Time Series Shapelets: A New Primitive for Data Mining

Lexiang Ye
Dept. of Computer Science & Engineering
University of California, Riverside, CA 92521
lexiangy@cs.ucr.edu

Eamonn Keogh
Dept. of Computer Science & Engineering
University of California, Riverside, CA 92521
eamonn@cs.ucr.edu

ABSTRACT

Classification of time series has been attracting great interest over the past decade. Recent empirical evidence has strongly suggested that the simple nearest neighbor algorithm is very difficult to beat for most time series problems. While this may be considered good news, given the simplicity of implementing the nearest neighbor algorithm, there are some negative consequences of this. First, the nearest neighbor algorithm requires storing and searching the entire dataset, resulting in a time and space complexity that limits its applicability, especially on resource-limited sensors. Second, beyond mere classification accuracy, we often wish to gain some insight into the data.

In this work we introduce a new time series primitive, *time series shapelets*, which addresses these limitations. Informally, shapelets are time series subsequences which are in some sense maximally representative of a class. As we shall show with extensive empirical evaluations in diverse domains, algorithms based on the time series shapelet primitives can be interpretable, more accurate and significantly faster than state-of-the-art classifiers.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications – Data Mining

General Terms

Algorithms, Experimentation

1. INTRODUCTION

While the last decade has seen a huge interest in time series classification, to date the most accurate and robust method is the simple nearest neighbor algorithm [4][12][14]. While the nearest neighbor algorithm has the advantages of simplicity and not requiring extensive parameter tuning, it does have several important disadvantages. Chief among these are its space and time requirements, and the fact that it does not tell us anything about *why* a particular object was assigned to a particular class.

In this work we present a novel time series data mining primitive called *time series shapelets*. Informally, shapelets are time series subsequences which are in some sense maximally representative of a class. While we believe shapelets can have many uses in data mining, one obvious implication of them is to mitigate the two weaknesses of the nearest neighbor algorithm noted above.

KDD'09, June 29–July 1, 2009, Paris, France

Because we are defining and solving a new problem, we will take some time to consider a detailed motivating example. Figure 1 shows some examples of leaves from two classes, *Urtica dioica* (stinging nettles) and *Verbena urticifolia*. These two plants are commonly confused, hence the colloquial name "false nettle" for *Verbena urticifolia*.

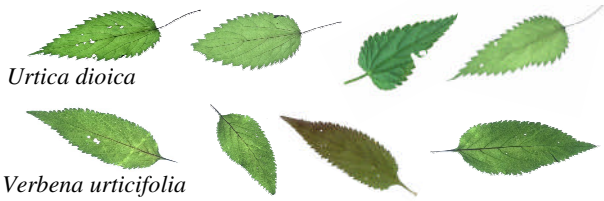

Figure 1: Samples of leaves from two species. Note that several leaves have the insect-bite damage

Suppose we wish to build a classifier to distinguish these two plants; what features should we use? Since the intra-variability of color and size within each class completely dwarfs the inter-variability between classes, our best hope is based on the shapes of the leaves. However, as we can see in Figure 1, the differences in the global shape are very subtle. Furthermore, it is very common for leaves to have distortions or "occlusions" due to insect damage, and these are likely to confuse any global measures of shape. Instead we attempt the following. We first convert each leaf into a one-dimensional representation as shown in Figure 2.

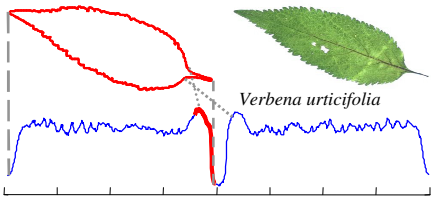

Figure 2: A shape can be converted into a one dimensional "time series" representation. The reason for the highlighted section of the time series will be made apparent shortly

Such representations have been successfully used for the classification, clustering and outlier detection of shapes in recent years [8]. However, here we find that using a nearest neighbor classifier with either the (rotation invariant) Euclidean distance or Dynamic Time Warping (DTW) distance does not significantly outperform random guessing. The reason for the poor performance of these otherwise very competitive classifiers seems to be due to the fact that the data is somewhat noisy (i.e. insect bites, and different stem lengths), and this noise is enough to swamp the subtle differences in the shapes.

Suppose, however, that instead of comparing the *entire* shapes, we only compare a *small* subsection of the shapes from the two classes that is particularly discriminating. We can call such subsections *shapelets*, which invokes the idea of a small "sub-shape." For the moment we ignore the details of how to formally define shapelets, and how to efficiently compute them. In Figure 3, we see the shapelet discovered by searching the small dataset shown in Figure 1.

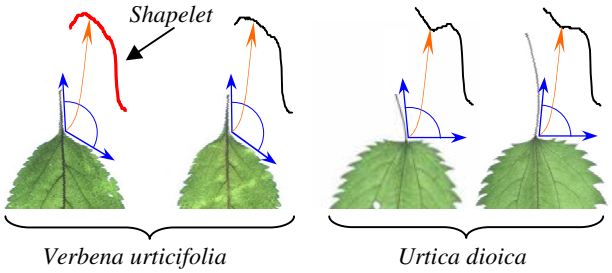

Figure 3: Here, the shapelet hinted at in Figure 2 (in both cases shown with a bold line), is the subsequence that best discriminates between the two classes

As we can see, the shapelet has "discovered" that the defining difference between the two species is that *Urtica dioica* has a stem that connects to the leaf at almost 90 degrees, whereas the stem of *Verbena urticifolia* connects to the leaf at a much shallower angle. Having found the shapelet and recorded its distance to the nearest matching subsequence in all objects in the database, we can build the simple decision-tree classifier shown in Figure 4.

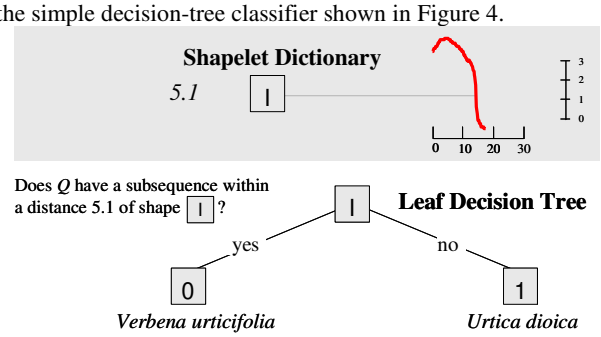

Figure 4: A decision-tree classifier for the leaf problem. The object to be classified has all of its subsequences compared to the shapelet, and if any subsequence is less than (the empirically determined value of) 5.1, it is classified as *Verbena urticifolia*

The reader will immediately see that this method of classification has many potential advantages over current methods:

- Shapelets can provide interpretable results, which may help domain practitioners better understand their data. For example, in Figure 3 we see that the shapelet can be summarized as the following: "*Urtica dioica* has a stem that connects to the leaf at almost 90 degrees." Most other state-of-the-art time series/shape classifiers do not produce interpretable results [4][7].
- Shapelets can be significantly more accurate/robust on some datasets. This is because they are local features, whereas most other state-of-the-art time series/shape classifiers consider global features, which can be brittle to even low levels of noise and distortions [4]. In our example, leaves which have insect bite damage are still usually correctly classified.
- Shapelets can be significantly faster at classification than existing state-of-the-art approaches. The classification time is just O(*ml*), where *m* is the length of the query time series and *l* is the length of the shapelet. In contrast, if we use the best performing global distance measure, rotation invariant DTW distance [8], the time complexity is on the order of O(km^3), where *k* is the number of reference objects in the training set. On real-world problems the speed difference can be greater than three orders of magnitude.

The leaf example, while from an important real-world problem in botany, is a contrived and small example to help develop the reader's intuitions. However, as we shall show in Section 5, we can provide extensive empirical evidence for all of these claims, on a vast array of problems in domains as diverse as anthropology, human motion analysis, spectrography, and historical manuscript mining.

2. RELATED WORK AND BACKGROUND

While there is a vast amount of literature on time series classification and mining [4][7][14], we believe that the problem we intend to solve here is unique. The closest work is that of [5]. Here the author also attempts to find local patterns in a time series which are predictive of a class. However, the author considers the problem of finding the *best* such pattern intractable, and thus resorts to examining a single, randomly chosen instance from each class, and even then only considering a reduced piecewise constant approximation of the data. While the author notes "*it is impossible in practice to consider every such subsignal as a candidate pattern,*" this is in fact *exactly* what we do, aided by eight years of improvements in CPU time, and, more importantly, an admissible pruning technique that can prune off more than 99.9% of the calculations (c.f. Section 5.1). Our work may also be seen as a form of a *supervised* motif discovery algorithm [3].

2.1 Notation

Table 1 summarizes the notation in the paper; we expand on the definitions below.

Table 1: Symbol table

Symbol	Explanation
T, *R*	time series
S	subsequence
m, |*T*|	length of time series
l, |*S*|	length of subsequence
d	distance measurement
D	time series dataset
A,*B*	class label
I	entropy
$\hat{I}$	weighted average entropy
sp	split strategy
k	number of time series objects in dataset
C	classifier
$S_{(k)}$	the *k*th data point in subsequence *S*

We begin by defining the key terms in the paper. For ease of exposition, we consider only a two-class problem. However, extensions to a multiple-class problem are trivial.

Definition 1: *Time Series*. A time series $T = t_1,\ldots,t_m$ is an ordered set of *m* real-valued variables.

Data points $t_1,\ldots,t_m$ are typically arranged by temporal order, spaced at equal time intervals. We are interested in the *local* properties of a time series rather than the *global* properties. A local subsection of time series is termed as a subsequence.

Definition 2: *Subsequence*. Given a time series T of length m, a subsequence S of T is a sampling of length $l \le m$ of contiguous positions from T, that is, $S = t_p,\ldots,t_{p+l-1}$, for $1 \le p \le m - l + 1$.

Our algorithm needs to extract all of the subsequences of a certain length. This is achieved by using a sliding window of the appropriate size.

Definition 3: *Sliding Window*. Given a time series T of length m, and a user-defined subsequence length of l, all possible subsequences can be extracted by sliding a window of size l across T and considering each subsequence S_p^l of T. Here the superscript l is the length of the subsequence and subscript p indicates the starting position of the sliding window in the time series. The set of all subsequences of length l extracted from T is defined as $\mathbf{S}_T^l$, $\mathbf{S}_T^l = \{S_p^l \text{ of } T, \text{ for } 1 \le p \le m - l + 1\}$.

As with virtually all time series data mining tasks, we need to provide a similarity measure between the time series $Dist(T, R)$.

Definition 4: *Distance between the time series*. $Dist(T, R)$ is a distance function that takes two time series T and R which are of the same length as inputs and returns a nonnegative value d, which is said to be the distance between T and R. We require that the function $Dist$ be symmetrical; that is, $Dist(R, T) = Dist(T, R)$.

The $Dist$ function can also be used to measure the distance between two subsequences of the same length, since the subsequences are of the same format as the time series. However, we will also need to measure the similarity between a short subsequence and a (potentially much) longer time series. We therefore define the distance between two time series T and S, with $|S| < |T|$ as:

Definition 5: *Distance from the time series to the subsequence*. $SubsequenceDist(T, S)$ is a distance function that takes time series T and subsequence S as inputs and returns a nonnegative value d, which is the distance from T to S. $SubsequenceDist(T, S) = \min(Dist(S, S'))$, for $S' \in \mathbf{S}_T^{|S|}$.

Intuitively, this distance is simply the distance between S and its best matching location somewhere in T, as shown in Figure 5.

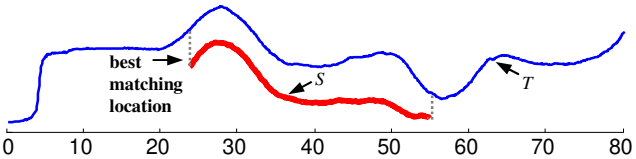

Figure 5: Illustration of best matching location in time series *T* for subsequence *S*

As we shall explain in Section 3, our algorithm needs some metric to evaluate how well it can divide the entire combined dataset into two original classes. Here, we use concepts very similar to the *information gain* used in the traditional decision tree [2]. The reader may recall the original definition of entropy which we review here:

Definition 6: *Entropy*. A time series dataset $\mathbf{D}$ consists of two classes, A and B. Given that the proportion of objects in class A is $p(A)$ and the proportion of objects in class B is $p(B)$, the entropy of $\mathbf{D}$ is: $I(\mathbf{D}) = -p(A)\log(p(A)) - p(B)\log(p(B))$.

Each splitting strategy divides the whole dataset $\mathbf{D}$ into two subsets, $\mathbf{D_1}$ and $\mathbf{D_2}$. Therefore, the information remaining in the entire dataset after splitting is defined by the weighted average entropy of each subset. If the fraction of objects in $\mathbf{D_1}$ is $f(\mathbf{D_1})$ and the fraction of objects in $\mathbf{D_2}$ is $f(\mathbf{D_2})$, the total entropy of $\mathbf{D}$ after splitting is $\hat{I}(\mathbf{D}) = f(\mathbf{D_1})I(\mathbf{D_1}) + f(\mathbf{D_2})I(\mathbf{D_2})$. This allows us to define the information gain for any splitting strategy:

Definition 7: *Information Gain*. Given a certain split strategy sp which divides $\mathbf{D}$ into two subsets $\mathbf{D_1}$ and $\mathbf{D_2}$, the entropy before and after splitting is $I(\mathbf{D})$ and $\hat{I}(\mathbf{D})$. So the information gain for this splitting rule is

$$Gain(sp) = I(\mathbf{D}) - \hat{I}(\mathbf{D}),$$
$$Gain(sp) = I(\mathbf{D}) - f(\mathbf{D_1})I(\mathbf{D_1}) + f(\mathbf{D_2})I(\mathbf{D_2}).$$

As hinted at in the introduction, we use the distance to a *shapelet* as the splitting rule. The shapelet is a subsequence of a time series such that most of the time series objects in one class of the dataset are close to the shapelet under *SubsequenceDist,* while most of the time series objects from the other class are far away from it.

To find the best shapelet, we may have to test many shapelet candidates. In the brute force algorithm discussed in Section 3.1, given a candidate shapelet, we calculate the distance between the candidate and every time series object in the dataset. We sort the objects according to the distances and find an optimal split point between two neighboring distances.

Definition 8: *Optimal Split Point* (*OSP*). A time series dataset $\mathbf{D}$ consists of two classes, A and B. For a shapelet candidate S, we choose some distance threshold d_{th} and split $\mathbf{D}$ into $\mathbf{D_1}$ and $\mathbf{D_2}$, such that for every time series object $T_{1,i}$ in $\mathbf{D_1}$, $SubsequenceDist(T_{1,i}, S) < d_{th}$ and for every time series object $T_{2,i}$ in $\mathbf{D_2}$, $SubsequenceDist(T_{2,i}, S) \ge d_{th}$. An *Optimal Split Point* is a distance threshold that

$$Gain(S, d_{OSP(\mathbf{D}, S)}) \ge Gain(S, d'_{th})$$

for any other distance threshold d'_{th}.

So using the shapelet, the splitting strategy contains two factors: the shapelet and the corresponding optimal split point. As a concrete example, in Figure 4 the shapelet is shown in red in the shapelet dictionary, and the optimal split point is 5.1.

We are finally in the position to formally define the shapelet.

Definition 9: *Shapelet*. Given a time series dataset $\mathbf{D}$ which consists of two classes, A and B, *shapelet*($\mathbf{D}$) is a subsequence that, with its corresponding optimal split point,

$$Gain(shapelet(\mathbf{D}), d_{OSP(\mathbf{D},\ shapelet(\mathbf{D}))}) \ge Gain(S, d_{OSP(\mathbf{D}, S)})$$

for any other subsequence S.

Since the shapelet is simply *any* time series of some length less than or equal to the length of the shortest time series in our dataset, there are an infinite amount of possible shapes it could have. For simplicity, we assume the shapelet to be a subsequence of a time series object in the dataset. It is reasonable to make this assumption since the time series objects in one class presumably contain some similar subsequences, and these subsequences are good candidates for the shapelet.

Nevertheless, there are still a very large number of possible shapelet candidates. Suppose the dataset $\mathbf{D}$ contains k time series objects. We specify the minimum and maximum length of the shapelet candidates that can be generated from this dataset as *MINLEN* and *MAXLEN*, respectively. Obviously *MAXLEN* $\le \min(m_i)$, m_i is the length of the time series T_i from the dataset, $1 \le i \le k$. Considering a certain fixed length l, the number of shapelet candidates generated from the dataset is:

$$\sum_{T_i \in D} (m_i - l + 1)$$

So the *total* number of candidates of all possible lengths is:

$$\sum_{l=MINLEN}^{MAXLEN} \sum_{T_i \in D} (m_i - l + 1)$$

If the shapelet can be any length smaller than that of the shortest time series object in the dataset, the number of shapelet candidates is linear in k, and quadratic in $\overline{m}$, the average length of time series objects. For example, the well-known Trace dataset [11] has 200 instances, each of length 275. If we set *MINLEN*=3, *MAXLEN*=275, there will be 7,480,200 shapelet candidates. For each of these candidates, we need to find its nearest neighbor within the k time series objects. Using the brute force search, it will take approximately three days to accomplish this. However, as we will show in Section 3, we can achieve an identical result in a tiny fraction of this time with a novel pruning strategy.

3. FINDING THE SHAPELET

We first show the brute force algorithm for finding shapelets, followed by two simple but highly effective speedup methods.

3.1 Brute-Force Algorithm

The most straightforward way for finding the shapelet is using the brute force method. The algorithm is described in Table 2.

Table 2: Brute force algorithm for finding shapelet

FindingShapeletBF (dataset **D**, *MAXLEN*, *MINLEN*)	
1	*candidates* ← GenerateCandidates(**D**, *MAXLEN*, *MINLEN*)
2	*bsf_gain* ← 0
3	**For each** *S* **in** *candidates*
4	*gain* ← CheckCandidate(**D**, *S*)
5	**If** *gain* > *bsf_gain*
6	*bsf_gain* ← *gain*
7	*bsf_shapelet* ← *S*
8	**EndIf**
9	**EndFor**
10	**Return** *bsf_shapelet*

Given a combined dataset **D**, in which each time series object is labeled either class *A* or class *B*, along with the user-defined maximum and minimum lengths of the shapelet, line 1 generates all of the subsequences of all possible lengths, and stores them in the unordered list *candidates*. After initializing the best information gain *bsf_gain* to be zero (line 2), the algorithm checks how well each candidate in *candidates* can separate objects in class *A* and class *B* (lines 3 to 7). For each shapelet candidate, the algorithm calls the function CheckCandidate() to obtain the information gain achieved if using that candidate to separate the data (line 4). As illustrated in Figure 6, we can visualize this as placing class-annotated points on the real number line, representing the distance of each time series to the candidate. Intuitively, we hope to find that this mapping produces two well-separated "pure" groups. In this regard the example in Figure 6 is very good, but clearly not perfect.

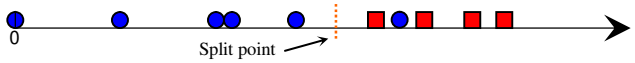

Figure 6: The CheckCandidate() function at the heart of the brute force search algorithm can be regarded as testing to see how mapping all of the time series objects on the number line based on their SubsequenceDist(*T*, *S*) separates the two classes

If the information gain is higher than the *bsf_gain*, the algorithm updates the *bsf_gain* and the corresponding best shapelet candidate *bsf_shapelet* (lines 5 to 7). Finally, the algorithm returns the candidate with the highest information gain in line 10. The two subroutines GenerateCandidates() and CheckCandidate() called in the algorithm are outlined in Table 3 and Table 4, respectively. In Table 3, the algorithm GenerateCandidates() begins by initializing the shapelet candidate pool to be an empty set and the shapelet length l to be *MAXLEN* (lines 1 and 2).

Table 3: Generate all the candidates from time series dataset

GenerateCandidates (dataset **D**, *MAXLEN*, *MINLEN*)	
1	*pool* ← Ø
2	*l* ← *MAXLEN*
3	**While** *l* ≥ *MINLEN*
4	**For** *T* **in D**
5	*pool* ← *pool* ∪ $\mathbf{S}_T^l$
6	**EndFor**
7	*l* ← *l* - 1
8	**EndWhile**
9	**Return** *pool*

Thereafter, for each possible length l, the algorithm slides a window of size l across all of the time series objects in the dataset **D**, extracts all of the possible candidates and adds them to the *pool* (line 5). The algorithm finally returns the *pool* as the entire set of shapelet candidates that we are going to check (line 9). In Table 4 we show how the algorithm evaluates the utility of each candidate by using the information gain.

Table 4: Checking the utility of a single candidate

CheckCandidate (dataset **D**, shapelet candidate *S*)	
1	*objects_histogram* ← Ø
2	**For each** *T* **in D**
3	*dist* ← SubsequenceDist(*T*, *S*)
4	insert *T* into *objects_histogram* by the key *dist*
5	**EndFor**
6	**Return** CalculateInformationGain(*objects_histogram*)

First, the algorithm inserts all of the time series objects into the histogram *objects_histogram* according to the distance from the time series object to the candidate in lines 1 to 4. After that, the algorithm returns the utility of that candidate by calling CalculateInformationGain() (line 6).

Table 5: Information gain of distance histogram optimal split

CalculateInformationGain (distance histogram *obj_hist*)	
1	*split_dist* ← OptimalSplitPoint(*obj_hist*)
2	$\mathbf{D_1}$ ← Ø, $\mathbf{D_2}$ ← Ø
3	**For** *d* **in** *obj_hist*
4	**If** *d.dist* < *split_dist*
5	$\mathbf{D_1}$ ← $\mathbf{D_1}$ ∪ *d.objects*
6	**Else**
7	$\mathbf{D_2}$ ← $\mathbf{D_2}$ ∪ *d.objects*
8	**EndIf**
9	**EndFor**
10	**Return** $I(\mathbf{D})$ - $\hat{I}(\mathbf{D})$

The CalculateInformationGain() subroutine, as shown in Table 5, takes an object histogram as the input, finds an optimal split point *split_dist* (line 1) and divides the time series objects into two subsets by comparing the distance to the candidate with *split_dist* (lines 4 to 7). Finally, it calculates the information gain (cf. definitions 6, 7) of the partition and returns the value (line 10).

After building the distance histogram for all of the time series objects to a certain candidate, the algorithm will find a split point that divides the time series objects into two subsets (denoted by the dashed line in Figure 6). As noted in definition 8, an optimal split point is a distance threshold. Comparing the distance from each time series object in the dataset to the shapelet with the threshold, we can divide the dataset into two subsets, which achieves the highest information gain among all of the possible partitions. Any point on the positive real number line could be a split point, so there are infinite possibilities from which to choose. To make the search space smaller, we check only the mean values

of each pair of adjacent points in the histogram as a possible split point. This reduction still finds all of the possible information gain values since the information gain cannot change in the region *between* two adjacent points. Furthermore, in this way, we maximize the margin between two subsets.

The naïve brute force algorithm clearly finds the optimal shapelet. It appears that it is extremely space inefficient, requiring the storage of all of the shapelet candidates. However, we can mitigate this with some internal bookkeeping that generates and then discards the candidates one at a time. Nevertheless, the algorithm suffers from high time complexity. Recall that the number of the time series objects in the dataset is k and the average length of each time series is $\overline{m}$. As we discussed in Section 2.1, the size of the candidate set is $O(\overline{m}^2 k)$. Checking the utility of one candidate takes $O(\overline{m}k)$. Hence, the overall time complexity of the algorithm is $O(\overline{m}^3 k^2)$, which makes the real-world problems intractable.

3.2 Subsequence Distance Early Abandon

In the brute force method, the distance from the time series T to the subsequence S is obtained by calculating the Euclidean distance of every subsequence of length $|S|$ in T and S and choosing the minimum. This takes $O(|T|)$ distance calculations between subsequences. However, all we need to know is the *minimum* distance rather than all of the distances. Therefore, instead of calculating the exact distance between every subsequence and the candidate, we can stop distance calculations once the partial distance exceeds the minimum distance known so far. This trick is known as *early abandon* [8], which is very simple yet has been shown to be extremely effective for similar types of problems [8].

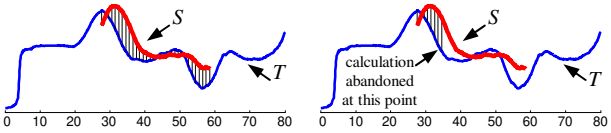

Figure 7: (*left*) Illustration of complete Euclidean distance. (*right*) Illustration of Euclidean distance early abandon

While it is a simple idea, for clarity we illustrate the idea in Figure 7 and provide the pseudo code in Table 6.

Table 6: Early abandon the non-minimum distance

```
SubsequenceDistanceEarlyAbandon(T, S)
1   min_dist ← ∞
2   stop ← False
3   For S_i in S_T^|S|
4       sum_dist ← 0
5       For k ← 1 to |S|
6           sum_dist ← sum_dist + (S_i(k) − S(k))^2
7           If sum_dist ≥ min_dist
8               stop ← True
9               Break
10          EndIf
11      EndFor
12      If not stop
13          min_dist ← sum_dist
14      EndIf
15  EndFor
16  Return min_dist
```

In line 1, we initialize the minimum distance *min_dist* from the time series T to the subsequence S to be infinity. Thereafter, for each subsequence S_i from T of length $|S|$, we accumulate the distance *sum_dist* between S_i and S, one data point at a time (line 6). Once *sum_dist* is larger than or equal to the minimum distance known so far, we abandon the distance calculation between S_i and S (lines 7 to 9). If the distance calculation between S_i and S finishes, we know that the distance is smaller than the minimum distance known so far. Thus, we update the minimum distance *min_dist* in line 13. The algorithm returns the true distance from the time series T to the subsequence S in line 16. Although the early abandon search is still $O(|T|)$, as we will demonstrate later, this simple trick reduces the time required by a large, constant factor.

3.3 Admissible Entropy Pruning

Our definition of the shapelet requires some measure of how well the distances to a given time series subsequence can split the data into two "purer" subsets. The reader will recall that we used the information gain (or entropy) as that measure. However, there are other commonly used measures for distribution evaluation, such as the Wilcoxon signed-rank test [13]. We adopted the entropy evaluation for two reasons. First, it is easily generalized to the multi-class problem. Second, as we will now show, we can use a novel idea called *early entropy pruning* to avoid a large fraction of distance calculations required when finding the shapelet.

Obtaining the distance between a candidate and its nearest matching subsequence of each of the objects in the dataset is the most expensive calculation in the brute force algorithm, whereas the information gain calculation takes an inconsequential amount of time. Based on this observation, instead of waiting until we have all of the distances from each of the time series objects to the candidate, we can calculate an *upper bound* of the information gain based on the currently observed distances. If at any point during the search the upper bound cannot beat the best-so-far information gain, we stop the distance calculations and prune that particular candidate from consideration, secure in the knowledge that it cannot be a better candidate than the current best so far.

In order to help the reader understand the idea of pruning with an upper bound of the information gain, we consider a simple example. Suppose as shown in Figure 8, ten time series objects are arranged in a one-dimensional representation by measuring their distance to the best-so-far candidate. This happens to be a good case, with five of the six objects from class *A* (represented by circles) closer to the candidate than any of the four objects from class *B* (represented by squares). In addition, of the five objects to the right of the split point, only one object from class *A* is mixed up with the class *B*. The optimal split point is represented by a vertical dashed line, and the best-so-far information gain is:

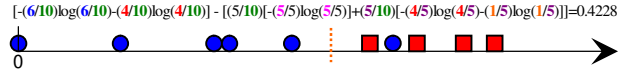

Figure 8: Distance arrangement of the time series objects in one-dimensional representation of best-so-far information gain. The positions of the objects represent their distances to the candidate

We now consider another candidate. The distances of the first five time series objects to the candidate have been calculated, and their corresponding positions in a one-dimensional representation are shown in Figure 9.

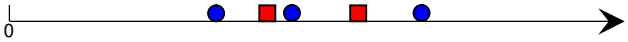

Figure 9: The arrangement of first five distances from the time series objects to the candidate

We can ask the following question: of the 30,240 distinct ways the remaining five distances could be added to this line, could any of them results in an information gain that is better than the best so far? In fact, we can answer this question in constant time. The idea is to imagine the most optimistic scenarios and test them. It is clear that there are only two optimistic possibilities: either all of the remaining class *A* objects map to the far right and all of the class *B* objects map to the far left, or vice versa. Figure 10 shows the former scenario applied to the example shown in Figure 9.

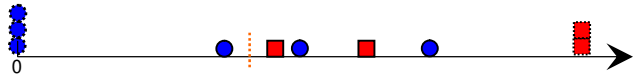

Figure 10: One optimistic prediction of distance distribution based on distances that have already been calculated in Figure 9. The dashed objects are in the optimistically assumed placements

The information gain of the better of the two optimistic predictions is:

[-(6/10)log(6/10)-(4/10)log(4/10)] - [(4/10)[-(4/4)log(4/4)]+(6/10)[-(4/6)log(4/6)-(2/6)log(2/6)]]=0.2911

which is *lower* than the best-so-far information gain. Therefore, at this point, we can stop the distance calculation for the remaining objects and prune this candidate from consideration forever. In this case, we saved 50% of the distance calculations. But in real-life situations, early entropy pruning is generally much more efficient than we have shown in this brief example. We will empirically evaluate the time we save in Section 5.1.

This intuitive idea is formalized in the algorithm outlined in Table 7. The algorithm takes as the inputs the best-so-far information gain, the calculated distances from objects to the candidate organized in the histogram (i.e the number line for Figures 8, 9 and 10) and the remaining time series objects in class *A* and class *B*, and returns TRUE if we can prune the candidate as the answer. The algorithm begins by finding the two ends of the histogram (discussed in Section 3.1). For simplicity, we make the distance values at two ends as 0 and maximum distance +1 (in lines 1 and 2). To build the optimistic histogram of the whole dataset based on the existing one (lines 3 and 8), we assign the remaining objects of one class to one end and those of the other class to the other end (lines 4 and 9). If in either case, the information gain of the optimistic histogram is higher than the best so far (lines 5 and 10), it is still possible that the actual information gain of the candidate can beat the best so far. Thus, we should continue to test the candidate(lines 6 and 11). Otherwise, if the upper bound of the actual information gain is lower than the best so far, we save all of the remaining calculations with this candidate (line 13).

Table 7: Information gain upper bound pruning

EntropyEarlyPrune (*bsf_gain*, *dist_hist*, c_A, c_B)	
1	*minend* ← 0
2	*maxend* ← largest distance value in *dist_hist* + 1
3	*pred_dist_hist* ← *dist_hist*
4	Add to the *pred_dist_hist*, c_A at *minend* and c_B at *maxend*
5	**If** CalculateInformationGain (*pred_dist_hist*) > *bsf_gain*
6	**Return FALSE**
7	**EndIf**
8	*pred_dist_hist* ← *dist_hist*
9	Add to the *pred_dist_hist*, c_A at *maxend* and c_B at *minend*
10	**If** CalculateInformationGain (*pred_dist_hist*) > *bsf_gain*
11	**Return FALSE**
12	**EndIf**
13	**Return TRUE**

The utility of this pruning method depends on the data. If there is any class-correlated structure in the data, we will typically find a good candidate that gives a high information gain early in our search, and thereafter the vast majority of candidates will be pruned quickly.

There is one simple trick we can do to get the maximum pruning benefit. Suppose we tested all of the objects from class *A* first, then all of the objects from class *B*. In this case, the upper bound of the information gain must always be maximum until at least after the point at which we have seen the first object from class *B*. We therefore use a round-robin algorithm to pick the next object to be tested. That is to say, the ordering of objects we use is a_1, b_1, a_2, b_3,…, a_n, b_n. This ordering lets the algorithm know very early in the search if a candidate cannot beat the best so far.

It is often the case that different candidates will have the same best information gain. This is particularly true for small datasets. We propose several options to break this tie depending on applications. We can break such ties by favoring the longest candidate, the shortest candidate or the one that achieves the largest margin between the two classes. We omit a more detailed discussion of this minor issue for brevity.

4. SHAPELETS FOR CLASSIFICATION

While we believe that shapelets can have implications for many time series data mining problems, including visualization, anomaly detection and rule discovery, for brevity we will focus just on the classification problem in this work.

Classifying with a shapelet and its corresponding split point produces a binary decision as to whether a time series belongs to a certain class or not. Obviously, this is not enough to deal with a multi-class situation. Even with two-class problems, a linear classifier is sometimes inadequate. In order to make the shapelet classifier universal, we frame it as a decision tree [2]. Given the discussion of the information gain above, this is a natural fit.

At each step of the decision tree induction, we determine the shapelet and the corresponding split point over the training subset considered in that step. (A similar idea is considered in [5].)

After the learning procedure finishes, we can assess the performance of the shapelet decision tree classifier by calculating the accuracy on the testing dataset. The way we predict the class label of each testing time series object is very similar to the way this is done with a traditional decision tree. For concreteness the algorithm is described in Table 8.

Table 8: Calculating the accuracy on the shapelet classifier

CalculateAccuracy (shapelet decision tree classifier **C**, dataset $\mathbf{D}_t$)	
1	**For each** *T* **in** $\mathbf{D}_t$
2	predict_class_label ← Predict(**C**, *T*)
3	**If** predict_class_label is the same as actual class label
4	correct ← correct + 1
5	**EndIf**
6	**EndFor**
7	**Return** correct / $\lvert \mathbf{D}_t \rvert$

The technique to predict the class label of each testing object is described in Table 9. For each node of the decision tree, we have the information of a single shapelet classifier, the left subtree and the right subtree. For the leaf node, there is additional information of a predicted class label. Starting from the root of a shapelet decision tree classifier, we calculate the distance from the testing object *T* to the shapelet in that node. If the distance is smaller than the split point, we recursively use the left subtree (lines 6 and 7) and otherwise use the right subtree (lines 8 and 9). This procedure continues until we reach the leaf node and return the predicted class label (lines 1 and 2).

Table 9: Predicting the class label of a testing object

Predict (shapelet decision tree classifier **C**, testing time series *T*)	
1	**If C** is the leaf node
2	**Return** label of **C**
3	**Else**
4	*S* ← shapelet on the root node of **C**
5	*split_point* ← split point on the root of **C**
6	**If** SubsequenceDistanceEarlyAbandon (*T*, *S*) < *split_point*
7	Predict (left substree of **C**, *T*)
8	**Else**
9	Predict (right substree of **C**, *T*)
10	**EndIf**
11	**EndIf**

5. EXPERIMENTAL EVALUATION

We begin by discussing our experimental philosophy. We have designed and conducted all experiments such that they are easily reproducible. With this in mind, we have built a webpage [15] which contains all of the datasets and code used in this work, together with spreadsheets which contain the raw numbers displayed in all of the figures, and larger annotated figures showing the decision trees, etc. In addition, this webpage contains many additional experiments which we could not fit into this work; however, we note that this paper is completely self-contained.

5.1 Performance Comparison

We test the scalability of our shapelet finding algorithm on the Synthetic Lightning EMP Classification [6], which, with a 2,000/18,000 train/test split, is the largest class-labeled time series dataset we are aware of. It also has the highest dimensionality, with each time series object being 2,000 data points long. Using four different search algorithms, we started by finding the shapelet in a subset of just ten time series, and then iteratively doubled the size of the data subset until the time for brute force made the experiments untenable. Figure 11 shows the results.

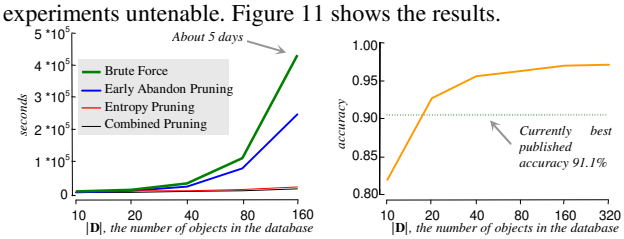

Figure 11: The time required to find the best shapelet (*left*) and the hold-out accuracy (*right*), for increasing large databases sizes

The results show that brute force search quickly becomes untenable, requiring about five days for just 160 objects. Early abandoning helps reduce this by a factor of two, and entropy based pruning helps reduce this by over two orders of magnitude. Both ideas combined almost linearly to produce three orders of magnitude speedup.

For each size data subset we considered, we also built a decision tree (which can be seen at [15]) and tested the accuracy on the 18,000 holdout data. When only 10 or 20 objects (out of the original 2,000) are examined, the decision tree is slightly worse than the best known result on this dataset (the one-nearest neighbor Euclidean distance), but after examining just 2% of the training data, it is significantly more accurate.

5.2 Projectile Points (Arrowheads)

Projectile point (arrowhead) classification is an important topic in anthropology (see [15] where we have an extensive review of the literature). Projectile points can be divided into different classes based on the location they are found, the group that created them, and the date they were in use, etc. In Figure 12, we show some samples of the projectile points used in our experiments.

Figure 12: Examples of the three classes of projectile points in our dataset. The testing dataset includes some broken points, and some drawings taken from anthropologist's field notes

We convert the shapes of the projectile points to a time series using the angle-based method [8]. We then randomly created a 36/175 training/test split. The result is shown in Figure 13.

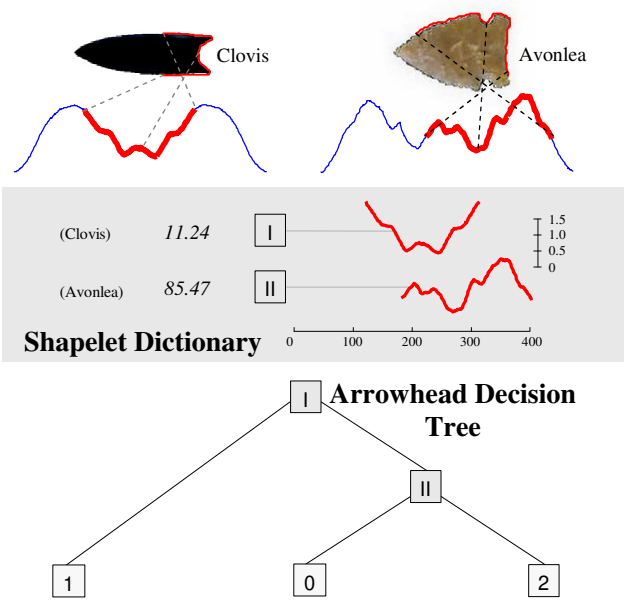

Figure 13: (*top*) The dictionary of shapelets, together with the thresholds d_{th}. (*bottom*) The decision tree for the 3-class projectile points problem

As shown in Figure 13 and confirmed by physical anthropologists Dr. Sang-Hee Lee and Taryn Rampley of UCR, the Clovis projectile points can be distinguished from the others by an un-notched hafting area near the bottom connected by a deep concave bottom end. After distinguishing the Clovis projectile points, the Avonlea points are differentiated from the mixed class by a small notched hafting area connected by a shallow concave bottom end.

The shapelet decision tree classifier achieves an accuracy of 80.0%, whereas the accuracy of *rotation invariant* one-nearest-neighbor classifier is 68.0%. Beyond the advantage of greater accuracy, the shapelet decision tree classifier produces the classification result 3×10^3 times faster than the *rotation invariant* one-nearest-neighbor classifier and it is more robust in dealing with the pervasive broken projectile points in most collections.

5.3 Mining Historical Documents

In this section we consider the utility of shapelets for an ongoing project in mining and annotating historical documents. Coats of arms or heraldic shields were originally symbols used to identify

individuals or groups on the battlefield. Since the beginning of the Middle Ages, thousands of annotated catalogues of these shields have been created, and in recent years hundreds of them have been digitized [1][9]. Naturally, most efforts to automatically extract and annotate these volumes concentrate on the colors and patterns of the shields; however, there is also useful information contained in the shape. Consider for example Figure 14, which shows examples of different shapes commonly associated with various countries' heraldic traditions.

Figure 14: Examples of the three classes in our dataset. The shields were hand-drawn one to six centuries ago

Note that in most of these documents, the shields were drawn freehand and thus have natural variability in shape, in addition to containing affine transformation artifacts introduced during the digital scanning.

We convert the shapes of the shields to a time series using the angle-based method [8]. Because some shields may be augmented with ornamentation (i.e far left in Figure 14) or torn (i.e Figure 16) and thus may have radically different perimeter lengths, we do not normalize the time series lengths.

We randomly select 10 objects from each class as the training dataset, and leave the remaining 129 objects for testing. The resulting classifier is shown in Figure 15.

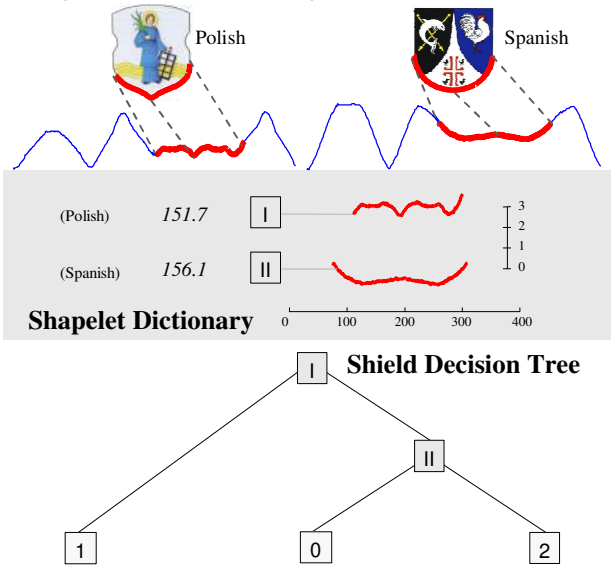

Figure 15: (*top*) The dictionary of shapelets, together with the thresholds d_{th}. (*bottom*) A decision tree for heraldic shields

Note that we can glean some information about this dataset by "brushing" the shapelet back onto the shields as in Figure 15 *top*. For example, both the Spanish and the French shields have right angle edges at the top of the shield, so the shapelet algorithm does *not* choose that common feature to discriminate between the classes. Instead, the unique semi-circular bottom of the Spanish crest is used in node II to discriminate it from the French examples. Nobody expects the Spanish Inquisition.

For our shapelet decision tree classifier, we achieve 89.9% accuracy; while for the *rotation invariant* one-nearest-neighbor Euclidean distance classifier the accuracy is only 82.9%. Beyond the differences in accuracy, there are two additional advantages of shapelets. First, the time to classify is approximately 3×10^4 times faster than for the *rotation invariant* one-nearest-neighbor Euclidean distance, although we could close that difference somewhat if we indexed the training data with a shape indexing algorithm [8]. Second, as shown in Figure 16, many images from historical manuscripts are torn or degraded. Note that the decision tree shown in Figure 15 can still correctly classify the shield of Charles II, even though a large fraction of it is missing.

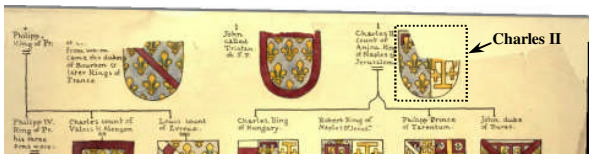

Figure 16: The top section of a page of the 1840 text, *A guide to the study of heraldry* [10]. Note some shields are torn

5.4 Understanding the Gun/NoGun Problem

The *Gun/NoGun* motion capture time series dataset is perhaps the most studied time series classification problem in the literature [4][14]. We take the standard train/test split for this dataset and use it to learn the decision tree shown in Figure 17.

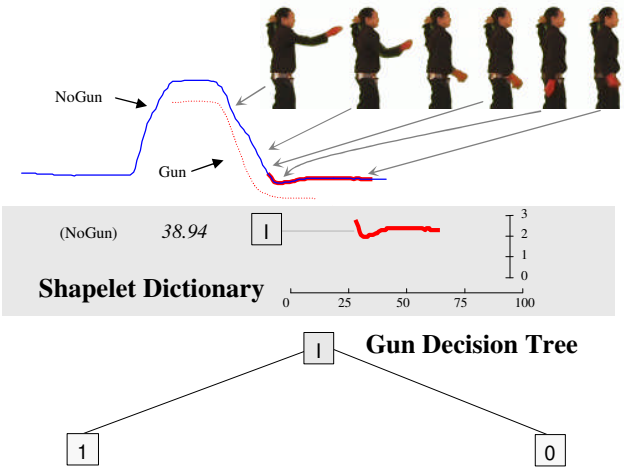

Figure 17: (*top*) The dictionary of shapelets, with the thresholds d_{th}. (*bottom*) The decision tree for the Gun/NoGun problem

The holdout accuracy for the decision tree is 93.3%, beating the one-nearest-neighbor Euclidean distance classifier, whose accuracy is 91.3%, and unconstrained or constrained DTW [4][14], with accuracies of 90.7% and 91.3%, respectively. More significantly, the time to classify using the decision tree is about four times faster than the one-nearest-neighbor Euclidean distance classifier. This is significant, since surveillance is a domain where classification speed can matter.

Moreover, by "brushing" the shapelet back onto the original video, we are able to gain some understanding of the differences between the two classes. In Figure 17, we can see that the NoGun class has a "dip" where the actor put her hand down by her side, and inertia carries her hand a little too far and she is forced to correct for it (a phenomenon known as "overshoot"). In contrast, when the actor has the gun, she returns her hand to her side more carefully, feeling for the gun holster, and no dip is seen.

5.5 Wheat Spectrography

This dataset consists of 775 spectrographs of wheat samples grown in Canada between 1998 and 2005. The data is made up of several different types of wheat, including *Soft White Spring*, *Canada Western Red Spring*, *Canada Western Red Winter*, etc. However, the class label given for this problem is the *year* in which the wheat was grown. This makes the classification problem very difficult, as some of the similarities/dissimilarities between objects can be attributed to the year grown, but some can be attributed to the wheat type, which we do not know. In Figure 18 we plot one example from each class; as the reader can see, the differences between classes are *very* subtle.

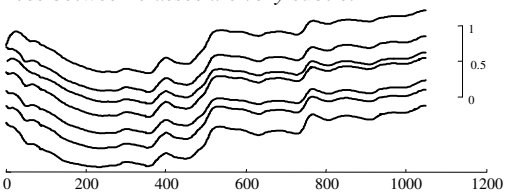

Figure 18 : One sample from each of the seven classes in the wheat problem. The objects are separated in the y-axis for visual clarity, as they all have approximately the same mean

We created a 49/726 train/test split, ensuring that the training set has seven objects from each class, and then tested the classification accuracy of the one-nearest-neighbor Euclidean distance classifier, which we find to be 44.1% (Dynamic Time Warping does not outperform Euclidean distance here). We then created a decision tree for the data, using the algorithm introduced in Section 4. The output is shown in Figure 19.

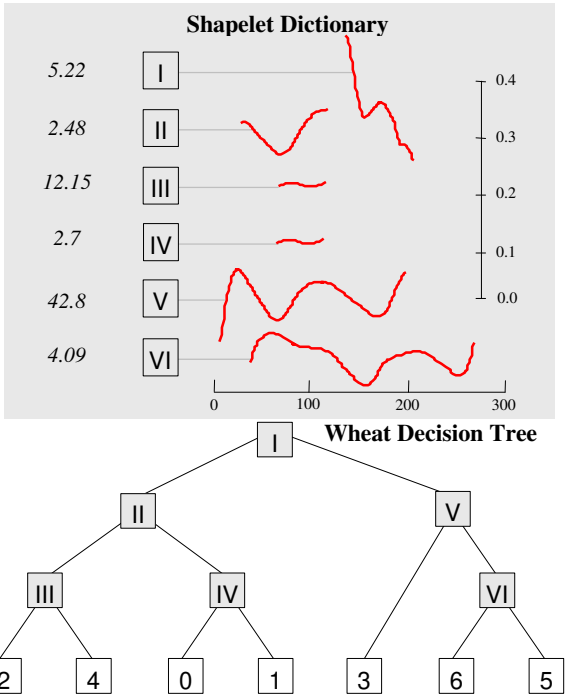

Figure 19: (*top*) The dictionary of shapelets, together with the thresholds d_{th}. (*bottom*) The decision tree for the wheat spectrography problem

The accuracy of the decision tree is 72.6%, which is significantly better than the 44.1% achieved by the nearest neighbor method.

6. CONCLUSIONS AND FUTURE WORK

We have introduced a new primitive for time series and shape mining, *time series shapelets*. We have shown with extensive experiments that we can find the shapelets efficiently, and that they can provide accurate, interpretable and fast classification decisions in a wide variety of domains. Ongoing and future work includes extensions to the multivariate case and detailed case studies in the domains of anthropology and MOCAP analyses.

Acknowledgements: We thank Ralf Hartemink for help with the heraldic shields and Dr. Sang-Hee Lee and Taryn Rampley for their help with the projectile point dataset. This work was funded by NSF 0803410 and 0808770.

7. REFERENCES

[1] Anon. 1525. Founders' and benefectors' book of Tewkesbury Abbey, in Latin England. Online version www.bodley.ox.ac.uk/dept/scwmss/wmss/medieval/mss/top/glouc/d/002.htm

[2] Breiman, L., Friedman, J., Olshen, R. A., and Stone, C. J. 1984. Classification and regression trees. Wadsworth.

[3] Chiu, B., Keogh, E., and Lonardi, S. 2003. Probabilistic Discovery of Time Series Motifs. In Proc of the 9th ACM SIGKDD. 493–498.

[4] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. 2008. Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. In Proc of the 34th VLDB. 1542–1552.

[5] Geurts, P. 2001. Pattern Extraction for Time Series Classification. In Proc of the 5th PKDD, 115–127.

[6] Jeffery, C. 2005. http://public.lanl.gov/eads/datasets/emp/index.html

[7] Keogh, E. and Kasetty, S. 2002. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In Proc' of the 8th ACM SIGKDD. 102–111.

[8] Keogh, E., Wei, L., Xi, X., Lee, S., and Vlachos, M. 2006. LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures. In the Proc of 32th VLDB. 882–893.

[9] Koschorreck,W. and Werner, W., editors. 1981. Facsimile edition with commentary: Kommentar zum Faksimile des Codex Manesse: Die grosse Heidelberger Liederhandschrift.

[10] Montagu, J.A. 1840. A guide to the study of heraldry. Publisher: London : W. Pickering. Online version www.archive.org/details/guidetostudyofhe00montuoft

[11] Rodríguez, J.J. and Alonso, C.J. 2004. Interval and dynamic time warping-based decision trees. In Proc of the 2004 ACM Symposium on Applied Computing, 548–552.

[12] Salzberg, S.L. 1997. On comparing classifiers: Pitfalls to avoid and a recommended approach. Data Mining and Knowledge Discovery, 1, 317–328, 1997.

[13] Wilcoxon, F. 1945. Individual Comparisons by Ranking Methods. Biometrics, 1, 80–83.

[14] Xi, X., Keogh, E., Shelton, C., Wei, L., and Ratanamahatana, C. A. 2006. Fast Time Series Classification Using Numerosity Reduction. In the Proc of the 23rd ICML. 1033–1040.

[15] Ye, L. 2009. The Time Series Shapelet Webpage. *www.cs.ucr.edu/~lexiangy/shapelet.html*