the appropriate feature selection and classification algorithms.

5.4.2 Classification accuracy

It is difficult to determine classification accuracy at this point, without looking at a given system. Obviously, it does depend on the following factors: the extent to which the extracted events are relevant to classification, how effective the event clustering algorithm is in picking up shapes that are really representative of underlying types of events and how effective the classifier is in making use of the extracted features.

[1] In fact, although not discussed in the model so far, due, perhaps to its impracticality, it is possible to run multiple clustering algorithms as well and use the synthetic events generated by each.

5.4.3 Meaningful descriptions

This is not easy to determine a priori. At a minimum, meaning can be extracted in the following ways: firstly, the features that are selected by the feature selector indicate which event clusters and global features are useful, hence this tells us something about which parts of a stream are important for classification purposes. Secondly, if the learner produces intelligible output, such as a rule builder, we can find out how the various events are related and this build an understandable model by combining the rules with the descriptions of the synthetic events. For example, it would be possible to convert from a rule description and the cluster information back into a prototype or set of prototypes for each class.

5.4.4 Fast learning curve

The speed of learning is difficult to assess in general, because of the number of factors that are interacting.

5.4.5 Time

Obviously, this depends on individual learners used. But since there are no iterative loops in the architecture, our time will only be the sum of the time required to complete each component of the architecture.

Testing time is significantly less than training time. Firstly, usually only a small subset of features needs to be extracted; rather than all possible features. Secondly, no clustering is done as the data arrives. Thus we can see that this architecture is biased towards speed of recognition, rather than training.

Chapter 6

Example application

As an example implementation of this architecture, a very simple example was developed which illustrates some of the concepts. Its application is then demonstrated on Australian Sign Language (Auslan). It is meant as a preliminary example, rather than evidence of the architecture's utility.

We discuss the whole system and then show some practical results.

6.1 Domain information

Auslan is the language of the Australian Deaf community. It is considered by researchers to be a dialect of British Sign Language (BSL) and is also closely related to New Zealand Sign Language (NZSL). Auslan has approximately 4000 to 5000 "well-defined" signs, but this underestimates the richness of the language, since fingerspelling can be used for any spoken word that doesn't have a sign and also because the visual medium can be employed in ways that the aural can not (e.g. in the detailed description of the appearance of objects). There are approximately 15,000 Auslan signers. A sign consists of a number of physical components, such as the handshape, location, palm orientation, movement of the palms and facial expression. Facial expression is minor in the formation of individual signs, but is fundamental in the construction of phrases [Joh89].

A small selection of isolated Auslan signs were captured using an instrumented glove called the PowerGlove. This glove is relatively primitive, providing only low-resolution information about position and finger bend of the the first four fingers and roll of the hand, and failing altogether to give any information about pitch, yaw and the bending of the little finger[1]. In addition, it only provides information about one hand, not both.

[1] The little finger is actually important in many sign languages. For example, in Auslan and British Sign Language, a fist with the little finger extended indicates *bad*. It is also used as a modifier for other signs that indicate badness - e.g. *sick* is a *bad* handshape against the body and *swear* is a *bad* handshape starting at the mouth and moving away.

Still, it is possible to perform some recognition using these data. In previous results [Kad95], it was shown that by manually selecting a set of global attributes and extracting them and trying different learners, recognition with accuracies up to approximately 80 per cent on seen signers could be achieved with a vocabulary of 95 signs. These signs were intentionally selected to be representative[2]. Its performance on unseen signers (i.e. trained on four signers, tested on fifth) was not so good, achieving only approximately 30 per cent accuracy.

A small subset of these signs (ten) were selected from a single signer for testing purposes. A total of 20 examples of each sign were used, with 15 of these used for training instances and 5 for testing.

6.2 Event extraction and global attributes

The only PEP used in these results is the straight-line event extractors discussed in section 5.2.2. No global attributes were applied.

6.3 Clustering algorithm

A k-means clustering algorithm was used initially used over the whole domain; one clusterer per channel. The k-means algorithm is shown in figure 6.1.

However, there are problems with such a technique: k-means requires the number of clusters to be specified beforehand. Determining the number of clusters is not easy. In addition, a quick check of the parameter space indicated that it appeared that clustering over all data would not be productive, since there did not appear to be any obvious clustering of the space. For example, figure 6.2 shows two dimensions (of four) of the parameter space of the events on the y channel, with the points coming from all ten classes. It is also difficult to determine the initial choice of centroids C.

So, as an alternative, we used per-class k-means clustering. For each class and each channel we clustered the events that came from that class only. Firstly, determining the number of clusters can be done by taking the average number of events over the set of training instances belong to the class; secondly, because things belonging to the same class are likely to have similar events, clustering is more likely to succeed. For example, one of the signs considered for learning was the sign *come*. Looking at the same two dimensions as we did for figure 6.2, but only plotting events belonging to five examples from the class *come* produces 6.3. The five examples had 5, 3, 3, 5 and 4 events on the y channel, with an average of four events. Thus

[2]This, of course, makes recognition more difficult. For example, approximately 10 per cent of the signs were two-handed, even though we only had one glove. Several used little finger information.

Inputs:

$P = \{p_1 ... p_k\}$ (Points to be clustered)

n (Number of clusters)

Outputs:

$C = \{c1..c_n\}$ (cluster centroids)

$m : P \rightarrow \{1..n\}$ (cluster membership)

procedure **KMeans**

Set C to initial value (e.g. random selection of P)

For each $p_i \in P$

$m(p_i) = \arg\min_{j \in \{1..n\}} distance(p_i, c_j)$

End

While m has changed

For each $i \in \{1..n\}$

Recompute c_i as the centroid of $\{p | m(p) = i\}$

End

For each $p_i \in P$

$m(p_i) = \arg\min_{j \in \{1..n\}} distance(p_i, c_j)$

End

End

End

Figure 6.1: K-Means algorithm

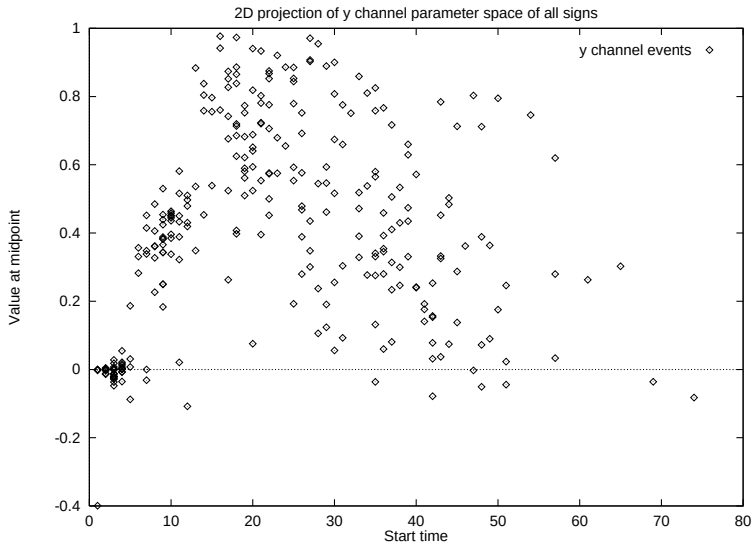


Figure 6.2: Parameter space with all classes

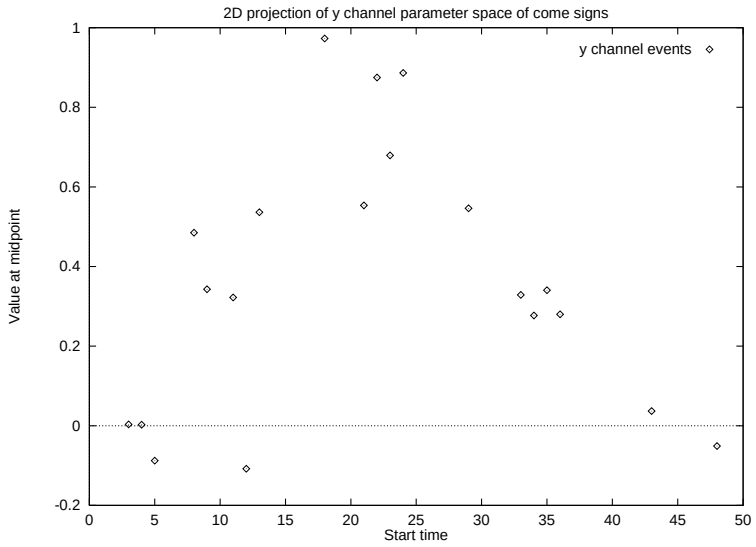


Figure 6.3: Parameter space with *come* class only

the algorithm was told to make four clusters. Finally, C can be guessed by doing an initial clustering based on the order of the events from each instance (so all the "first" events go in one cluster, all the "second" events go in one cluster) normalised by the number of events, and then computing the centroid.

6.4 Event attribution

The output of the k-means algorithm is a set of clusters, each cluster described in terms of its centroid (described as a point in the parameter space) and the points that belong to cluster. The standard deviation for each dimension of each cluster is also computed.

It is possible to make a rough estimate of the confidence of membership of a particular event to a given cluster. This estimate is based on the following assumptions:

- The distribution of the points belonging to the cluster is Gaussian, with the mean of the Gaussian in each dimension being the centroid of the cluster.
- The distributions of the points along each dimension of the feature space are independent of one another.

Our estimate of confidence is based on calculating the probability that an element of this cluster could be as far or further out from the centroid of this cluster as the given event.

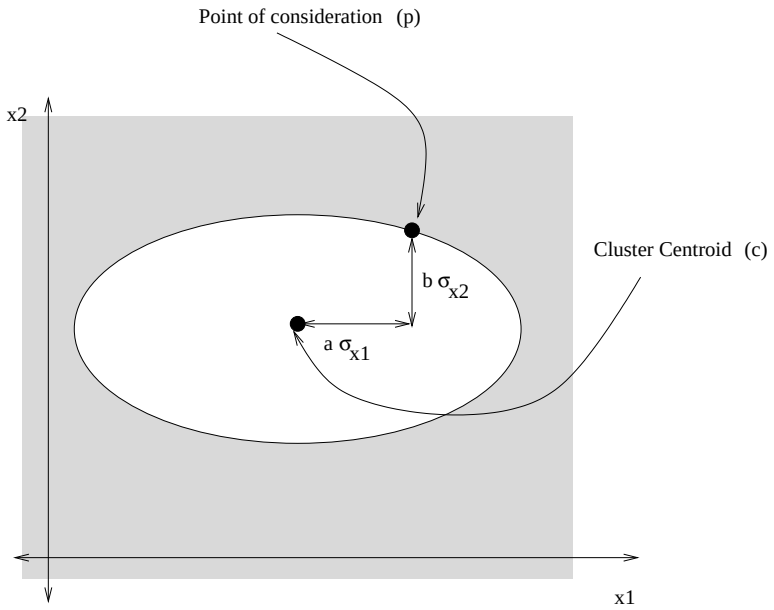


Figure 6.4: An example illustrating how the confidence estimate can be made

In figure 6.4[3] let $x1$ and $x2$ be two dimensions of the parameter space. The probability we are trying to estimate is the probability that a point in the shaded area belongs to the cluster. Let σ_{x1} be the standard deviation of $x1$ and σ_{x2} be the standard deviation of $x2$. We can express the difference along the $x1$ axis between p and c as a multiple of the standard deviation, so that $p_{x1} - c_{x1} = a\sigma_{x1}$. Similarly, we can express the distance along the $x2$ axis as $p_{x2} - c_{x2} = b\sigma_{x2}$. If $x1$ and $x2$ are independent and the cluster has a Gaussian distribution, then we can estimate the probability that an element belonging to the cluster is in the grey area as:

$$P(|Z| \geq a) \times P(|Z| \geq b)$$

where Z is the standard normal distribution (i.e. a normal distribution with a mean of zero and a standard deviation of 1).

In other words, it's the probability that the point of consideration is more than a standard deviations away from the centroid in the $x1$ dimension and more than b standard deviations away from the centroid in the $x2$ dimension. The values of $P(|Z| \geq a)$ can be found easily using table lookup. This method can be generalised to higher dimensions.

To understand how this works, consider some examples: imagine we have

[3]Note that while the boundary between the shaded and the white area is drawn as an ellipse, this is not the correct shape. The shape (for which a closed form is extremely difficult to compute) is the locus of all values of $(c_{x1} + a'\sigma_{x1}, c_{x2} + b'\sigma_{x2})$ such that $P(|Z| \geq a') \times P(|Z| \geq b') = c$, where $c = P(|Z| \geq a) \times P(|Z| \geq b)$.

an event that matches the centroid of the cluster exactly. Then $a = b = 0$. But $P(|Z| \geq 0)$ will occur with a probability 1. Thus our confidence that an event that matches the centroid of the cluster belongs to the cluster is 1. This is intuitive, since the centroid can be thought of as the most obvious example of an element belonging to the cluster.

Now consider what happens when p is σ_{x1} away from c in the $x1$ dimension and σ_{x2} in the $x2$ dimension. From a statistics text it can be determined that $P(|Z| \geq 1) = 0.3174$. Thus the probability that a point lies $(\sigma_{x1}, \sigma_{x2})$ or further away from the centroid, but still belongs to this cluster is $0.3174^2 \simeq 0.1$. Alternatively, a point which is $2\sigma_{x1}$ away in the $x1$ dimension, but has the same $x2$ value as the centroid will have a confidence of $P(|Z| \geq 2) \times P(|Z| \geq 0) = 0.0456 \times 1 = 0.0456$.

For convenience, we take the logarithm of the confidence. This simplifies the mathematics (since multiplication becomes addition) and also avoids underflow problems (with very small probabilities) when these techniques are implemented using floating point operations. Thus, since the range of confidence is [0..1], the range of the logarithm of the confidence is $[-\infty..0]$.

Again, we use a per-class technique. Thus we only check for the presence or absence of clusters coming from the appropriate class. For each training instance and for each cluster, we find the event (if there is one) in that training instance which has the highest probability of belonging to that cluster. This becomes our estimate of the confidence that the training instance has an event belonging to that cluster.

6.5 Feature selection and learning

We did not do any feature selection, other than that built into the two algorithms. We considered two different learners, but in adherence with the rest of the system, we found that a per-class technique was practically most effective. Thus for each class, we created a binary learner. In addition, each learner could only use synthetic events generated by the clustering of the instances of that class. For example, a cluster from the class *bad* was not used as an attribute for learning the definition of the class *come*. Thus for each class, the learner was used to build a binary classifier that returned true if the instance was a member of the class and false otherwise.

The two learners we considered were:

- A naive Bayes learner [Mit97] – a learning technique which assumes that all the attributes are conditionally independent and makes an estimate of the probability that a particular training instance belongs to the class as the product of the probabilities that the each value came from the class. Each class learner predicted the probability that the test instance was a member of the class. The binary classifiers were combined by looking for the classifier that returned the highest

probability that the instance was a member of its class.

- C4.5 [Qui93], a decision tree builder. Each class learner was asked to classify whether each test instance belonged to it or not. The classifiers produced by this technique were combined in the following way: If no binary classifier "claimed" the test instance, it was unclassified. If one binary classifier claimed the test instance, it was classified as that class. If more than one binary classifier "claimed" the test instance, the classifier that claimed to have the lowest error rate got it. In addition, the decision trees generated by C4.5 were analysed and considered to see if they produced intelligible results.

6.6 Recognition

Recognition was achieved by applying the event attribution process to the test data. The test data were then fed into both of the classifier used above. Note that the test data were a complete holdout set; i.e. it was not used for training or clustering.

6.7 Results

The naive Bayes learner attained an accuracy of approximately 76 per cent. It seemed that some classes were classified far more accurately than others.

The C4.5 learner produced the correct classification 42 per cent of the time, the incorrect classification 16 per cent of the time, and no classification 42 per cent of the time. This high number of "unclassifieds" may be caused by the uneven distribution of positive and negative instances to the learners. Since there are ten classes, only 10 per cent of the instances will be positive and 90 per cent will be negative. Faced with this, C4.5 favours negative classifications as a default.

More interesting was the results obtained by looking at the trees produced for each class. For example, consider the sign *come*. The tree produced is shown in figure 6.5. This particular tree had approximately 80 per cent accuracy. It was also observed that the trees produced in this way were very small. The smallest trees had five nodes and the largest trees had nine nodes, which is within the bounds of human comprehension.

Note that `come-z-2, come-y-1` and `come-fore-5` are names for clusters; the parameters of which are shown in table 6.1. Recall also that the numbers in table 6.1 represent the logs of the confidences we determined for our clusters.

Surprisingly, this is actually understandable when one considers the sign for *come*. It is a moving of the hand initially away from the body, towards a person standing in front of you. Then the hand is brought towards the body,

```
come-z-2 > -5.48763 : come
come-z-2 <= -5.48763 :
|   come-y-1 <= -2.01513 : not-come
|   come-y-1 > -2.01513 :
|   |   come-fore-5 <= -6.26429 : not-come
|   |   come-fore-5 > -6.26429 : come
```

Figure 6.5: Decision tree produced for *come*

Cluster	Duration	Value at midpt	Start time	Gradient
come-z-2	10.3	-0.22	18.6	0.1
come-y-1	12.2	0.2	14.4	0.05
come-fore-5	2.7	0.23	25.5	0.19

Table 6.1: Clusters used by C4.5 to classify *come*.

usually with a slight upwards motion, finally closing the finger.

By looking simultaneously at the clusters and the decision tree, the above can roughly be translated as: If the person moves his hand away from himself rapidly and towards someone standing in front of him, then it is a *come* sign. Otherwise see if there is a medium speed upwards motion. If that upward motion ends with the finger being bent, then it is a *come* sign. Otherwise it is not.

The above can also be represented in a graphical form. This could be accomplished in the following way:

- Start with a blank window, much like figure 4.2, but with no channels drawn on it.
- For each leaf node labelled *come*, trace the path from the root to that node. Take only those nodes that check for the presence of a synthetic event (i.e. contain '>' confidence comparisons, in other words, check that the event is present with a confidence greater than a certain amount, rather than '<' comparisons, which are checks for the absence of a particular event with a certain confidence).
- Take the sequence of synthetic events generated in the previous steps, look up their parameters in a table like table 6.1 and do the reverse of the parametrisation process: convert the event back into a frame-channel representation. For example, from `come-z-2`, we could deduce that channel z of frame 19 has a value of -0.8, going up to a value of 0.3 by frame 28. Draw these values onto the window.
- Also show any appropriate global features.

- Repeat the process for each node labelled *come*.

What we now have is a set of prototypes for the *come* sign. In the example shown, we would have two prototypes: one containing only the `come-z-2` synthetic event and the other indicating the `come-y-1` and `come-fore-5` synthetic events.

6.8 Conclusions

These results and techniques are meant to be a simple illustration of how some aspects of the general architecture presented can be applied to a real domain. It does not illustrate the use of different PEPs, or global feature calculation. In addition, the selected event search is currently implemented in exactly the same way as the event attribution process. While the accuracy results aren't particularly spectacular, the descriptions generated of the classes are certainly interesting and already showing promise. Most probably as the system is refined and more PEPs are added, then the accuracy (God-willing) will improve.

Chapter 7

Conclusions and future work

7.1 Future work

There is much work to be done to improve the performance of the architecture. These are discussed below.

7.1.1 Short term

More results, more PEPs, better software

The immediate future work involves applying this technique to a variety of domains. Further investigation involves consideration and analysis of what PEPs are appropriate for given domains and which clustering and learning algorithms work under which circumstances.

The software being used for evaluation, called TClass, is also being rewritten. In particular the primary design goal is to have an expandable system, so that different PEPs, clusterers and learners can be added as painlessly as possible[1].

7.1.2 Long term

Automatic PEP selection and/or generation

What PEPs are appropriate for a given domain? Using the current architecture we can get a very crude guide to that information by considering the features selected at the feature selection and classification stages. However, it doesn't really use all of the data.

[1] The software is being written in Java to simplify expandability and portability, though it does suffer a performance penalty by doing so. Once completed, the source will be released publicly with the explicit goal of allowing and facilitating others adding their own PEPs, clusterers, feature selectors and learners.

Perhaps it is possible to have a library of PEPs available and by analysing the training streams to determine which of the PEPs are appropriate. It might be considered how well each PEP models the data.

Further down the line, it may also be possible to consider techniques for looking at a domain and *extracting* the PEPs for that domain from training data.

Making more use of temporal correlations

In section 3.2.1, it was discussed that temporal concurrency was very indicative of uniqueness of a class. The current architecture does not make full use of this. It might be possible to have a more complex PEP model, or to have meta-PEPs (parametrised events representing relationships between other events).

Provable theoretic properties

The theoretic properties of such learners appears to be very difficult to work on because of the unconstrained number of attributes; in addition to the complexity of having several components, several of which are difficult to prove anything about individually. It may still be possible to at least prove some properties.

7.2 Conclusions

An architecture for temporal classification has been developed which appears to have the potential to overcome some of the disadvantages of existing approaches for temporal classification: the amount of fine tuning required, the amount data required to learn from and the production of meaningful descriptions of the concepts that have been learnt. Preliminary results are promising, though not conclusive. Further investigation of the architecture is definitely warranted and is proceeding.

Bibliography

[Ben96] Yoshua Bengio. *Neural Networks for Speech and Sequence Recognition*. International Thomson Publishing Inc., 1996.

[Dan98] Andrea Danyluk. Predicting the future: AI approaches to time-series problems. Technical Report WS-98-07, AAAI Press, 1998.

[DLM+98] Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. Rule discovery from time series. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*. AAAI Press, 1998.

[DM86] T. G. Dietterich and R. S. Michalski. *Machine Learning: An Artificial Intelligence Approach, Volume II*, chapter Learning to predict sequences, pages 63–106. Morgan Kaufmann, Los Altos, CA, 1986.

[FMR98] Nir Friedman, Kevin Murphy, and Stuart Russell. Learning the structure of dynamic probabilistic networks. In *Proceeding Uncertainty in Artificial Intelligence Conference 1998 (UAI-98)*. AAAI Press, 1998.

[HHS98] Michael Harries, Kim Horn, and Claude Sammut. Extracting hidden context. *Machine Learning*, 32(2), August 1998.

[JKP94] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the International Conference on Machile Learning 1994*, pages 121–129, 1994.

[Joh89] Trevor Johnston. *Auslan Dictionary: a Dictionary of the Sign Language of the Australian Deaf Community*. Deafness Resources Australia Ltd, 1989.

[Kad95] Mohammed Waleed Kadous. GRASP: Recognition of Australian sign language using instrumented gloves. Honours thesis, School of Computer Science and Engineering, University of New South Wales, 1995.

[KM87] K. Kumar and A. Mukerjee. Temporal event conceptualization. In *Proc. of the 10th IJCAI*, pages 472–475, Milan, Italy, 1987.

[KP98] Eamonn J. Keogh and Michael J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *Predicting the Future: AI Approaches to Time-Series Problems* [Dan98], pages 44–51.

[Man97] Stefanos Manganaris. *Supervised Classification with Temporal Data*. PhD thesis, Computer Science Department, School of Engineering, Vanderbilt University, December 1997.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[MR81] C. S. Myers and L. R. Rabiner. A comparative study of several dynamic time-warping algorithms for connected word recognition. *The Bell System Technical Journal*, 60(7):1389–1409, September 1981.

[MTV95] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, pages 210–215, 1995.

[OJC98] Tim Oates, David Jensen, and Paul R. Cohen. Discovering rules for clustering and predicting asynchronous events. In *Predicting the Future: AI Approaches to Time-Series Problems* [Dan98], pages 73–79.

[Pal97] Georgios Paliouras. *Refinement of Temporal Constraints in an Event Recognition System using Small Datasets*. PhD thesis, University of Manchester, 1997.

[Ped89] Edwin P. D. Pednault. Some experiments in applying inductive inference to surface reconstruction. In *AAAI Conference Proceedings*, pages 1603–1608. AAAI, 1989.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[RC98] Michal T. Rosenstein and Paul R. Cohen. Concepts from time series. In *AAAI '98: Fifteenth National Conference on Artificial Intelligence*, pages 739–745. AAAI, AAAI Press, 1998.

[RJ86] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE Magazine on Accoustics, Speech and Signal Processing*, 3(1):4–16, 1986.

[RM86] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing: Explorations in the microstructure of Cognition*, volume 1. Foundations. MIT Press/Bradford Books, 1986.

[SM95] Yuval Shahar and Mark A. Musen. Knowledge-based temporal abstraction in clinical domains. Technical report, Stanford University, 1995.

[Wid96] Gerhard Widmer. Recognition and exploitation of contextual clues via incremental meta-learning. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 525–533. Morgan Kaufmann, 1996.

[ZR98] Geoffrey Zweig and Stuart Russell. Speech recognition with dynamic Bayesian networks. In *Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 173–180. AAAI Press, 1998.