An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback

Eamonn J. Keogh and Michael J. Pazzani

Department of Information and Computer Science
University of California, Irvine, California 92697 USA
{pazzani,eamonn}@ics.uci.edu

Abstract

We introduce an extended representation of time series that allows fast, accurate classification and clustering in addition to the ability to explore time series data in a relevance feedback framework. The representation consists of piece-wise linear segments to represent shape and a weight vector that contains the relative importance of each individual linear segment. In the classification context, the weights are learned automatically as part of the training cycle. In the relevance feedback context, the weights are determined by an interactive and iterative process in which users rate various choices presented to them. Our representation allows a user to define a variety of similarity measures that can be tailored to specific domains. We demonstrate our approach on space telemetry, medical and synthetic data.

1.0 Introduction

Time series account for much of the data stored in business, medical, engineering and social science databases. There are innumerable statistical tests one can perform on time series, such as determining autocorrelation coefficients, measuring linear trends, etc. Much of the utility of collecting this data, however, comes from the ability of humans to visualize the *shape* of the (suitably plotted) data, and classify it. For example:

- Cardiologists view electrocardiograms to diagnose arrhythmias.
- Chartists examine stock market data, searching for certain shapes, which are thought to be indicative of a stock's future performance.

Unfortunately, the sheer volume of data collected means that only a small faction of the data can ever be viewed.

Attempts to utilize classic machine learning and clustering algorithms on time series data have not met with great success. This, we feel, is due to the typically high dimensionality of time series data, combined with the difficulty of defining a similarity measure appropriate for the domain. For an example of both these difficulties, consider the three time series in Figure 1. Each of them contains 29,714 data points, yet together they account for less than .0001 % of the database from which they were extracted. In addition, consider the problem of clustering these three examples. Most people would group A and C together, with B as the outgroup, yet common distance measures on the raw data (such as correlation, absolute distance or squared error) group B and C together, with A as the outgroup.

What is needed is a representation that allows efficient computation on the data, and extracts higher order features. Several such representations have been proposed, including Fourier transformations (Faloutsos et al. 1994), relational trees (Shaw & DeFigueiredo, 1990) and envelope matching/R+ trees (Agrawal et al. 1995). The above approaches have all met with some success, but all have shortcomings, including sensitivity to noise, lack of intuitiveness, and the need to fine tune many parameters.

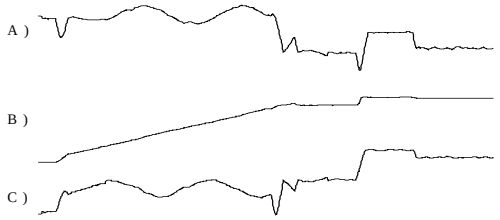

Figure 1: Examples of time series.

Piece-wise linear segmentation, which attempts to model the data as sequences of straight lines, (as in Figure 2) has innumerable advantages as a representation. Pavlidis and Horowitz (1974) point out that it provides a useful form of data compression and noise filtering. Shatkay and Zdonik (1996) describe a method for fuzzy queries on linear (and higher order polynomial) segments. Keogh and Smyth (1997) further demonstrate a framework for probabilistic pattern matching using linear segments.

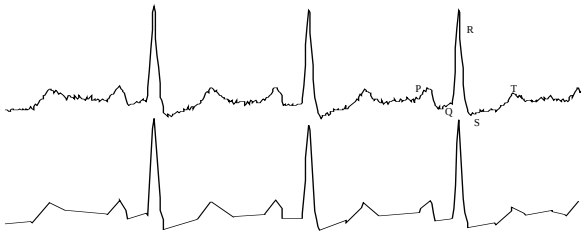

Figure 2: An example of a time series and its piece-wise linear representation.

Although pattern matching using piece-wise linear segments has met with some success, we believe it has a major shortcoming. When comparing two time series to see if they are similar, all segments are considered to have equal importance. In practice, however, one may wish to assign different levels of importance to different parts of the time series. As an example, consider the problem of pattern matching with electrocardiograms. If a cardiologist is attempting to diagnose a recent myocardial infarction (MI) she will pay close attention to the S-T wave, and downplay the importance of the rest of the electrocardiogram. If we wish an algorithm to reproduce the cardiologist's ability, we need a representation that allows us to weight different parts of a time series differently.

In this paper, we propose such a representation. We use piece-wise linear segments to represent the shape of a time series, and a weight vector that contains the relative importance of each individual linear segment. We will show that this representation allows fast and accurate clustering and classification. Additionally, we will show that this representation allows us to apply relevance feedback techniques from the information retrieval literature to time series databases.

The rest of this paper is organized as follows. Section 2 introduces notation and two important operators. Section 3 demonstrates how these techniques are used for relevance feedback, and section 4 describes a classification algorithm which takes advantage of our representation to greatly boost accuracy.

2.0 Representation of time series

There are numerous algorithms available for segmenting time series, many of which where pioneered by Pavlidis and Horowitz (1974). An open question is how to best choose *K*, the 'optimal' number of segments used to represent a particular time series. This problem involves a trade-off between accuracy and compactness, and clearly has no general solution. For this paper, we utilize the segmentation algorithm proposed in Keogh (1997). This method segments a time series, and automatically selects the best value for *K*. We emphasize, however, that all the algorithms presented in this paper will work regardless of how the segmentation has obtained.

2.1 Notation

For clarity we will refer to 'raw', unprocessed temporal data as time series, and a piece-wise representation of a time series as a sequence. We will use the following notation throughout this paper. A time series, sampled at *k* points, is represented as an uppercase letter such as A. The segmented version of A, containing *K* linear segments, is denoted as a bold uppercase letter such as **A**, where **A** is a 5-tuple of vectors of length *K*.

$$\mathbf{A} \equiv \{AXL, AXR, AYL, AYR, AW\}$$

The i^{th} segment of sequence **A** is represented by the line between $(AXL_i\ AYL_i)$ and $(AXR_i\ AYR_i)$, and AW_i, which represents the segments weight. Figure 3 illustrates this notation.

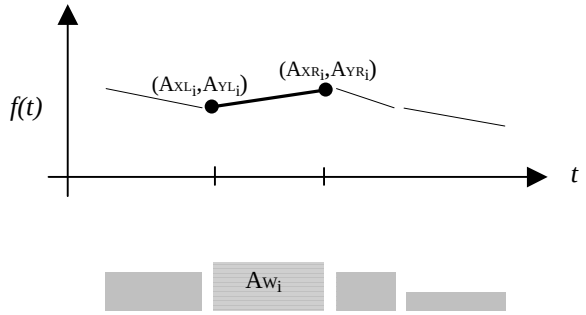

Figure 3: We represent times series by a sequence of straight segments, together with a sequence of weights (shown as the histogram) which contain the relative importance of each segment.

After a time series is segmented to obtain a sequence, we initialize all the weights to one. Thereafter, if any of the weights are changed, the weights are renormalized such that the sum of the products of each weight with the length of its corresponding segment, equals the length of the entire sequence, so that the following is always true:

$$\sum_{i=1}^{K} W_i * (AXR_i - AXL_i) = AXR_k - AXL_1$$

This renormalization is important because it gives the following property: the total weight associated with a sequence of a given length is constant, regardless of how many segments are used to represent it. Therefore, any operation that changes a weight has the effect of redistributing all the weights. For example, if the weight of a single segment is decreased, all other segments will have their weight slightly increased. So the weights reflect the relative, not absolute, importance of the segments.

2.2 Comparing time series

An advantage in using the piece-wise linear segment representation is that it allows one to define a variety of distance measures to represent the distance between two time series. This is important, because in various domains, different properties may be required of a distance measure. Figure 4 shows some of the various distortions one can encounter in time series, and we briefly consider each below.

The segmentation algorithm, which produces the sequences, acts as a noise filter, so we do not need to consider handling noise directly (Pavlidis and Horowitz 1974, Keogh 1997). Most domains require a distance measure that is insensitive to offset translation. As an example, consider two stocks, whose values fluctuate around $100 and $20 respectively. It is possible that the stock movements are very similar, but are separated by a constant amount.

Amplitude scaling, where two sequences are alike, but one has been 'stretched' or 'compressed' in the y-axis, can be dealt with easily. It simply requires normalizing the sequences before applying the distance operator. Agrawal et al. (1995) describe how to do this with raw time series. Normalizing with sequences is similar, but can be accomplished *k*/*K* times faster. Dealing with longitudinal

scaling ('stretching' or 'compressing' in the time-axis) is possible, but much more difficult. We refer the interested reader to Keogh 1997.

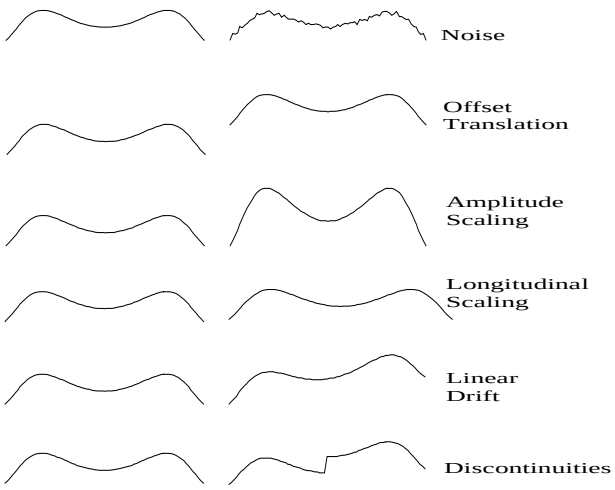

Figure 4: Some of the difficulties encountered in defining a distance measure for time series.

Linear drift occurs naturally in many domains. As an example, consider two time series, which measure the absolute sales of ice cream in two cities with similar climates and populations. We would expect the two time series to be very similar, but if one city's population remains constant while the other experiences steady growth, we will see linear drift. It is possible to remove linear trends (Box and Jenkins 1970), but it is computationally expensive. However, using the segmented representation, it is possible to define a distance measure that is insensitive to linear drift (see below).

Finally, many datasets (including the Shuttle dataset referred to in section 4.2) contain discontinuities. These are typically sensor calibration artifacts, but may have other causes. Again, one could attempt to find and remove them, but our approach is to simply define a distance measure that is (relatively) insensitive to them

It is possible to define distance measures for sequences that are invariant to any subset of the above distortions. For the experiments in this paper we used the simple distance measure given below. This measure is designed to be insensitive to offset translation, linear trends and discontinuities.

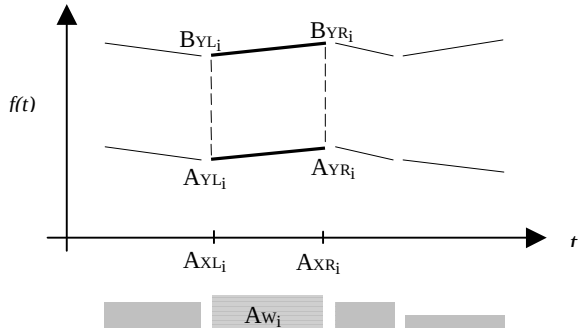

Figure 5: Comparing two time series is equivalent to summing the absolute difference between the pair of dash lines, (weighted by the appropriate weighting factor) for every segment.

It is convenient for notational purposes to assume that the endpoints of the two sequences being compared are aligned, as in Figure 5. In general, with real data, this is not the case. So we will have to process the sequences first, by breaking some segments at a point which corresponds to an endpoint in the other sequence. This can be done in $O(K)$.

$$D(A,B) = \sum_{i=1}^{K} AW_i * BW_i * |(AYL_i - BYL_i) - (AYR_i - BYR_i)|$$

Intuitively, this metric measures how close corresponding segments from **A** and **B** are to being parallel. Note that it has the following desirable properties:

$$D(\mathbf{A},\mathbf{B}) = D(\mathbf{B},\mathbf{A})$$
$$D(\mathbf{A},\mathbf{B}) \geq 0$$
$$D(\mathbf{A},\mathbf{A}) = 0$$
$$D(\mathbf{A},\mathbf{B}) = 0 \quad \rightarrow \quad \mathbf{A} = \mathbf{B}$$

Another important property is that it is efficient to compute. In particular, comparing two sequences **A** and **B** is approximately k/K faster than comparing the two corresponding time series A and B. This is especially important as one may need to calculate distance frequently. For example, suppose one wishes to hierarchically cluster n data items using a group average method. This requires $O(n^3)$ comparisons. The time series represented by the sequences shown in Figure 6 contain 29,714 data points, but their sequence representations contain only an average of 35 segments. This results in a speedup of 29,714 / 35 ≈ 848.

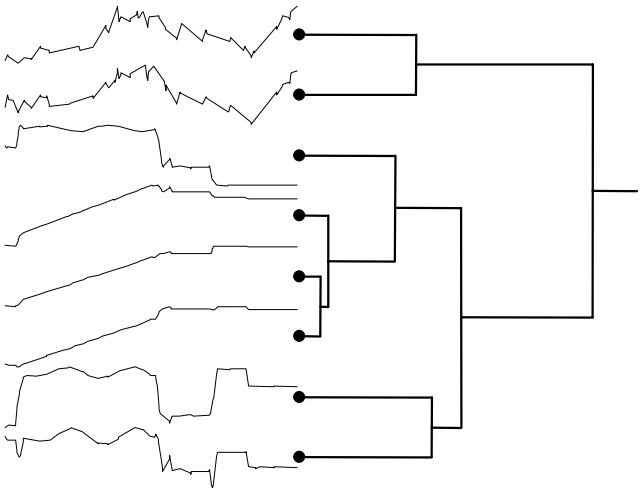

Figure 6: An example of hierarchically clustered time series.

2.3 Merging time series

In this section we define an operation on sequences which we call 'merge'. The merge operator allows us to combine information from two sequences, and repeated

application of the merge operator allows us to combine information from multiple sequences. The basic idea is that the merge operator takes two sequences as input, and returns a single sequence whose shape is a compromise between the two original sequences, and whose weight vector reflects how much corresponding segments in each sequence agree. As in the distance operator we assume that the endpoints of the sequence are aligned.

When merging sequences one may wish for one of the two input sequences to contribute more to the final sequence than the other does. To accommodate this, we associate a term called 'influence' with each of the input sequences. The influence term associated with a sequence **S** is a scalar, denoted as **S**I, and may be informally considered a 'mixing weight'. Where the influence term comes from depends on the application and is discussed in detail in sections 3.1 and 4.1 below.

To merge the two sequences **A** and **B** with influence terms **A**I and **B**I respectively, we use the following algorithm that creates the sequence **C**:

```
if (AI * BI < 0) then      sign = -1
else                       sign =  1
end

mag  = min(|AI|,|BI|) / max(|AI|,|BI|)

scale = max(max(AYL),(AYR)) -
min(min(AYL),(AYR))

for i = 1 to K
     CXL_i = AXL_i
     CXR_i = AXR_i
     CYL_i = ((AYL_i * AI)+(BYL_i * BI))/(AI+BI)
     CYR_i = ((AYR_i * AI)+(BYR_i * BI))/(AI+BI)
     run   = AXR_i - AXl_i
     rise  = |(AYL_i - BYL_i)-(AYR_i - BYR_i)|
     diff  = (rise / run) * scale
     CW_i = (AW_i*BW_i)*(1+(sign * mag)/(1 +
diff))
end
CW = normalize(CW)
```

Table 1: The merge algorithm.

Figure 7 shows two sequences that have been merged with various values for the influence terms. Note that the resultant sequence is more like the sequence with the higher influence. Note also that the weights are more differentiated the closer the two influence terms are to each other. This is because neither sequence is dominating, and the maximum amount of compromise is taking place. As you would expect, the trivial operations **C** = merge([**A**,**A**I],[**B**,0]) and **C** = merge([**A**,**A**I],[**A**,**A**I]) both result in **C** = **A**.

2.4 Learning prototypes

Although the merge operator is designed to be a component in the more sophisticated algorithms presented below, it can, by itself, be considered a simple learning algorithm that creates a prototypical sequence. Creating a prototype solely from positive examples works in the following manner. We have a model sequence **A**, which is a typical example of a class of sequences. If we merge sequence **A** with sequence **B**, another member of the same class, the resultant sequence **C** can be considered a more general model for the class. In particular the differences in shape are *minimized* by averaging, and the weights for similar segments are *increased*.

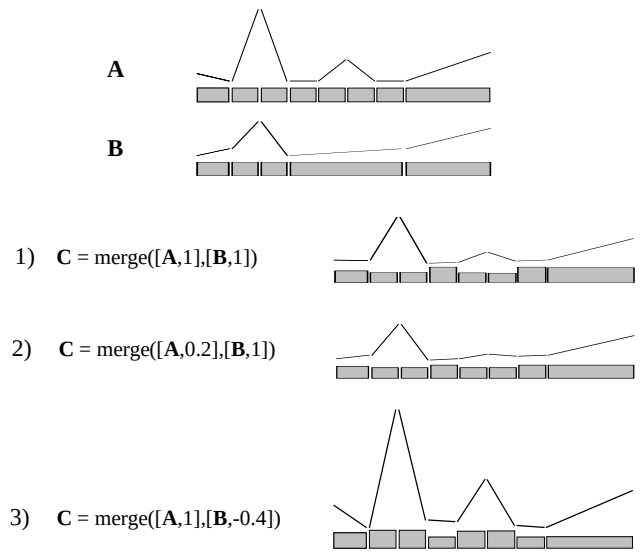

Figure 7: Examples of the merge operator with various influence terms.

1) With two equal influence terms, the shape of the resultant sequence **C** is "halfway" between **A** and **B**.
2) With **B**'s influence term much larger than **A**'s, the shape of the resulting sequence **C** is much closer to **B** than **A**.
3) With a negative influence term for **B** the shape of the resulting sequence **C** looks like **A** where the differences between **A** and **B** have been exaggerated.

In contrast, creating a prototype from both positive and negative examples uses a negative influence for the negative examples. As before, suppose we have a sequence **A**, which is an example of one class of sequences. However, suppose **B** is an example of a different class. If we merge **A** with **B**, using a negative influence term for **B**, the resultant sequence **C** is a new prototype for **A**'s class where the differences in shape between **A** and **B** are *exaggerated* and the weights for similar segments are *decreased*.

The above maps neatly to our intuitions. If we are learning a prototype for a class of sequences from a set of positive examples, we want the shape learned to be an average of all the examples, and we want to increase the weight of segments that are similar, because those segments are good predictors of the class. If however, we are trying to learn from a negative example, we want to exaggerate the differences in shape between classes, and decrease the weight of similar segments, because segments that are similar across classes have no discriminatory power.

Figure 7.3 shows an illustration of learning from a negative example. The negative example is given a

negative influence term. As before the magnitude of the influence term reflects how much the corresponding sequence affects the resultant sequence.

3.0 Relevance feedback

Relevance feedback is the reformulation of a search query in response to feedback provided by the user for the results of previous versions of the query. It has an extensive history in the text domain, dating back to Rocchio's classic paper (1971). Recently, there have been attempts to utilize relevance feedback techniques in other domains, notably the MARS project (Rui et al. 1997). However, to the best of the authors' knowledge, no one has attempted explore time series databases in a relevance feedback framework, in spite of the fact that relevance feedback has been shown to significantly improve the querying process in text databases (Salton & Buckley, 90). In this section we present a simple relevance feedback algorithm which utilizes our representation and we demonstrate it on a synthetic dataset.

Our relevance feedback algorithm works in the following manner. An initial query sequence **Q** is used to rank all sequences in the database (this query may be hand drawn by the user). Only the best *n* sequences are shown to the user. The user assigns influences to each of *n* sequences. A positive influence is given to sequences that the user approves of. Negative influences are given to sequences that the user finds irrelevant.

The relative magnitude of influence terms reflects how strongly the user feels about the sequences. So if a user "likes" $\mathbf{S}_i$ twice as much as $\mathbf{S}_j$ he can assign influences of 1,½ or 2,1 etc. The sequences are then merged to produce a new query, and the process can be repeated as often as desired.

$$\mathbf{Q}_{new} = \text{merge}([\mathbf{Q}_{old}, \mathbf{Q}_{old}\mathbf{I}], [\mathbf{S}_1, \mathbf{S}_1\mathbf{I}], [\mathbf{S}_2, \mathbf{S}_2\mathbf{I}], \ldots, [\mathbf{S}_n, \mathbf{S}_n\mathbf{I}])$$

$$\mathbf{Q}_{new}\mathbf{I} = \mathbf{Q}_{old}\mathbf{I} + \mathbf{S}_1\mathbf{I} + \mathbf{S}_2\mathbf{I} + \ldots + \mathbf{S}_n\mathbf{I}$$

3.1 Experimental results

To test the above algorithm, we conducted the following experiment. We constructed 500 "Type A", and 500 "Type B" time series, which are defined as follows:

- Type A: $\text{Sin}(x^3)$ normalized to be between zero and one, plus Gaussian noise with $\sigma = .1$ $\quad -2 \le x \le 2$
- Type B: $\text{Tan}(\text{Sin}(x^3))$ normalized to be between zero and one, plus Gaussian noise with $\sigma = .1$ $\quad -2 \le x \le 2$

The time series, which were originally sampled at 800 points, were segmented. Figure 8 shows an example of each type. Note that they are superficially very similar, although Type B has a somewhat sharper peak and valley. We built an initial query by averaging all 1,000 time series and segmenting the result.

Twenty-five experimental runs where made. Each run consisted of the following steps. A coin toss decided whether Type A or Type B was to be the "target" shape

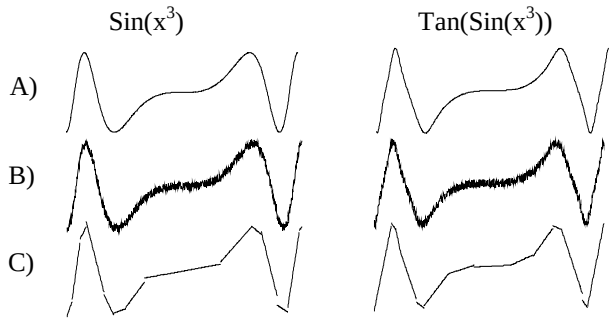

Figure 8: Synthetic data created for relevance feedback experiment.
A) The original time series.
B) The original time series with noise added.
C) The segmented version of the time series.

(that is, the shape to be considered "relevant" for that particular experiential run). The initial query was made, and the quality of the ranked sequences was measured as defined below. The best 15 sequences were shown to the user, who then rated them by assigning influences that reflected how closely he thought they resembled the target shape. A new query was built and the search/rate process was repeated twice more.

We evaluated the effectiveness of the approach by measuring the average precision of the top 15 sequences, and the precision at the 25, 50 and 75 percent recall points. Precision (*P*) is defined as the proportion of the returned sequences which are deemed relevant, and recall (*R*) is defined as the proportion of relevant items which are retrieved from the database. These results are shown in Table 2.

In order to see if the ability to assign negative influence terms is helpful, we did the following. For each experimental run we also built queries which contained just the feedback from the sequences judged relevant. These results are shown in parentheses in Table 2.

	Initial Query	Second Query	Third Query
P of top 15	.51	.91 (.68)	.97 (.72)
P at 25% *R*	.52	.91 (.69)	.96 (.71)
P at 50% *R*	.49	.89 (.66)	.95 (.69)
P at 75% *R*	.51	.87 (.63)	.95 (.68)

Table 2: Results of relevance feedback experiments. The values recorded in parentheses are for the queries built just from positive feedback.

As one might expect, the initial query (which does not have any user input) returns the sequences in essentially random order. The second query produces remarkable improvement, and the third query produces near perfect ranking. The queries built from just positive feedback do produce improvement, but are clearly inferior to the more general method. This demonstrates the utility of learning from both positive and negative instances.

4.0 Classification

Although others have done work in defining distance measures for time series, this work has generally not been presented in a machine learning framework, with the well-defined task of building an algorithm which can, after examining a training set of labeled examples, accurately classify future unlabeled instances. In this section we describe a novel classification algorithm which takes advantage of our representation, and we evaluate it using classic machine learning methodology, including cross validation and comparison with alternative approaches.

One difficulty in casting time series classification problems in a machine learning context is that in machine learning problems we are typically presented with two or more mutually exclusive, well defined classes, such as "sick"/"healthy". In time series problems, however, it is more common to have a single well-defined class, and non-class instances that do not exhibit any particular structure. As a good example, consider Figure 9. So instead of attempting to decide if an instance is closer to class X or class Y, the algorithm must decide if an instance is *sufficiently* close to class X to be classified as such. Naturally, deciding how close is sufficiently close is something we wish the algorithm to induce from the training set.

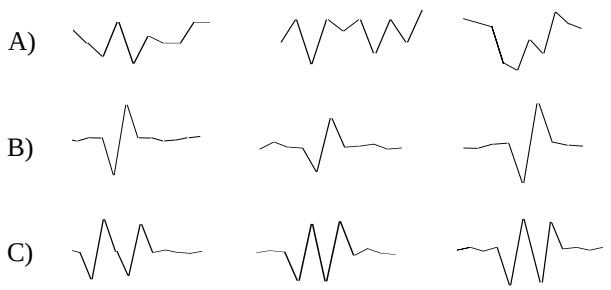

Figure 9: Instances from the Heart dataset. Note that the negative instances, shown in row A, do not exhibit any particular structure. However, the positive instances, shown in rows B and C, seem to fall into two types, single peaked, as in B, and double peaked as in C.

Given our representation, an obvious approach to classification is to merge all positive instances in the training set, and use the resultant sequence as a template to which the instances to be classified are compared. This may work in some circumstances, but in some domains it may totally fail. The reason for the potential failure is that there may be two or more distinct shapes that are typical of the class. Figure 9 demonstrates that this problem exists in the heart dataset.

A similar problem can occur in other time series domains, there may be several distinct shapes that are prototypical of a single class, and merging them into a single prototype will result in a shape which does not particularly resemble any individual member of the class. To avoid this problem, an algorithm needs to be able to detect the fact that there are multiple prototypes for a given class, and classify an unlabeled instance to that class if it is sufficiently similar to *any* prototype. In the next section we describe such an algorithm, which we call CTC (Cluster, Then Classify).

4.1 Classification algorithm

Table 3 shows an outline of our learning algorithm. The input is *S*, the set of *n* sequences that constitute the training data. The output is *P*, a set of sequences, and a positive scalar ε.

An unseen sequence **U** will be classified as a member of a class if and only if, the distance between **U** and at least one member of *P* is less than ε.

The algorithm clusters the positive examples using the group average agglomerative method as follows. The algorithm begins by finding the distance between each negative instance in S, and its closest positive example. The mean of all these distances, $neg\text{-}dis_1$ is calculated. Next the distance between each positive instance in *S*, and the most similar positive example is calculated. The mean of all these distances, $pos\text{-}dis_1$ is calculated. The fraction $q_1 = pos\text{-}dis_1 / neg\text{-}dis_1$ can now be calculated.

At this point, the two closest positive examples are replaced with the result of merging the two sequences, and the process above is repeated to find the fraction $q_2 = pos\text{-}dis_2 / neg\text{-}dis_2$. The entire process is repeated until a single sequence remains. Figure 10 shows a trace through this part of the algorithm for a dataset that contains just 4 positive instances. The set of sequences returned is the set for which q_i is minimized. The ε returned is $(pos\text{-}dis_i + neg\text{-}dis_i)/2$.

The algorithm as described is optimized for simplicity and requires $O(n^3)$ comparisons. By storing the distances calculated in the first iteration and only recalculating distances where necessary, it is possible to achieve a speedup of several constant factors.

```
i = 1
Let P_i be the set of all positive instances in S
Cluster all sequences in P_i
for i = 1 to n
 neg-dis_i = mean distance between all negative
   instances and their closest match in P_i
 pos-dis_i = mean distance between all positive
   instances and their closest match in P_i
 q_i = pos-dis_i / neg-dis_i
 Let A and B be the closest pair of sequences in P_i
 C  = merge([A,AI],[B,BI])
 CI = AI + BI
 Remove A and B from P_i
 Add C to P_i
end
 let best equal the i for which q_i is minimized
 return P_best , (pos-dis_best + neg-dis_best) / 2
```

Table 3: The CTC learning algorithm.

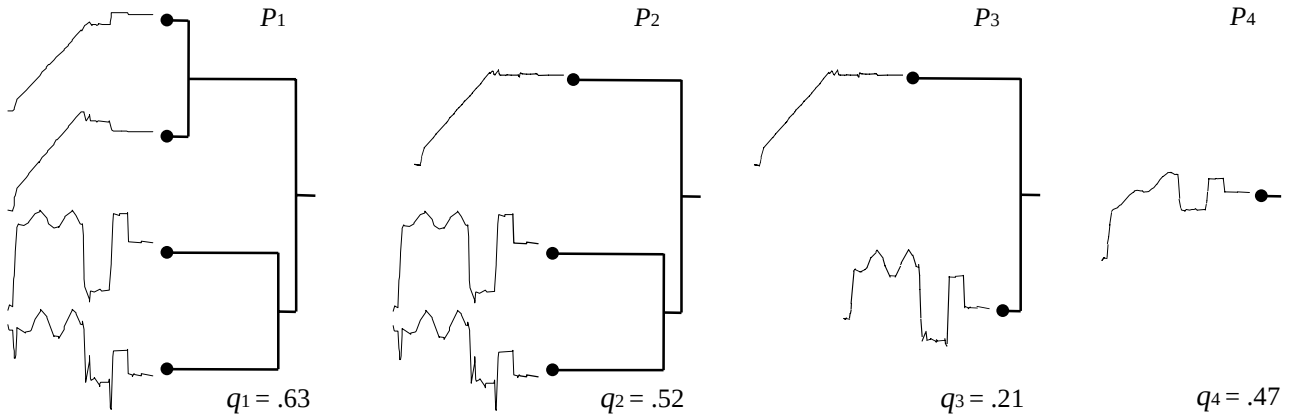

Figure 10: A trace through the CTC algorithm on a small dataset. The first set of prototypes P_1, are too specific and do not generalize to the test set. The final set P_4 contains a single sequence that is too general, because it is trying to model two distinct shapes. The set P_3 is the best compromise.

4.2 Experimental results

To test the algorithm presented above we ran experiments on the following datasets.

- **Shuttle**: This dataset consists of the output of 109 sensors from the first eight-hours of Space Shuttle mission STS-067. The sensors measure a variety of phenomena. 18 of them are Inertia Movement Sensors, measured in degrees (examples are shown in Figures 1 and 10). The task is to distinguish these from the other 91 sensors. The sensors are sampled at different frequencies. This is not a problem for our algorithm, but is a great difficulty for methods which work with the raw data. So we created "clean" 1,000 point versions of each sensor and used those for our experiments.
- **Heart**: This dataset consists of RR intervals obtained form Holter ECG tapes sampled at 128 Hz (Zebrowski 1997). The data is in a long sequence that contains 96 ventricular events. We extracted the 96 events, and 96 other sections, of equal length, chosen at random. Figure 9 shows some examples of both the ventricular events class and the non-class data.

For comparison purposes we evaluated 4 algorithms on the two datasets described above. **CTC** is the algorithm described in section 4.1. **CTC**$_{UW}$ is the same algorithm with the weight feature disabled (we simply hardcoded the weights to equal one). **NN** is a simple nearest neighbor algorithm that uses the raw data representation of the time series. An unlabeled instance is assigned to the same class as its closest match in the training set. We used absolute error as a distance measure, having empirically determined that it was superior to the other obvious candidates (i.e. squared error, correlation) on these two datasets. **NN**$_S$ is the same algorithm as **NN** except it uses the sequence representation and the distance measure defined in section 2.2.

We ran 2 fold cross validation 100 times. All algorithms were trained and tested on exactly the same folds. The results are presented in table 4.

	CTC	**CTC**$_{UW}$	**NN**	**NN**$_S$	Default
Shuttle	98.1	94.4	83.9	87.7	83.5
Heart	84.5	71.4	61.2	66.3	50.0

Table 4: Experiment results of classification experiments

On both datasets the CTC algorithm performs the best. Its ability to outperform CTC$_{UW}$ seems to be a justification of our weighted representation. On the Shuttle dataset NN performs at the base rate. We surmise this is probably due to its sensitivity to discontinuities, which are a trait of this dataset. NN$_S$ ability to do significantly better supports this hypothesis.

5.0 Related work

There has been no work on relevance feedback for time series. However, in the text domain there is an active and prolific research community. Salton and Buckley (1990) provide an excellent overview and comparison of the various approaches.

Agrawal et al. (1995) demonstrates a distance measure for time series that works by dividing up the sequences into windows. Corresponding windows from two time series are compared, and the two time series are said to be similar if enough of these windows are similar. The individual windows are said to be similar if one lies within an envelope of a specified width around another. Each window has its data normalized to remove the effects of amplitude scaling and offset translation.

In general, pattern matching on 'raw data' is not feasible because of the sheer volume of data. In addition, raw data may contain spikes, dropouts or other noise that could confuse the matching process. A variety of higher-level representations of times series have been proposed, most notably the Discrete Fourier Transform. This approach

involves performing a Discrete Fourier Transform on the original time series, discarding all but the *K* most informative coefficients, and then mapping these coefficients into *K*-dimensional space. The original work by Agrawal, Faloutsos and Swami (1993) only allowed the comparison of two time series of equal length, but was extended by Faloutsos, Ranganathan and Manolopoulos (1994) to include subsequence matching.

Others, including, Shatkay and Zdonik (1996) recognize that a piece-wise linear (and higher order) order representation, greatly reduces the required storage and search space for a time series, but fail to suggest a robust distance measure.

6.0 Conclusions

We introduced a new enhanced representation of time series and empirically demonstrated its utility for clustering, classification and relevance feedback. Future directions for our research include a more extensive evaluation of our algorithms, and incorporating query expansion (Salton and Buckley 1990) into our relevance feedback algorithm.

References

Agrawal, R., Faloutsos, C., & Swami, A.(1993). Efficient similarity search in sequence databases. *Proc. of the 4th Conference on Foundations of Data Organization and Algorithms*, Chicago, October.

Agrawal, R., Lin, K. I., Sawhney, H. S., & Shim, K.(1995). Fast similarity search in the presence of noise, scaling, and translation in times-series databases. In *VLDB*, September.

Box, G. P., & Jenkins, G.M (1970). Time series analysis, forecasting and control. San Francisco, Ca. Holden-Day

Cheng, Y. C., & Lu, S. Y. (1982). Waveform correlation using tree matching. *IEEE Conf. PRIP*.

Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. SIGMOD - *Proceedings of Annual Conference*, Minneapolis, May.

Hagit, S., & Zdonik, S. (1996). Approximate queries and representations for large data sequences. *Proc. 12th IEEE International Conference on Data Engineering*. pp 546-553, New Orleans, Louisiana, February.

Keogh, E. (1997). Fast similarity search in the presence of longitudinal scaling in time series databases. *Proceedings of the 9th International Conference on Tools with Artificial Intelligence*. pp 578-584. IEEE Press.

Keogh, E., Smyth, P. (1997). A probabilistic approach to fast pattern matching in time series databases. *Proceedings of the 3rd International Conference of Knowledge Discovery and Data Mining*. pp 24-20, AAAI Press.

Pavlidis, T., Horowitz, S., (1974). Segmentation of plane curves. *IEEE Transactions on Computers*, Vol. C-23, No 8, August.

Salton, G., & Buckley, C. (1990). Improving retrieval performance by relevance feedback. JASIS 41. pp. 288-297.

Shaw, S. W. & DeFigueiredo, R. J. P. (1990). Structural processing of waveforms as trees. *IEEE Transactions on Acoustics, Speech, and Signal Processing.* Vol. 38 No 2 February.

Rocchio, J. J., Jr.(1971).Relevance feedback in information retrieval: The Smart System - *Experiments in Automatic Document Processing*, ed. Salton, G., Prentice-Hall Inc., pp. 337-354.

Rui,Y., Huang,T.S., Mehrotra, S. & Ortega, M. (1997). Automatic matching tool selection using relevance feedback in MARS. *Proceedings of 2nd Int. Conf. On Visual Information Systems.*

Zebrowski, J,J. (1997). http://www.mpipks-dresden.mpg.de/~ntptsa/Data/Zebrowski-D/