Fast Time Series Classification Using Numerosity Reduction

Xiaopeng Xi XXI@CS.UCR.EDU
Eamonn Keogh EAMONN@CS.UCR.EDU
Christian Shelton CSHELTON@CS.UCR.EDU
Li Wei WLI@CS.UCR.EDU
Computer Science & Engineering Department, University of California, Riverside, CA 92521 USA

Chotirat Ann Ratanamahatana ANN@CP.ENG.CHULA.AC.TH
Department of Computer Engineering, Chulalongkorn University, Bangkok 10330, Thailand

Abstract

Many algorithms have been proposed for the problem of time series classification. However, it is clear that one-nearest-neighbor with Dynamic Time Warping (DTW) distance is exceptionally difficult to beat. This approach has one weakness, however; it is computationally too demanding for many real-time applications. One way to mitigate this problem is to speed up the DTW calculations. Nonetheless, there is a limit to how much this can help. In this work, we propose an additional technique, numerosity reduction, to speed up one-nearest-neighbor DTW. While the idea of numerosity reduction for nearest-neighbor classifiers has a long history, we show here that we can leverage off an original observation about the relationship between dataset size and DTW constraints to produce an extremely compact dataset with little or no loss in accuracy. We test our ideas with a comprehensive set of experiments, and show that it can efficiently produce extremely fast accurate classifiers.

1. Introduction

The problem of time series classification has attracted great interest recently, finding applications in domains as diverse as medicine, finance, entertainment, and industry. Many algorithms have been proposed for time series classification, including decision trees (Rodriguez & Alonso, 2004), neural networks (Nanopoulos & Manolopoulos, 2001), Bayesian classifiers, SVM (Wu & Chang, 2004), etc. However, as we shall show, the simple combination of one-nearest-neighbor with Dynamic Time Warping (DTW) distance has proven exceptionally difficult to beat. This approach has one weakness, however; it is computationally too demanding for many real-time applications. One way to mitigate this problem is to speed up the DTW calculations, and great progress has been made in this area in the past few years (Keogh, 2002). Recent results, however, suggest that we are at the asymptotic limit of speeding up DTW (Ratanamahatana & Keogh, 2005).

Numerosity reduction (Pekalska & Duin et al, 2006; Wilson & Martinez, 1997) offers an additional possibility to speed up one-nearest-neighbor DTW (1NN-DTW). We can simply discard a large fraction of the training data to improve the performance. It is well known that for general numerosity reduction algorithms, if we are careful in choosing which objects we discard, we can significantly reduce the classification time while maintaining high accuracy, in some cases actually improving the accuracy.

While the idea of numerosity reduction for nearest-neighbor classifiers has a long history, we show here that we can leverage off an original observation about the relationship between dataset size and DTW constraints to produce extremely compact datasets with little or no loss in accuracy. The observation is that the optimal amount of warping we should allow the DTW algorithm to attempt, depends on number of objects in the training set. This is only an empirical observation; however, we have confirmed it on thirteen diverse datasets.

The essence of our approach is therefore to limit the amount of warping freedom given to the DTW algorithm when we have a large dataset, but gradually increase this amount of warping freedom as we begin to discard objects. We do this with a simple greedy search algorithm. Because our idea involves multiple computations of DTW for different amounts of warping freedom, a naïve brute force implementation would be intractable for large datasets. However, we show that a combination of caching and pruning based on admissible lower bounds allows us to make our approach tractable for very large datasets.

2. Background and Related Work

Appearing in *Proceedings of the 23nd International Conference on Machine Learning*, Pittsburgh, PA, 2006. Copyright 2006 by the author(s)/owner(s).

2.1 Dynamic Time Warping

DTW may be considered simply as a tool to measure the dissimilarity between two time series, after aligning them.

Suppose we have two time series Q and C, of length p and m, respectively, where:

$$Q = q_1,q_2,\ldots,q_i,\ldots,q_p \qquad (1)$$
$$C = c_1,c_2,\ldots,c_j,\ldots,c_m \qquad (2)$$

To align two sequences using DTW, we construct a p-by-m matrix where the (i^{th}, j^{th}) element of the matrix contains the distance $d(q_i, c_j)$ between the two points q_i and c_j (i.e., $d(q_i, c_j) = (q_i - c_j)^2$). Figure 1 illustrates this notation. Each matrix element (i, j) corresponds to the alignment between the points q_i and c_j. A warping path W, is a contiguous set of matrix elements that defines a mapping between Q and C. The k^{th} element of W is defined as $w_k = (i, j)_k$; so we have:

$$W = w_1, w_2, \ldots,w_k,\ldots,w_K \qquad \max(m, p) \le K \le m+p\text{-}1 \quad (3)$$

The warping path must satisfy several constraints. 1) **Boundary conditions:** $w_1 = (1, 1)$ and $w_K = (p, m)$, this requires the warping path to start and finish in diagonally opposite corners. 2) **Continuity:** Given $w_k = (a, b)$ then $w_{k-1} = (a', b')$ where $a – a' \le 1$ and $b - b' \le 1$. This restricts the allowable steps in the warping path to adjacent cells. 3) **Monotonicity:** Given $w_k = (a, b)$ then $w_{k-1} = (a', b')$ where $a – a' \ge 0$ and $b - b' \ge 0$. This forces the points in W to be monotonically spaced in time.

Of the exponentially many warping paths that satisfy the above conditions, we are only interested in the path that minimizes the warping cost:

$$DTW(Q,C) = \min\left\{\sqrt{\sum_{k=1}^{K} w_k}\right. \qquad (4)$$

This path can be found using dynamic programming to evaluate the following recurrence which defines the cumulative distance $\gamma(i, j)$ as the distance $d(i, j)$ found in the current cell and the minimum of the cumulative distances of the adjacent elements:

$$\gamma(i, j) = d(q_i, c_j) + \min\{ \gamma(i\text{-}1, j\text{-}1),\gamma(i\text{-}1, j),\gamma(i, j\text{-}1) \} \quad (5)$$

An obvious observation is that an intuitive alignment path is unlikely to drift very far from the diagonal. This observation has been exploited by limiting the warping path to a warping window of size r, directly above and to the right of the diagonal. This warping window is illustrated in the right of Figure 1 as two straight lines parallel to the diagonal. We can specify r as a percentage of the length of the longer of the two time series. Note that the Euclidean distance between two sequences can be seen as a special case where r = 0%.

Clearly, the size of the warping window greatly affects the speed of the DTW computation. If r is small, a large fraction of the matrix does not need to be examined (or even constructed), and the search for the optimal warping path is correspondingly faster. In addition to this speedup by reducing the area of matrix examined, warping windows can be exploited to create tight lower bounds to the DTW distance. This idea, called LB_Keogh, was introduced in Keogh (2002) and has been used in dozens of applications where fast DTW is required, including motion capture indexing (Fu & Keogh et al, 2005), query by humming (Zhu & Shaha, 2003), and P2P searching (Karydis et al, 2005).

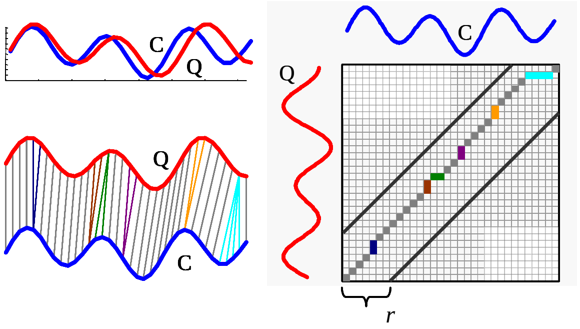

Figure 1. Left) Two time series sequences which are similar but out of phase. Right) To align the sequences, we construct a warping matrix, and search for the optimal warping path (solid squares). Note a band with width r is used to constrain the warping.

Because the value of DTW(Q,C) can only decrease with a larger value of r, it was widely assumed in the literature that wider warping windows are “better” and that warping windows are only useful for speeding up the computations. A recent paper (Ratanamahatana & Keogh, 2005) showed that this is a myth. Relatively tight warping windows actually *improve* accuracy of classification, clustering, and query by content. On thirteen diverse datasets, we exhaustively searched for the value of r that maximizes the classification accuracy. Table 1 summarizes the results (see also, Figure 3 which visually explains how this was done for Gun-Point).

Table 1. The value of r that produces the most accurate one-nearest-neighbor classifier on thirteen diverse datasets.

DATASET	SIZE	BEST *r* (%)	ERROR (%)
Cylinder-Bell-Funnel	300	1	0.00
ControlChart	600	8	0.33
ECG	200	0	10.0
Face (four)	112	3	3.57
Gun-Point	200	3	1.00
Leaf	442	8	3.85
Lighting(FORTE-2)	121	5	9.09
Pulse	100	0	1.00
Trace	200	3	0.00
Two Patterns	5000	3	0.00
Wafer	7164	1	0.07
Word Spotting	905	3	20.0
HapticX	80	15	35.0

There are still a handful of papers that do not seem to accept this. For example, Shou & Manoulis et al. (2005) say “*there may still be cases where unconstrained warping is useful*.” Tellingly, they do not show such a case. As we can see, relatively tight warping constraints produce more accurate classifiers.

2.2 Time Series Classification

A central claim of this work is that 1NN-DTW is an exceptionally competitive classifier, in spite of a massive research effort on time series classification problems. We arrived at this conclusion after an extensive literature search, which we highlight below.

In Rodriguez & Alonso (2004), the authors use a DTW based decision tree to classify time series. On the Two Patterns dataset, they report an error rate of 4.9%, but our experiments on the same dataset using 1NN give an error rate of 1.04% for Euclidean distance and 0.0% for DTW. Similar results apply for all the datasets they test on.

In Rodriguez & Alonso et al. (2000), the authors use first order logic rules with boosting to classify time series. On the ControlChart problem, they report an error rate of 3.6%, but our experiments on the same dataset using 1NN-DTW give an error rate of 0.33%.

In Nanopoulos & Alcock et al. (2001), the authors use a multi-layer perceptron neural network on the ControlChart problem to achieve their best performance of 1.9% error rate. Using 1NN-DTW on the same dataset gives 0.33% error rate.

In Wu & Chang (2004), the authors use a "super-kernel fusion scheme" to achieve an error rate of 0.79% on ControlChart dataset. Using 1NN-DTW on the same dataset gives an error rate of 0.33%.

In Chen & Kamel (2005), the authors use "Static Minimization-Maximization approach" to build Multiple Classifier Systems. They test several flavors of their approach on the ControlChart problem to achieve their best performance of 7.2% error. Using 1NN-DTW on the same dataset gives an error rate of 0.33%.

In Kim & Smyth et al. (2004), the authors use hidden Markov Models to achieve 98% accuracy on the PCV-ECG classification problem, but both DTW and Euclidean distance achieves a perfect accuracy on the same problem.

In Hayashi & Mizuhara et al. (2005), the authors use DTW distances to embed time series into a lower dimensional space using a Laplacian eigenmap. This embedding is designed to both improve accuracy and performance. They show that they can achieve 100% accuracy on a subset of the ControlChart, and show that DTW only gets 99% accuracy. However, when we reimplemented the experiment, we found that DTW actually gets 100% on this problem.

In Chen & Ozsu et al. (2005), the authors use a measure based on multi-scale histograms. They try every possible permutation of two parameters on the ControlChart problem to achieve a best performance of 6.0% error. However, using 1NN-DTW on the same dataset gives an error rate of 0.33%.

In Eads & Glocer et al. (2005), the authors apply grammar-guided feature extraction for time series classification, and they get the best result for FORTE-2 of 13.22% error rate. While using 1NN-DTW distance, we can get the performance of 9.09%.

The above list is truncated for brevity. There are dozens of similar examples in the literature. In addition to the above, there are a handful of papers in the literature that do *explicitly* claim to have a distance measure that beats DTW. For example, in Megalooikonomou & Wang et al. (2005), the authors introduce a symbolic multiresolution approach that appears to slightly outperform DTW. However, the approach is only tested on very small datasets, including one of size 24 and one of size 15, and they trained and tested on the same data (Megalooikonomou, 2006). Likewise, Lei & Govindaraju (2004) claim that DTW gets 96.5% accuracy on the Gun-Point problem whereas their approach gets 98.0%. However, DTW *actually* gets 99.0% on that problem.

Note that the works in the list above *do* make contributions in telling us something about decision trees, boosting, or other classification methods. In addition, the authors are to be commended for experimenting on datasets that are in the public domain. Our point is simply that if you want accurate classification of time series, 1NN-DTW is *very* hard to beat.

Finally, we make the claim that DTW is at least as accurate as Euclidean distance for classification problems, and generally is significantly better. To show this, we test on thirteen datasets using leaving-one-out nearest neighbor evaluation. Euclidean distance has no parameters, and DTW requires one parameter, which we learn by looking only at training data. Figure 2 sees a visual summary of the results.

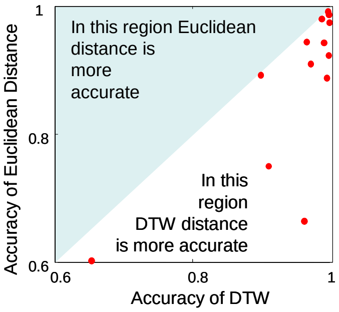

Figure 2. Thirteen time series classification problems, plotted as points with the DTW accuracy as the *x*-axis and the Euclidean distance accuracy as the *y*-axis.

Given all the above, 1NN-DTW seems to be the best approach for time series classification. However, if naively implemented, it is computationally demanding. The good news is that recent results show that the amortized cost of DTW is essentially O(*n*) using LB_Keogh lower bounding technique (Ratanamahatana & Keogh, 2005). This fact strongly suggests that we are at the limit of speeding up DTW; if we wish to make classification even faster, we must look elsewhere. In the next section, we consider a possibility of gaining speedup by using numerosity reduction.

3. Naïve Rank Numerosity Reduction

The nearest neighbor classification algorithm has linear complexity both in storage and query time. It has long been noted that "*data reduction techniques can be applied to obtain a reduced representative of the data set which is much smaller in volume, yet closely maintains the integrity of the original data*" (Han & Kamber, 2000). Existing data reduction techniques include random reduction and data condensing/editing (Dasarathy, 1991; Pekalska et al., 2006).

Here, we consider the work of Wilson & Martinez (1997) in more detail, since it is perhaps the most referenced work in this area. Wilson and Martinez proposed three numerosity reduction algorithms, RT1, RT2, and RT3. Their intuition is that if the removal of an instance *p* will not cause any other instances be misclassified, *p* can then be discarded. They also introduced two novel definitions, nearest *enemy* and *associates*: each instance *p* has a nearest *enemy* which is the nearest instance in a different class, and those instances that have *p* as one of their *k* nearest neighbors are called *associates* of *p*. Their various algorithms decide whether an instance *p* should be discarded based on its nearest *associates* and *enemy*. These algorithms have been shown to be able to maintain high classification accuracy while providing substantial storage reduction. Note that the RT algorithms require storing the *k* nearest neighbors, one nearest *enemy*, and all *associates* for each instance. In this section, we present a competitive rank-based reduction method that only uses one nearest neighbor. Because of its simplicity, we call it Naïve Rank Reduction.

3.1 Naïve Rank Reduction

The Naïve Rank Reduction algorithm works in two steps, *ranking* and *thresholding*. We first assign ranks to all instances based on their contribution in classification on the training set. Then we use a user defined threshold *n* to decide how many instances to keep and simply keep these *n* highest ranked instances for future classification.

In the *ranking* step, we begin by removing duplicated instances, if any. Then we apply one-nearest-neighbor classification on the training set. We give lowest ranks to misclassified instances. This is motivated by the observation in Wilson & Martinez (1997) that these instances tend to be noisy and may drastically change the ranking of other instances. For correctly classified instances, we assign their ranks by the following formula

$$rank(x) = \sum_j \begin{cases} 1 & \text{if } \operatorname{class}(x) = \operatorname{class}(x_j) \\ -2 & \text{otherwise} \end{cases} \quad (5)$$

where x_j is the instance having x as its nearest neighbor. The intuition is that we attempt to keep *good* instances by giving positive value to the instances that help correctly classify other data, and giving more negative value to those that misclassify others.

If two instances have the same rank, we break the tie by assigning different priorities to them. The priority of an instance x is calculated by:

$$priority(x) = \sum_j \frac{1}{d(x, x_j)^2} \quad (6)$$

where x_j is the instance having x as its nearest neighbor and $d(x,x_j)$ is the distance between instances x and x_j. The intuition is that if an instance is far away from its nearest neighbor, it may be noisy or simply atypical of its class, which suggests that it should be discarded earlier. So, if two instances have the same rank, the one with lower priority will be discarded first. Since we have already removed duplicate instances before calculating eq. 6, we can be assured that the denominator of the fraction is non-zero.

Note that our algorithm ranks the instances iteratively. Assuming the training set size is *N*, the algorithm will run *N* times, where in each round it disregards the instance with lowest rank and continues to re-rank the rest of the instances.

Table 2 formalizes the Naïve Rank algorithm. Given the training set *T*, the algorithm will store the instances in list *S* according to their ranks.

Table 2. Naïve Rank Reduction.

```
Function S = Naïve_Rank_Reduction( T )
remove any duplicate instances from T ;
leave-one-out 1-NN classification on T ;
N = size of T;
S = { };
loop_num = 0 ;
while loop_num < N
    for each instance $x_i$ in T - S
        rank($x_i$) is calculated using eq.5 ;
    end
    adjust the rank ties by priorities using eq. 6;
    worst_index = arg_i (min(rank[$x_i$] & min( priority[$x_i$])));
    push(S, $x_{worst\_index}$);    // discard instance with lowest rank
    loop_num = loop_num + 1;
end
```

In the *thresholding* step, we keep the *n* highest ranking instances in *S* as the training set for future classification, where *n* is the threshold given by the user. This threshold may be given based on the maximum space or time the user has for a problem. For example, in recent work on classifying insects with resource limited sensors (Wei & Keogh, 2005), the authors can only afford 200k memory to store the entire training database. It is also possible to give the threshold in other formats, such as "*give me the smallest training dataset with an expected error rate of less than 5%,*" or "*give me the smallest training dataset with a leave-one-out error that does not differ from the error rate on the entire dataset by more than 1%.*"

4. Fast Numerosity Reduction with Adaptive Warping Window

As we will show in our empirical evaluation, at least for time series problems, the Naïve Rank Reduction algorithm is competitive with the more complicated RT algorithms. In this section, we will show an observation which when used together with the Naïve Rank Reduction algorithm produces a dramatic improvement in the accuracy/dataset size tradeoff.

4.1 Adaptive Warping Window

When calculating the DTW distance, warping window constraints (recall Figure 1) are used to prevent pathological warpings, where a uniform constraint (i.e., the "*Sakoe-Chiba band*" in Sakoe & Chiba, 1978) is widely accepted in the data mining community. People may think that wider warping window would contribute to higher accuracy.

However, according to recent research by Ratanamahatana & Keogh (2005), too large a warping window constraint may actually hurt accuracy rather than improving it. Their claim is summarized in the maxim "*a little warping is good while too much warping is bad*." To verify the claim, we experimented on the Gun-Point Dataset using one-nearest-neighbor DTW (a description of the dataset can be found in Ratanamahatana & Keogh, 2005). Each time, half of the instances are randomly removed from the dataset. We try all the warping window size, and show the accuracy in Figure 3.

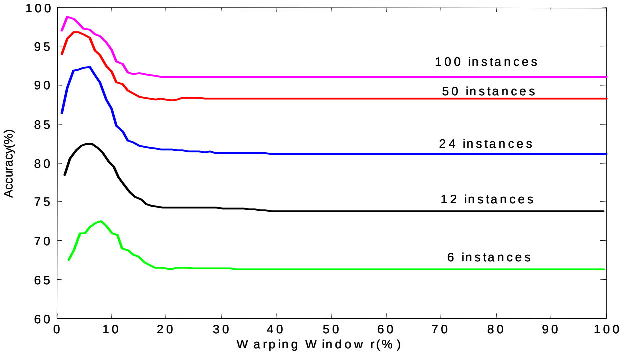

Figure 3. With fewer objects in the dataset, the accuracy decreases and peaks at larger window size.

Figure 3 shows that no matter how big the dataset is, the warping window size *r* that achieves best accuracy is always quite small (We did experiments on other datasets, and obtained similar results; see Keogh, 2006). This is consistent with the claim in Ratanamahatana & Keogh (2005). In addition, two other observations can be made:

- The classification accuracy declines when the size of the dataset decreases, an obvious and expected result.
- Larger warping window size gives better accuracy on smaller datasets, a fact that appears unknown outside of Ratanamahatana & Keogh (2005).

Motivated by the above observations, we propose the algorithm AWARD (Adaptive WARping winDow), which dynamically adjusts warping window size *during* numerosity reduction. The pseudo-code is shown in Table 3. The algorithm first tries all possible warping window sizes from 0% to 100% on the training set *T*, and initializes the warping window size *r* to the one yielding the highest leave-one-out classification accuracy (we call this preprocessing step). Then, the algorithm begins numerosity reduction by calling Naïve Rank Reduction function in Section 3.1.

AWARD differs from the Naïve Rank Reduction in having an additional operator during the reduction procedure. This operator tests the accuracy achieved by window size *r*+1. If this is greater than the accuracy achieved by current window size *r*, we update the window size to *r*+1. In other words, as we search for instances to discard, we also consider the possibility of slightly expanding the warping window.

Note that AWARD algorithm is used in the training phase to find the warping window sizes for all possible threshold values from the size of training set to the number of classes. The pairs of the threshold and the corresponding warping window size are recorded in the table *WarpingT*. Later, when we start classifying the test set, given a user-defined threshold *n*, we would know which subset of instances to keep and which window size to use by simple table lookup.

For example, if the size of the training set is 1,000, and initially, we find *r* = 3 is the best window size to use, we will insert an entry (1000, 3) into the table. If, after discarding 200 instances, we find that the window size should be changed to 4 to maintain the accuracy, we then insert another entry (800, 4) into the table. We will show later that keeping track of this table also helps us easily adapt the algorithm to an anytime algorithm in Section 4.3. Note that the *1NN_DTW_LB* function (line 6, 7) uses dataset *S* to classify dataset *T* with one-nearest-neighbor approach, and the distance measure used is DTW with warping window size *r*. To accelerate classification, it adopts the widely used LB_Keogh lower bounding technique proposed in Keogh (2002) to locate the nearest neighbor.

Table 3. Numerosity reduction with adaptive warping window.

```
Function [WarpingT ] = AWARD(T )
1    initialize r to best warping window size on T;
2    N = size of T;
3    S = Naïve_Rank_Reduction (T );
4    while (N > num_of_class)
5      remove the instance with lowest rank from S ;
6      accuracy_1 = 1NN_DTW_LB (S, T, r );
7      accuracy_2 = 1NN_DTW_LB (S, T, r + 1 );
8      if (accuracy_2 > accuracy_1 )          // should we expand r ?
9          r = r + 1;
10     end
11     insert (N, r ) into WarpingT;
12     N = N -1 ;
13   end
```

The time complexity of AWARD is O(n^3) since the loop will be executed *n* times, and each time it calls function *1NN_DTW_LB* once whose time complexity is O(n^2). This is already much faster than the *brute force* approach which finds the nearest neighbor directly without using LB_Keogh lower bounding. In the next section, we explore the possibility of reusing the results of previous computation to speed up the algorithm even more.

4.2 Fast Numerosity Reduction

In AWARD algorithm, every time one instance is discarded from the training set, we compute the classification accuracy using warping window size *r* and *r*+1 from scratch, which is actually unnecessary. Recall that when we are looking for the best warping window size at the beginning, we have already found the nearest neighbor for each instance (along with the distance between them). We can keep this information for future use and thus reduce the computational complexity.

For this purpose, we keep three structures during the computation, nearest neighbor matrix *A*, distance matrix *B*, and accuracy array *ACC*. Matrix A_r records the nearest neighbor of each instance using warping window size *r*. Its entries are defined as

$$A_r(i,j)=\begin{cases}1 & \text{if } o_j \text{ is } o_i\text{'s} \ \text{ nearest neighbor}\\ 0 & \text{otherwise}\end{cases} \tag{7}$$

Matrix B_r records the distance between two instances using warping window size r. Since we use LB_Keogh lower bound here, we do not need to actually calculate the distance between each pair of instances. To differentiate the actual DTW distance and the distance lower bound, we represent the bounds as negative values. More concretely, the entries in matrix B_r are defined as

$$B_r(i,j)=\begin{cases}d(o_i,o_j) & \text{if } d(o_i,o_j) \text{ is the actual distance}\\ -d(o_i,o_j) & \text{if } d(o_i,o_j) \text{ is the lower bound}\end{cases} \tag{8}$$

Array *ACC* records the accuracy achieved by different warping window. The r^{th} element of *ACC* is the number of objects being correctly classified with warping window *r*.

During the numerosity reduction procedure, every time an instance o_p is discarded from the training set, we need to update A_r, B_r, *ACC*, and check whether *ACC*[*r*+1] is greater than *ACC*[*r*]. This is easy since only those instances that chose o_p as their nearest neighbors will be affected and their corresponding entries need to be updated. The algorithm is given in Table 4. As we will show, this optimization speeds up the reduction process by many orders of magnitude.

Table 4. Fast classification with numerosity reduction.

```
Function [WarpingT] = FastAWARD( T )
    initialize r to best warping window size on T;
    initialize A_k, B_k and ACC for k = 0 to 100;
    N = size of T;
    S = Naïve_Rank_Reduction (T );
    while (N > num_of_class)
        remove the instance o_p with lowest rank from S ;
        for k = r to 100
            [ A_k, B_k, ACC [k ] ] = Update(A_k, B_k, ACC [k ], o_p );
        end
        if (ACC [r +1] > ACC [r ] )
            r = r + 1;
        end
        insert (N, r ) into WarpingT ;
        N = N – 1;
    end

Function [A, B, accuracy ] = Update( A, B, accuracy, o_p )
    A(:, p) = 0;                       // set column p in A to zero
    B(:, p) = inf;                     // set column p in B to infinity
    for each object o_i having o_p as its nearest neighbor
        find a new nearest neighbor o_p' for o_i ;
        A(i, p' ) = 1;                 // update matrix A
        update the i th row of B accordingly;   // update matrix B
        if label(o_i ) == label(o_p) && label(o_i ) != label(o_p' )
            accuracy = accuracy – 1;
        end
        if label(o_i) != label(o_p) && label(o_i) == label(o_p')
            accuracy = accuracy + 1;
        end
    end
```

Since FastAWARD gives identical *accuracy* results to AWARD by definition, we will use the two interchangeably except when discussing *efficiency*.

4.3 Anytime Classification Algorithm

Classification of truly massive time series datasets may be very time consuming (for example, it may take days or even weeks to run). In some domains, it can be useful if users can interrupt the algorithm at any time and still get an output of a certain quality. Algorithms allowing this are called anytime algorithms (Grass & Zilberstein, 1996). Fortunately, we can take our AWARD algorithm and trivially convert it into an anytime algorithm, by visiting the instances in the order defined by the ranking algorithm, the experimental results are given in Section 5.3.

5. Experimental Results

In this section, we test our approach with a comprehensive set of experiments. Note that in order to allow reproducibility, all the datasets and additional experiment results are freely available in Keogh (2006).

5.1 Effectiveness of Our Approach

We begin by considering four approaches: the *AWARD* algorithm introduced in this work; *RankFix*, fixed window with naïve rank reduction*;* *RandomFix*, fixed window with random reduction; *RandomEu*, random reduction using Euclidean distance. Among the four approaches, *AWARD*, *RankFix*, and *RandomFix* use DTW distance, while *Random-Eu* uses Euclidean distance. Note that *AWARD* changes the size of the warping window adaptively, while *RankFix* and *RandomFix* first find the best warping window on the whole training set and stick to this best window during reduction.

In each experiment, we randomly split the data into training set and test set, which have zero intersection. For *AWARD* approach, we use the training set to learn the *thresholding* (subset of data to keep) and the corresponding warping window size, which are recorded in table *WarpingT* as shown in Table 4. The other three approaches follow the same procedure, except that their warping window sizes are fixed (*RankFix* and *RandomFix*) at the value that maximizes accuracy for the full training dataset, or they have no warping window size to record (*RandomEu*). We evaluate the accuracy on the test set for every possible user threshold *n* from the *num_of_class* to the full size of training set.

5.1.1 Two Patterns Dataset

The Two Patterns Dataset was introduced in Geurts (2002). It contains 5,000 instances, each of length 128. There are four classes, denoting the presence of two patterns in a definite order, namely, down-down, up-down, down-up, and up-up. We randomly choose 1,000 instances as training set, and use the remaining 4,000 instances as test set. Figure 4 shows the classification accuracies obtained on the test set by four approaches for different values of *n*. The numbers above the *x*-axis show the warping window size used by the *AWARD* approach in that area. However, *RankFix* and *RandomFix* always use warping window size of 4%.

There are two obvious observations from Figure 4. First, the *AWARD* approach always achieves the highest accuracy, even when a large fraction of the training data has been discarded. This empirically demonstrates the utility of adjusting the warping window when the size of the training set changes. Second, the three approaches (*AWARD*, *RankFix*, and *RandomFix*) using DTW distance measure obtain significantly higher accuracy than the one using the Euclidean distance (*RandomEu*). This verifies the claim in Section 2.2 that generally DTW is better than Euclidean in time series classification.

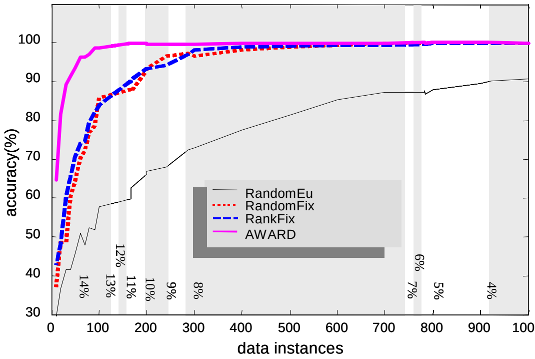

Figure 4. Classification accuracies of four different approaches on Two Patterns test set. The *x*-axis represents *n*, the number of instances in the training set. Note that since we discard instances from the training set gradually, the *x*-axis should be read from right to left.

We have already shown that *AWARD* gives better accuracy than *RankFix*. And then we will compare our *AWARD* approach to the most referenced approaches in the numerosity reduction literature, namely *RT1*, *RT2*, and *RT3* (Wilson & Martinez, 1997). As in the previous experiments, we split the data into training and test sets and use the information learned from the training set to classify data in the test set. The results are shown in Figure 5. It is easy to see that *AWARD* always beats fixed window algorithms, no matter which numerosity reduction technique is used. This strongly suggests that it is the strategy of changing warping window size dynamically that dominates the performance.

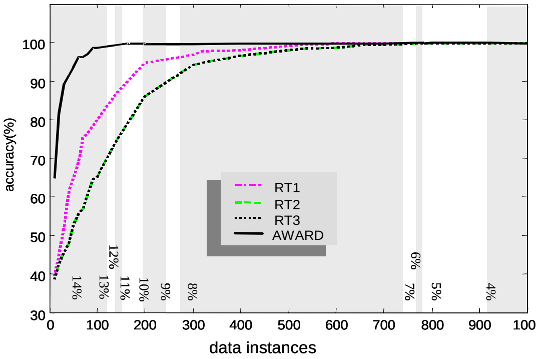

Figure 5. Classification accuracies of four different approaches on Two Patterns test set. Note that for this experiment, the performances of *RT2* and *RT3* are similar; so, their curves overlap.

5.1.2 Leaf Dataset

This dataset comprises of 1,125 Swedish leaf images with 15 classes. It has been shown that images can be converted into "pseudo time series" (Ratanamahatana & Keogh, 2005). Here, we have converted each leaf image into a time series by measuring the distances between the contour points to its centroid. We randomly choose 500 leaf images as training set and the remaining 625 as test set. From Figure 6, we can see that *AWARD* outperforms the *RT* algorithms that use fixed warping window. This again demonstrates that changing the warping window size is more effective than simply pruning the instances from the training set. Note that with our algorithm, we can discard all but one example of each class and still achieve high accuracy.

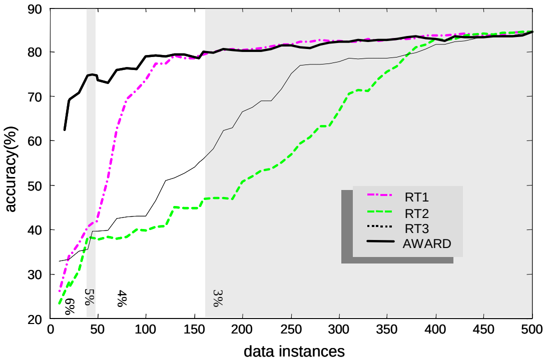

Figure 6. Classification accuracies of four different approaches on Swedish Leaf test set.

5.2 Efficiency of Our Approach

We compare *AWARD* and *FastAWARD* to brute force approach, to see how much speedup could be achieved by our various optimizations in Section 4.2. Brute force is the most straightforward approach that implements DTW with *Sakoe-Chiba band*. For these three algorithms, because the time spent in preprocessing step is essentially identical, i.e., finding the best warping window size *r*, which only counts for a small fraction of total execution time, we only compare the time spent on DTW computation during numerosity reduction, which is computationally expensive.

We conduct the experiments on Two Patterns dataset, and the results are shown in Table 5. It is easy to see that *AWARD* is already more than ten times faster than brute force, in terms of both the processing time and the DTW computation involved. The speedup achieved by *FastAWARD* is even more dramatic.

Table 5. Time comparison on Two Patterns dataset.

	FastAWARD	AWARD	Brute Force
Number of DTW computation	53	7×10^7	6×10^8
Processing Time (sec)	2	2×10^5	3×10^6

5.3 Anytime Classification

In Section 4, we show how to make fast classification using numerosity reduction. Note that we could trivially convert *AWARD* into an anytime algorithm through selecting the instances with highest ranks first, instead of pruning instances. For Two Patterns dataset, with trained warping window sizes from the 1000-instance training set, we apply an interruptible classification algorithm to test set and compare four strategies: *RandomEu*, *RandomFix*, *RankFix*, and *AWARD*. We choose instances in the descending order of their ranks. At the beginning, *AWARD* uses wider window size of 14%, and decreases it gradually as more instances are selected. Result shows that *AWARD* is much better than other three algorithms in Figure 7. If only first 120 instances are examined before the algorithm is interrupted, *AWARD* can achieve accuracy as high as 99.85%, while *RankFix* and *RandomFix* (which use 4% as fixed warping window size) get about 87%.

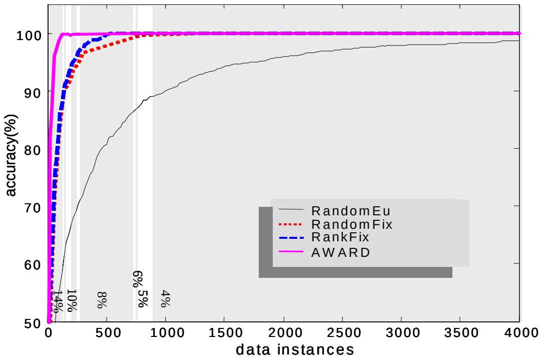

Figure 7. Anytime classification of Two Patterns test set. *AWARD* gives almost 100% accuracy at very early stage. The *x*-axis should be read from left to right, and the number above the *x*-axis shows the warping window size used by *AWARD*.

6. CONCLUSIONS

In this work, we have summarized the vast literature on time series classification and shown that the simple combination of one-nearest-neighbor with DTW distance is exceptionally difficult to beat. We further show that we can leverage a little-known relationship between dataset size and warping constraints to produce very compact classifiers that lose little or nothing in terms of accuracy.

References

Chen, L. & Kamel, M.S. (2005). Design of Multiple Classifier Systems for Time Series Data. *Multiple Classifier Systems*, pp. 216-225.

Chen, L., Özsu, M.T., & Oria, V. (2005). Using Multi-Scale Histograms to Answer Pattern Existence and Shape Match Queries. *SSDBM '05*.

Dasarathy, B.V. (1991). Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. *IEEE Computer Society Press*, pp. 388-397.

Eads, D., Glocer, K., Perkins, S., & Theiler, J. (2005). Grammar-guided feature extraction for time series classification. *NIPS ‘05*.

Fu, A.W., Keogh, E., Lau, L.Y.H., & Ratanamahatana, C.A. (2005). Scaling and Time Warping in Time Series Querying. *VLDB ‘05*, pp. 649-660.

Geurts, P. (2002). Contributions to decision tree induction: bias/variance tradeoff and time series classification. Ph.D. thesis, University of Liege.

Grass, J. & Zilberstein, S. (1996). Anytime Algorithm Development Tools. *Sigart Artificial Intellligence*, Vol 7, ACM Press.

Han, J. & Kamber, M. (2000). *Data Mining Concepts and Techniques*. Morgan Kaufmann Publishers.

Hayashi, A., Mizuhara, Y., & Suematsu, N. (2005). Embedding Time Series Data for Classification. *Machine Learning and Data Mining in Pattern Recognition*, pp. 356-365.

Karydis, I., Nanopoulos, A., Papadopoulos, A., & Manolopoulos, Y. (2005). Music Retrieval in P2P Networks Under the Warping Distance, *7th International Conference on Enterprise Information Systems*.

Keogh, E. (2002). Exact Indexing of Dynamic Time Warping. *VLDB ‘02*, pp. 406-417, Hong Kong, Aug 20-23.

Keogh, E. (2006). UCR Time Series Archive www.cs.ucr.edu/~eamonn/TSDMA/

Kim, S., Smyth, P., & Luther, S. (2004). Modeling waveform shapes with random effects segmental hidden Markov Models. Technical Report, UCI-ICS 04-05.

Lei, H. & Govindaraju, V. (2004). Regression Time Warping for Similarity Measure of Sequence. *CIT’04*, pp. 826-830.

Megalooikonomou, V., Wang, Q., Li, G., & Faloutsos, C. A. (2005). Multiresolution Symbolic Representation of Time Series. *ICDE ‘05*, pp. 668-679.

Megalooikonomou, V. (2006). Personal Communication.

Nanopoulos, A., Alcock, R., & Manolopoulos, Y. (2001). Feature-based Classification of Time-series Data. *International Journal of Computer Research*, pp. 49-61.

Pekalska, E., Duin, R. P.W., & Paclik, P. (2006). Prototype Selection for Dissimilarity-Based Classifiers. *Pattern Recognition*, 39:2, pp. 189-208.

Ratanamahatana, C.A. & Keogh, E. (2005). Three myths about Dynamic Time Warping Data Mining. *SDM '05* .

Rodríguez, J.J. & Alonso, C.J. (2004). Interval and dynamic time warping-based decision trees. In *Proceedings of the 2004 ACM symposium on Applied computing (SAC)*, pp. 548-552.

Rodríguez, J.J., Alonso, C.J., & Boström, H. (2000). Learning First Order Logic Time Series Classifiers: Rules and Boosting. *PKDD ‘00*, pp. 299-308.

Sakoe, H. & Chiba, S. Dynamic programming algorithm optimization for spoken word recognition. (1978). *IEEE Trans. Acoustics, Speech, and Signal Proc.*, Vol. ASSP-26.

Shou, Y., Mamoulis, N., & Cheung, D.W. (2005). Fast and exact warping of time series using adaptive segmental approximations. *Machine Learning*, Vol 28, pp. 231-267.

Wei, L., Keogh, E., Van Herle, H., & Mafra-Neto, A. (2005). Atomic Wedgie: Efficient Query Filtering for Streaming Time Series. *ICDM ’05*, pp. 490-497.

Wilson, D.R. & Martinez, T.R. (1997). Instance Pruning Techniques. *ICML’97*, Morgan Kaufmann, pp. 403-411.

Wu, Y. & Chang, E.Y. (2004). Distance-function design and fusion for sequence data. *CIKM ‘04*, pp. 324-333.

Zhu, Y. & Shasha, D. (2003). Query by Humming: a Time Series Database Approach, *SIGMOD ‘03*.