Data Min Knowl Disc (2011) 22:149–182
DOI 10.1007/s10618-010-0179-5

Time series shapelets: a novel technique that allows accurate, interpretable and fast classification

Lexiang Ye · Eamonn Keogh

Received: 21 May 2009 / Accepted: 11 May 2010 / Published online: 18 June 2010

Abstract Classification of time series has been attracting great interest over the past decade. While dozens of techniques have been introduced, recent empirical evidence has strongly suggested that the simple nearest neighbor algorithm is very difficult to beat for most time series problems, especially for large-scale datasets. 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 high 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 and to make the classification result more explainable, which global characteristics of the nearest neighbor cannot provide. 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. We can use the distance to the shapelet, rather than the distance to the nearest neighbor to classify objects. As we shall show with extensive empirical evaluations in diverse domains, classification algorithms based on the time series shapelet primitives can be interpretable, more accurate, and significantly faster than state-of-the-art classifiers.

Keywords Time series · Data mining · Classification · Decision tree

Responsible editor: Bart Goethals.

L. Ye (✉) · E. Keogh
Department of Computer Science and Engineering, University of California,
Riverside, CA 92521, USA
e-mail: lexiangy@cs.ucr.edu

E. Keogh
e-mail: eamonn@cs.ucr.edu

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 (Ding et al. 2008; Salzberg 1997; Xi et al. 2006). 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, since during the classification, the algorithm needs the object to be compared with each object in the training set, leading to quadratic time and linear space complexity. Another drawback is the fact that it tells us very limited information about *why* a particular object is assigned to a particular class.

In this work we present a novel time series data mining primitive called *time series shapelets* (Yamada et al. 2003). Informally, shapelets are time series subsequences which are in some sense maximally representative of a class. While we believe shape-lets can have many uses in data mining, one of their obvious implications is their ability to mitigate the two weaknesses of the nearest neighbor algorithm noted above.

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*.

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 Fig. 1, their differences in the global shapes are very subtle. Furthermore, it is very common for leaves to have distortions or "occlusions" due to insect damage, which 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 Fig. 2.

Such representations have been successfully used for the classification, clustering and outlier detection of shapes in recent years (Keogh et al. 2006). However, here we find that using the nearest neighbor classifier with either the (rotation invariant)

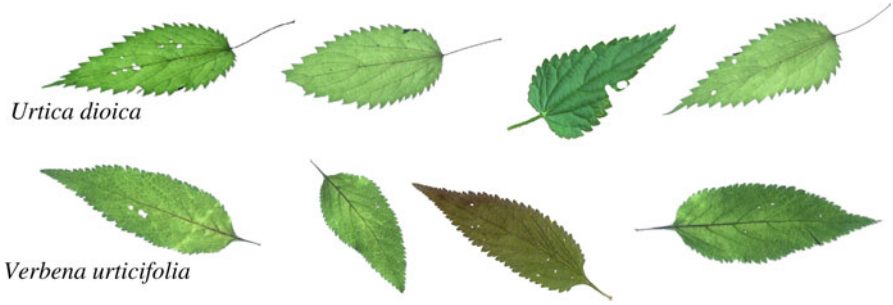

Fig. 1 Samples of leaves from two species. Note that several leaves have insect-bite damage

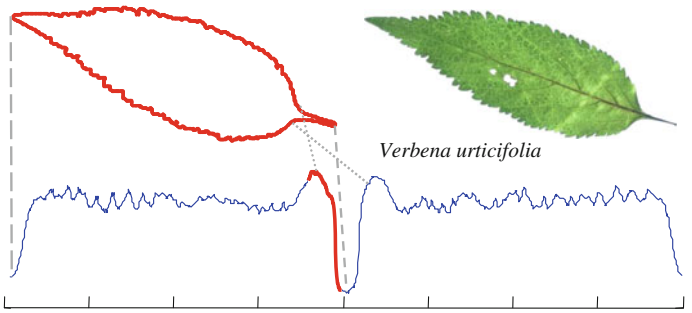

Fig. 2 A shape can be converted into a one dimensional "time series" representation. The reason for the highlighted section of the time series will be made apparent shortly

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, 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 Fig. 3, we see the shapelet discovered by searching the small dataset shown in Fig. 1.

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 wider 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 Fig. 4.

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

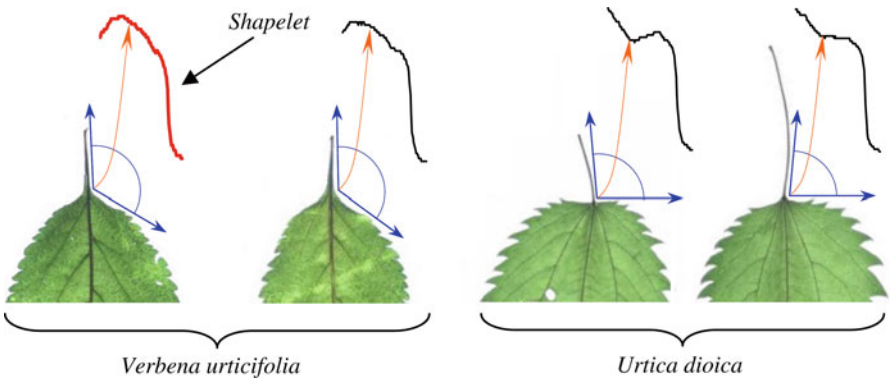

Fig. 3 Here, the shapelet hinted at in Fig. 2 (in both cases shown with a *bold line*) is the subsequence that best discriminates between the two classes

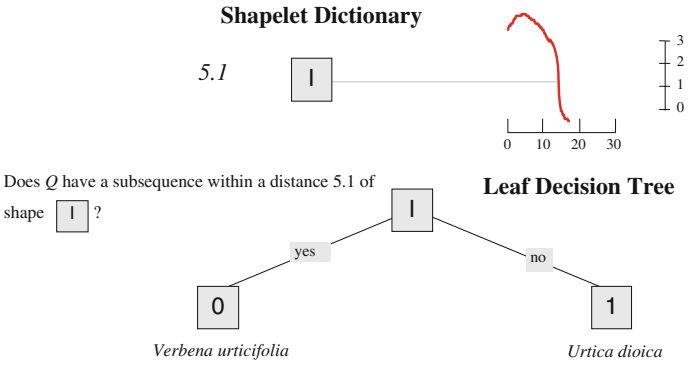

Fig. 4 A decision-tree classifier for the leaf problem. The object to be classified has all of its subsequences compared to the shapelet, and if any subsequence is less than (the empirically determined value of) 5.1, it is classified as *Verbena urticifolia*

Shapelets can provide interpretable results, which may help domain practitioners better understand their data. For example, in Fig. 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 (Ding et al. 2008; Keogh and Kasetty 2002).

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 (Ding et al. 2008). 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 (Keogh et al. 2006), the time complexity is on the order of $O(km^3)$, where k is the number of reference objects in the training set.[1] On real-world problems the speed difference can be greater than three orders of magnitude.

Shapelets save considerable space compared to the state-of-the-art approaches. The space required by the shapelet is $O(nl)$, where l is the length of the shapelet and n is related to the number of classes of the training set, since shapelets indicate the features of the objects in the entire class. For the example-based approach, the space requirement is $O(km)$, where k is the number of reference objects and m is the length of reference objects in the training set. Therefore, n is much smaller than k.

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 Sect. 5, we can provide extensive empirical evidence for all of these claims

[1] There are techniques to mitigate the cubic complexity of rotation invariant DTW, but unlike shapelets, the time *is* dependent on the size of the training dataset.

on a vast array of problems in domains as diverse as anthropology, human motion analysis, spectrography, and historical manuscript mining.

The rest of this paper is organized as follows. In Sect. 2, we review related work and background material. Section 3 introduces concrete algorithms to efficiently find shapelets. Sect. 4 discusses the method of classification using shapelets as a tool. In Sect. 5, we perform a comprehensive set of experiments on various problems of different domains. Finally, in Sect. 6 we conclude our work and suggest directions for future work.

2 Related work and background

2.1 Related work

2.1.1 Classification for interpretability

Classification has attracted significant attention over the past decade. Among all the time series classification algorithms, the simple nearest neighbor algorithm has been shown to be the most robust and accurate method by experiments on a great variety of datasets (Xi et al. 2006). Though there are different representation methods, such as Discrete Fourier Transformation (DFT) (Faloutsos et al. 1994) and Symbolic Aggregate approXimation (SAX) (Lin et al. 2007), and different similarity measures, such as Euclidean Distance (Faloutsos et al. 1994) and Dynamic Time Warping (DTW) (Berndt and Clifford 1994; Keogh and Ratanamahatana 2005), a recent large scale set of experiments indicates that the difference between different methods and measures diminishes or disappears as the size of the dataset becomes larger (Ding et al. 2008).

In many real-world applications, classification tasks should focus not only on accuracy, but also on *interpretability*. Rule-based methods, like the decision tree method, are generally more interpretable than instance-based methods. However, it is difficult to find a general method to extract rules from time series datasets.

Previous work uses either global or predefined features to construct the model. In (Kadous 1999), the method classifies and interprets the data via two ways: global feature calculation and event extraction. The parameterized events are clustered in the parameter space and synthetic event attributes are extracted from the clusters. The global features and event attributes combine together to form the classifier. The drawback of this method is that all of the events must be defined by a user on a case-by-case basis. It becomes particularly difficult to determine which events should be used in the clusters when the dataset becomes large, even if you already know the rule of classifying examples from different classes.

The work in (Yamada et al. 2003) also uses the decision tree classifier to explore comprehensibility. However, instead of extracting the common feature from the time series in one class, their method compares test examples with one entire time series at each internal node of the decision tree. The search space of extracting one entire time series is several orders of magnitude smaller than extracting a shapelet. In the meantime, viewing the time series in the global view also significantly reduces both

the interpretability and efficiency of the classification, especially when the application contains high-dimensional, large-scale, or noisy datasets.

The closest work is that of (Geurts 2001). 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; even then the author only considers 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% of the calculations (c.f. Sect. 5). Our work may also be seen as a form of a *supervised* motif discovery algorithm (Chiu et al. 2003), in that we are looking for repeated patterns (as in motif discovery). However, we are trying to find "motifs" which exist mostly in one class and not the other. So motif discovery can at most be seen as a subroutine in our work.

2.1.2 Similar problems in other domains

In string processing, there is a somewhat similar problem called the "distinguishing substring selection" (DSSS) (Gramm et al. 2003). The problem is defined as follows: Two sets of strings, one "good" and the other "bad", produce a substring, which is "close" to the strings in the "bad" set, and "away" from those in the "good" set. This problem is simpler than the shapelet discovery problem. First, it is only defined for a finite alphabet of the strings. In most instances in the literature only a binary alphabet is considered. Second, it uses the Hamming metric as the distance measure. Moreover, only two-class problems are considered. Most methods initiate the candidate-distinguishing substring with the inverse of the first string in the "good set," and adjust the candidate substring in all possible ways to move it away from some string in the "good set" if the candidate is too close to the string.

2.2 Notation

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

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 and spaced at equal time intervals. We are typically more interested in the *local* shape properties of a time series rather than the *global* statistical properties, such as (global) mean, standard deviation, skewness, etc. 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 $l \leq m$ of contiguous positions from T, that is, $S = t_p, \ldots, t_{p+l-1}$, for $1 \leq p \leq m - l + 1$.

Table 1 Symbol table

Symbol	Explanation
T, R	Time series
S	Subsequence
$\mathbf{S}_T^l$	Set of all subsequences of length l from time series T
m, $\|T\|$	Length of time series
$\overline{m}$	Average length of time series in a dataset
l, $\|S\|$	Length of subsequence
d	Distance measurement
$\mathbf{D}$	Time series dataset
A, B	Class label
H	Entropy
$\hat{H}$	Weighted average entropy
sp	Split strategy
k	Number of time series objects in the dataset
$\mathbf{C}$	Classifier
$S_{(k)}$	The kth data point in subsequence S

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 $\mathbf{S}_T^l$, $\mathbf{S}_T^l = \{S_p^l \text{ of } T, \text{ for } 1 \leq p \leq m - l + 1\}$.

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 of equal 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 $S^* \in \mathbf{S}_T^{|S|}$.

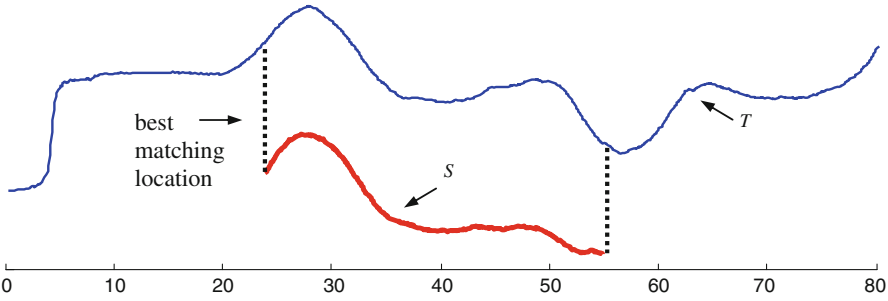

Fig. 5 Illustration of the best matching location in time series T for subsequence S

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

As we shall explain in Sect. 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 (Breiman et al. 1984). The reader may recall the original definition of entropy which we review here:

Definition 6 *Entropy*. A time series dataset $\mathbf{D}$ consists of data from two classes, labeled A and B (for example A = "stinging nettles" and B = "false nettles"). Given that the proportion of time series objects in class A is $p(A)$ and the proportion of time series objects in class B is $p(B)$, the entropy of $\mathbf{D}$ is:

$$H(\mathbf{D}) = -p(A)\log(p(A)) - p(B)\log(p(B)).$$

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 $\hat{H}(\mathbf{D}) = f(\mathbf{D}_1)H(\mathbf{D}_1) + f(\mathbf{D}_2)H(\mathbf{D}_2)$. 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 $H(\mathbf{D})$ and $\hat{H}(\mathbf{D})$. So the information gain for this splitting rule is

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

As we noted 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 Sect. 3, 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 $\mathbf{D}_2$, $SubsequenceDist(T_{2,i}, S) \geq d_{th}$. An *Optimal Split Point* is a distance threshold that

$$Gain(S, d_{OSP(\mathbf{D},S)}) \geq Gain(S, d^*_{th}),$$

for any other distance threshold d^*_{th}.

Using shapelets in a classifier requires two factors: a shapelet and the corresponding optimal split point. As a concrete example, in Fig. 4 the shapelet is shown in red in the shapelet dictionary along with its optimal split point of 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,

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

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 is 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 $MAXLEN \leq \min(m_i)$, where m_i is the length of the time series T_i from the dataset, $1 \leq i \leq k$. 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 (Roverso 2000; Rodríguez and Alonso 2004) has 200 instances of length 275. If we set $MINLEN = 3$ and $MAXLEN = 275$, 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 Sect. 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.

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 3–7). For each shapelet candidate, the algorithm calls the function CheckCandidate() to obtain the information gain achieved if that candidate is used to separate the data (line 4). As illustrated in Fig. 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

Table 2 Brute force algorithm for finding shapelet

```
Function FindingShapeletBF (dataset D, MAXLEN, MINLEN, STEPSIZE)
1   candidates ← GenerateCandidates (D, MAXLEN, MINLEN, STEPSIZE)
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
```

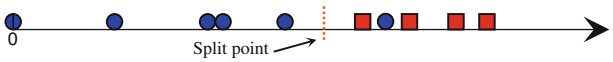

Fig. 6 The CheckCandidate() function at the heart of the brute force search algorithm can be regarded as testing to see how mapping all of the time series objects on the number line, based on their *SubsequenceDist*(T, S), separates the two classes

mapping produces two well-separated "pure" groups. In this regard the example in Fig. 6 is very good, but clearly not perfect.

If the information gain is higher than *bsf_gain*, the algorithm updates *bsf_gain* and the corresponding best shapelet candidate *bsf_shapelet* (lines 5–8). 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 Tables 3 and 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).

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 3 Generate all the candidates from a time series dataset

	Function `GenerateCandidates` (`dataset` **D**, *MAXLEN, MINLEN, STEPSIZE*)
1	*pool* ← Ø
2	*l* ← *MAXLEN*
3	**While** *l* ≥ *MINLEN*
4	**For** *T* **in D**
5	*pool* ← *pool* ∪ S_T^l
6	**EndFor**
7	*l* ← *l* –*STEPSIZE*
8	**EndWhile**
9	**Return** *pool*

Table 4 Checking the utility of a single candidate

	Function `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*)

Table 5 Information gain of distance histogram optimal split

```
Function CalculateInformationGain(distance histogram obj_hist)
1   split_dist ← OptimalSplitPoint(obj_hist)
2   D1 ← Ø, D2 ← Ø
3   For d in obj_hist
4       If d.dist<split_dist
5           D1 ← D1∪ d.objects
6       Else
7           D2 ← D2∪ d.objects
8       EndIf
9   EndFor
10  Return H(D)-Ĥ(D)
```

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 1–4. After that, the algorithm returns the utility of that candidate by calling CalculateInformationGain() (line 6).

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 4–7). Finally, it calculates the information gain (cf. Definitions 6 and 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 Fig. 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 Sect. 2, the size of the candidate set is $O(\overline{m}^2 k)$. Checking the utility of one candidate requires its comparison with $O(\overline{m}k)$ subsequences and each comparison (Euclidean distance) takes $O(\overline{m})$ on average. Hence, the overall time complexity of the algorithm is $O(\overline{m}^4 k^2)$, which makes its usage for 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*, which is very simple yet has been shown to be extremely effective for similar types of problems (Keogh et al. 2006).

While the premise is simple, for clarity we illustrate the idea in Fig. 7 and provide the pseudo code in Table 6.

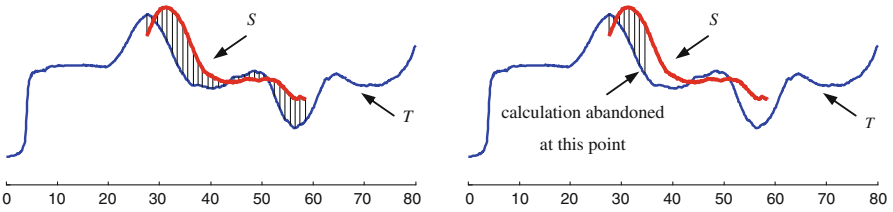

Fig. 7 (*left*) Illustration of complete Euclidean distance. (*right*) Illustration of Euclidean distance early abandon

Table 6 Early abandon the non-minimum distance

```
Function SubsequenceDistanceEarlyAbandon(T, S)
1   min_dist ← ∞
2   For S_i in S_T^|S|
3       stop ← False
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 7–9). 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 average running time required by a large factor.

3.3 Admissible entropy pruning

Our definition of the shapelet requires some measure of how well the distances to a time series subsequence can split the data into two "purer" subsets. The readers 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 (Wilcoxon 1945). 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 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; we can be secure in the knowledge that it cannot be a better candidate than the current best so far.

In order to help the readers understand the idea of pruning with an upper bound of the information gain, we consider a simple example. Suppose as shown in Fig. 8, ten time series objects are arranged in a one-dimensional representation by measuring their distances 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:

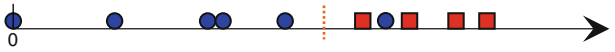

Fig. 8 Distance arrangement of the time series objects in one-dimensional representation of best-so-far information gain. The positions of the objects represent their distances to the candidate

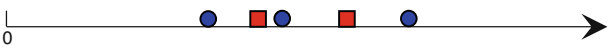

Fig. 9 The arrangement of first five distances from the time series objects to the candidate

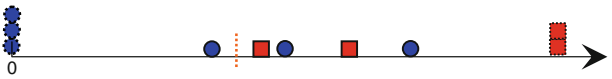

Fig. 10 One optimistic prediction of distance distribution based on distances that have already been calculated in Fig. 9. The dashed objects are in the optimistically assumed placements

[-(6/10)log(6/10)-(4/10)log(4/10)] - [(5/10)[-(5/5)log(5/5)-(0/5)log(0/5)]+(5/10)[-(4/5)log(4/5)-(1/5)log(1/5)]]=0.4228

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 Fig. 9.

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 result 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. In particular, given the placement of the first five objects, one of these two scenarios *must* maximize the probability of best separating the objects from two classes and thus maximize the information gain. Figure 10 shows the former scenario applied to the example shown in Fig. 9.

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)-(0/4)log(0/4)]+(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. In Sect. 5, we will empirically evaluate the time we save.

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 Figs. 8, 9 and 10), and the remaining time series objects in class *A* and class *B*. It returns True if we can prune the candidate as the answer. The algorithm begins by finding the two ends of the histogram (discussed in Sect. 2). For simplicity, we set the distance values at the two ends at 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 information gain (lines 5 and 10), it is still possible that

Table 7 Information gain upper bound pruning

```
Function 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 actual information gain of the candidate can beat the best so far. Thus, we cannot prune the candidate and we should continue with the test (lines 6 and 11). Otherwise, if the upper bound of the actual information gain is lower than the best so far, we are saved from all of the remaining distance calculations with this candidate (line 13).

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 $a_1, b_1, a_2, b_3, \ldots, a_n, b_n$. This ordering lets the algorithm know very early in the search if a candidate cannot beat the best so far.

3.4 Techniques for breaking ties

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.

3.4.1 Longest candidate

The longest candidate always contains the entire distinguishing feature that is in one class and absent from the other class. However, it is possible that the longest candidate

might also contain some irrelevant information or noise near the distinguishing feature subsequence; this is likely to reduce the accuracy in some applications.

3.4.2 Shortest candidate

In contrast, favoring the shortest candidate can avoid noise in the shapelet. The shortest candidate is useful when there are multiple, discontinuous features in one class. To enumerate each of these shapelets, each time after the algorithm finds a shapelet we replace the feature subsequences in all of the time series objects with random walk subsequences of the same length and rerun the algorithm. By running the algorithm multiple times like this, the method will return short, discontinuous shapelets.

3.4.3 Maximum separation

Using the maximum separation to break the tie follows the same basic idea of SVMs. The algorithm finds the shapelet that maximizes the distance between the two classes. We calculate the distance between two classes by: first, the mean distances from the time series objects in each individual class to the shapelet, then return the difference between the mean distances as the distance between two classes.

Based on comprehensive experiments, the best accuracy is most often achieved when breaking the tie using the shapelet that has the maximum margin, which is intuitive, since this method maximizes the difference between classes.

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 concreteness we will focus just on classification in this work.

Classifying by a shapelet and its corresponding split point makes a binary decision as to whether a time series object belongs to a certain class or not. Obviously, this method 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 (Breiman et al. 1984).

At each step of the decision tree induction, we determine the shapelet and its corresponding split point over the training subset considered in that step. (A similar idea is illustrated in (Geurts 2001).)

After the learning procedure finishes, we can assess the performance of the shapelet classifier by calculating the accuracy of 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 completeness the algorithm is described in Table 8.

The technique to predict the class label of each testing object is described in Table 9. For each internal node of the decision tree we have the information of a single shapelet classifier, the left subtree, and the right subtree. The leaf nodes contain information about a predicted class label. Starting from the root of a shapelet classifier, we calculate the distance from the testing object T to the shapelet in that node. If the distance

Table 8 Calculating the accuracy on the shapelet classifier

```
Function CalculateAccuracy(shapelet classifier C, dataset D_t)
1  For each T in D_t
2      predict_class_label ← Predict(C, T)
3      If predict_class_label is same as actual class label
4          correct ← correct + 1
5      EndIf
6  EndFor
7  Return correct/|D_t|
```

Table 9 Predict the class label of a testing object

```
Function Predict (shapelet classifier C, testing time series T)
1  If C is a 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 node of C
6      If SubsequenceDistanceEarlyAbandon (T, S)<split_point
7          Predict(left subtree of C, T)
8      Else
9          Predict(right subtree of C, T)
10     EndIf
11 EndIf
```

is smaller than the split point, we recursively use the left subtree (lines 6 and 7) or otherwise use the right subtree (lines 8 and 9). This procedure continues until we reach a leaf node and return the predicted class label (lines 1 and 2).

5 Experimental evaluation

In this section, we consider several examples of applications of the shapelet classifier. First we describe the experiment strategies and philosophy.

5.1 Experiment setup

5.1.1 Experiment methodology

We emphasize the importance of statistical validity in the design of the experiments. Our case studies include several real-world datasets from various domains such as shape, spectrography, motion capture, as well as synthetic datasets. For all of these datasets except the Mallat and Coffee datasets, we compare the shapelet classifier with the (rotation invariant where appropriate) one-nearest neighbor classifier. For the Mal-

lat dataset, the shapelet classifier is compared with the decision tree classifier presented in a previous work (Jeong et al. 2006). For the Coffee dataset, the shapelet classifier is compared with linear discriminant analysis (Martinez and Kak 2001) utilized in a previous work (Briandet et al. 1996). Note that recent extensive empirical evaluations have shown that the one-nearest neighbor classifier is (by a significant margin) the best general time series classifier (Xi et al. 2006).

We use separate subsets for training and testing. The performance comparisons are averaged over ten randomly split runs. For all of the case studies where there is no fixed training/testing split from other literatures which we can compare our result to, the reported shapelets are from a randomly selected training/testing split.

In the shapelet classifier, there are four parameters: *MAXLEN*, *MINLEN*, *STEPSIZE,* and *SEPERATION METHOD*. However, there is essentially zero effort in tuning the parameters. For *MAXLEN*, we always set the longest possible length to the length of the shortest time series in the dataset. For *MINLEN*, we hardcoded the shortest possible length to three since three is the minimum meaningful length. The only special case in our experiments is in the lightning dataset, which is the largest dataset considered. Here, based on a one-time visual inspection of the data, *MINLEN* is set at 1/20 of the entire length of time series and *MAXLEN* at 1/10 of the entire length of time series. The reason for this exception is to reduce the size of the search space, thus making the experiment tenable. The only parameter which really needs to be decided is *SEPERATION METHOD*, which we have discussed in Sect. 3.4. This parameter can be decided using domain knowledge of the application. In the case that the user only cares about the classification accuracy, we can try all three and pick the one with highest training accuracy. In our experiments, we tried each of the three separation methods and picked the one with highest accuracy on the testing data. The slight difference in accuracy using different separation methods are reported in (Ye 2009). For tractability, we have the parameter of *STEPSIZE*, which defines the possible lengths of shapelet candidates. We increase the *STEPSIZE* parameter in large datasets: 50 for the lightning dataset, 10 for the wheat dataset, and one for all of the other datasets. In our free code for shapelet discovery, we offer an interface that allows the user to set these parameters.

5.1.2 Performance metrics

The performance metrics are the running time, the accuracy, and (somewhat subjectively) the interpretability of the results. There are two aspects of the running time: classification running time and model learning time. We compare the classification runtime with the one-nearest neighbor classifier's and the model learning time with the original brute force algorithm's (cf. Table 2). The accuracy is compared against the (rotation invariant) one-nearest neighbor classifier if the dataset has not been studied by other researchers in the past. Otherwise, we compare the classification accuracy with previous works.

5.1.3 Reproducibility

We have designed and conducted all experiments such that they are easily reproducible. With this in mind, we have built a webpage (Ye 2009) which contains all of

the datasets and code used in this work, spreadsheets which contain the raw numbers displayed in all of the figures, larger annotated figures showing the decision trees, etc. However, we note that this work is completely self-contained.

5.2 Performance comparison

We test the scalability of our shapelet finding algorithm on the Synthetic Lightning EMP Classification (Jeffery 2005), which, with a 2,000/18,000 training/testing split, is the largest class-labeled time series dataset we are aware of. It also has the highest dimensionality, with each time series 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 objects. We then iteratively double the size of the data subset until the time for brute force made the experiments untenable. Figure 11 shows the results. Each computational time and classification accuracy is averaged over ten runs, with each run a subset is randomly selected from the training dataset.

The results show that the 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 one order of magnitude. Combining both ideas produces a two orders of magnitude speedup when the training set is the size of 160.

For different sizes of training datasets, we study how the various lengths of time series would affect the computational time. To keep the original properties and distributions of the data, data points in each of time series are resampled to the lengths of 250, 500, and 1000. The computational time of different sampled lengths of the same datasets is reduced by early abandon pruning. As illustrated in Fig. 12, the practical complexity is approximately quadratic on the length of time series, while the theoretical worst-case complexity is $O(m^4)$ on the length of time series.

For each size of the data subset we considered, we also built a decision tree (which can be seen at (Ye 2009)) and tested the accuracy on the 18,000 holdout data (Fig. 13). 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 (Lang et al. 2009) (using the one-nearest neighbor Euclidean distance), but after examining just 2% of the training data, it is significantly more accurate. The comparison to the results in (Lang et al. 2009) is fair, since we use exactly the same training/testing split as they

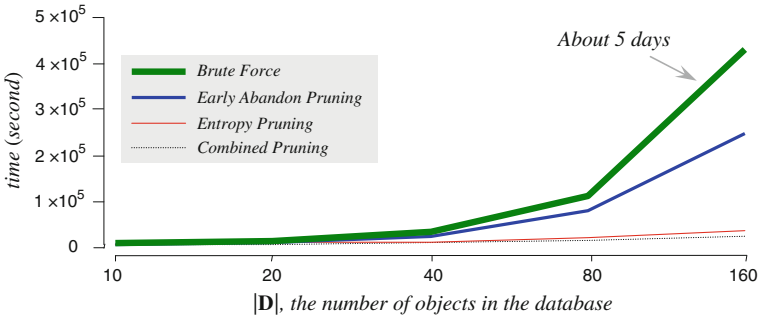

Fig. 11 The time required to find the best shapelet for increasing large dataset sizes

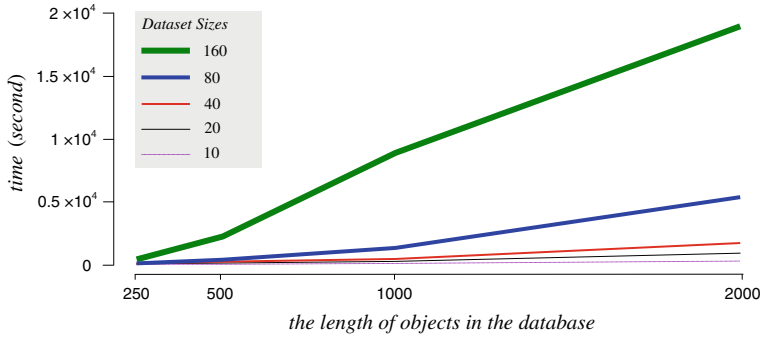

Fig. 12 The time required to find the best shapelet for increasing lengths of time series objects in the dataset, with various dataset sizes

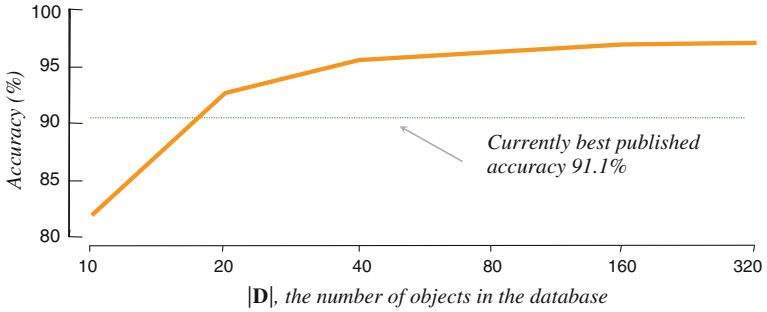

Fig. 13 The hold-out accuracy for increasing large dataset sizes

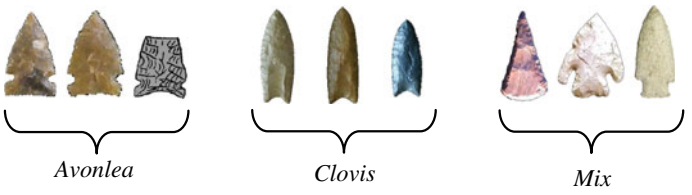

Fig. 14 Examples of the three classes of projectile points in our dataset. The testing dataset includes some broken points and some drawings taken from anthropologists' field notes

do. Note that their best result, which we so easily beat, is the best of eight diverse and highly optimized approaches they consider.

5.3 Projectile points (arrowheads)

Projectile point (arrowhead) classification is an important topic in anthropology (see (Ye 2009) where we have an extensive review of the literature). Projectile points can be divided into different classes based on the location in which they are found, the group that created them, the date they were in use, etc. In Fig. 14, we show some samples of the projectile points used in our experiments.

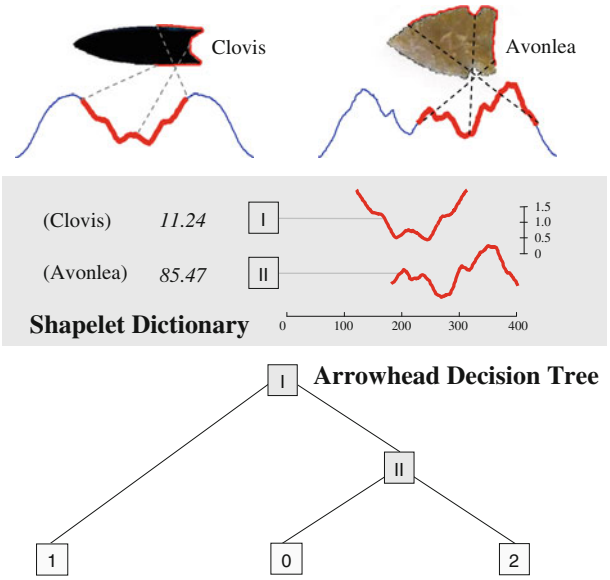

Fig. 15 (*top*) The dictionary of shapelets together with the thresholds d_{th}. The x-axis shows the position of the shapelets in the original time series. (*bottom*) The decision tree for the 3-class projectile points problem

We convert the shapes of the projectile points to a time series using the angle-based method (Keogh et al. 2006). We then randomly create a 36/175 training/test split. The result is shown in Fig. 15.

As shown in Fig. 15 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 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 classifier produces the classification result 3×10^3 times faster than the *rotation invariant* one-nearest neighbor classifier and is more robust in dealing with the pervasive broken projectile points in the collections.

5.4 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 (Anon 1525; Koschorreck and Werner 1981). 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 Fig. 16, which shows examples of different shapes commonly associated with various countries and heraldic traditions.

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. Because some shields may be augmented with ornamentation (i.e. far right in Fig. 16) or torn (i.e. Fig. 18) and thus may have radically different perimeter lengths, we do not normalize the time series lengths.

We randomly select ten objects from each class as the training dataset, and leave the remaining 129 objects for testing. The final classifier is shown in Fig. 17.

Note that we can glean some information about this dataset by "brushing" the shapelet back onto the shields as in Fig. 17 *top*. For example, both the Spanish and the

Fig. 16 Examples of the three classes in our dataset. The shields were hand-drawn one to six centuries ago

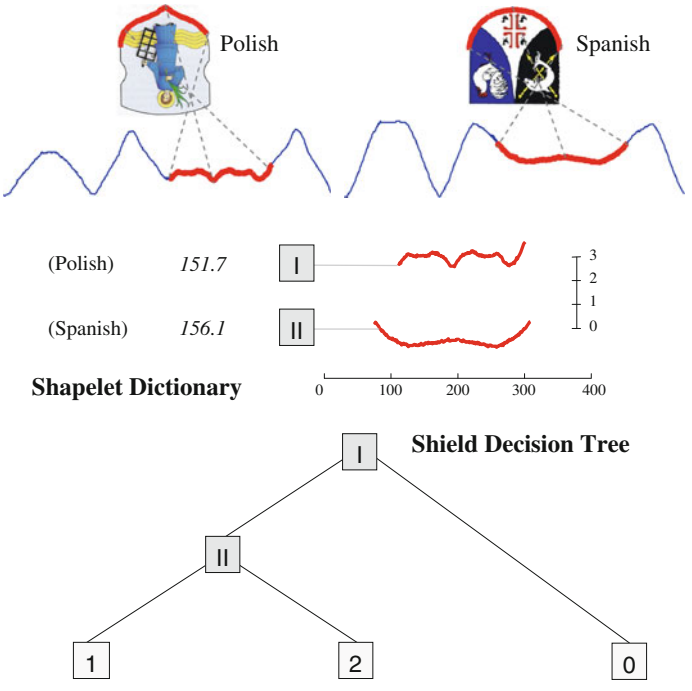

Fig. 17 (*top*) The dictionary of shapelets, together with the thresholds d_{th}. The x-axis shows the position of the shapelets in the original time series. (*bottom*) A decision tree for heraldic shields

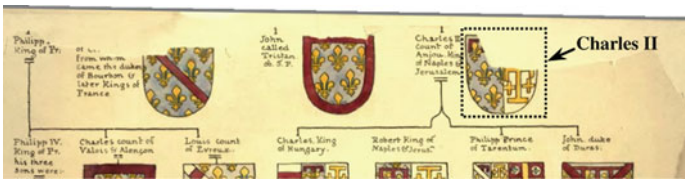

Fig. 18 The top section of a page of the 1840 text, *A guide to the study of heraldry* (Montagu 1840). Note that some shields are torn

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 end of the Spanish crest is used in node II to discriminate it from the French examples.

For our shapelet classifier, we achieve 89.9% accuracy using a maximum mean separation method to break ties 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 (Keogh et al. 2006). Second, as shown in Fig. 18, many images from historical manuscripts are torn or degraded. Note that the decision tree shown in Fig. 17 can still correctly classify the shield of Charles II, even though a large fraction of it is missing.

5.5 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 (Ding et al. 2008; Xi et al. 2006). The dataset consists of 100 instances from each class. In the *Gun* class, the actors have their hands by their sides, draw a gun from a hip-mounted holster, point it at a target for approximately one second, and then return the gun to the holster and their hands to their sides. In contrast, in the *NoGun* class, actors do the similar hands-down, point, hold, and return motion without the gun in their hands and therefore are pointing to a target using the index finger. Our classification problem is to distinguish these two very similar types of motions. The problem is made somewhat more complicated by the fact that there are two actors, who differ in height (by 12 inches) and "style".

We take the standard training/testing split for this dataset and use it to learn the decision tree shown in Fig. 19.

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 (Ding et al. 2008; Xi et al. 2006), 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 Fig. 19,

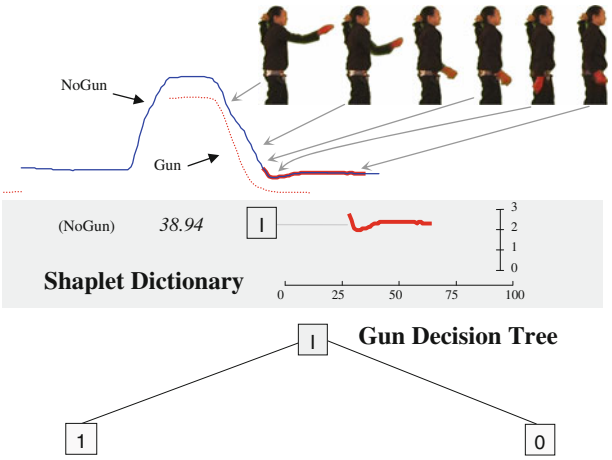

Fig. 19 (*top*) The dictionary of shapelets, with the thresholds d_{th}. The x-axis shows the position of the shapelet in the original time series. (*bottom*) The decision tree for the Gun/NoGun problem

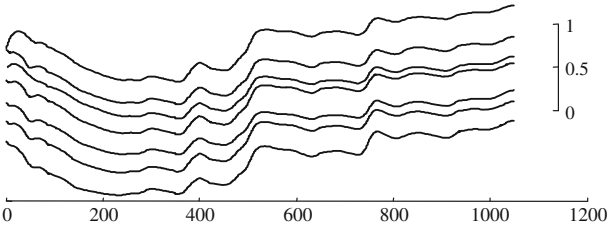

Fig. 20 One sample from each of the seven classes in the wheat problem. The objects are separated in the y-axis for visual clarity, as they all have approximately the same mean

we can see that the NoGun class has a "dip" where the actor puts 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.

5.6 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 Fig. 20 we plot one example from each class; as the reader can see, the differences between classes are *very* subtle.

We create a 49/726 training/testing split, ensuring that the training set has seven objects from each class, and then test the classification accuracy of the one-nearest neighbor Euclidean distance classifier, which we find to be 44.1% (Dynamic Time

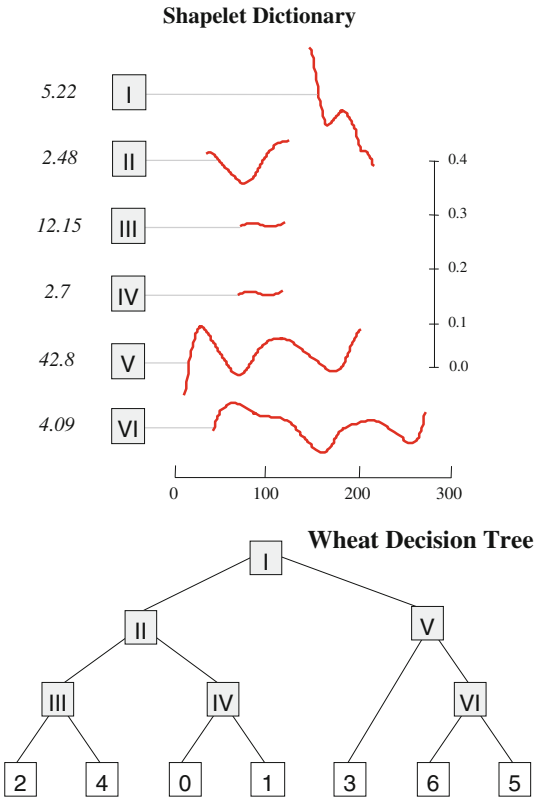

Fig. 21 (*top*) The dictionary of shapelets, together with the thresholds d_{th}. The x-axis shows the position of the shapelets in the original time series. (*bottom*) The decision tree for the wheat spectrography problem

Warping does not outperform Euclidean distance here). We then create a decision tree for the data, using the algorithm introduced in Sect. 4. The output is shown in Fig. 21.

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

5.7 Coffee spectrography

The two main species of coffee cultivated worldwide are *Arabica* (∼90%) and *Canephora* variant *Robusta* (∼9%) (Briandet et al. 1996). They are different in ingredients, flavor, cultivated environment, and commercial value. Arabica has a finer flavor, and thus is valued higher than Robusta; Robusta is less susceptible to disease and is cultivated as a substitute for Arabica. Robusta also contains about 40–50% more caffeine than Arabica [25].

In this experiment, 56 pure samples of freeze-dried coffee samples are analyzed under DRIFT (Wilson and Goodfellow 1994) analysis. The data is obtained from the work (Briandet et al. 1996). We split the data into 28 samples (14 Arabica and 14 Robusta) for training and 28 samples (15 Arabica and 13 Robusta) for testing. The resulting shapelet decision tree is shown in Fig. 22.

As stated in (Briandet et al. 1996), bands between the region 1550–1775 cm^{-1} represent the ingredient of caffeine. Since the spectrography data we experiment on is down-sampled, the corresponding region of bands 1550–1775 cm^{-1} is 187.7–247.3. The shapelet found by our algorithm is between 203–224, which reveals the major ingredient difference between the Arabica and the Robusta.

In the original paper, using the PCA method, the cross-validation accuracy is 98.8%. In our experiment, we are using much less training data, but achieving perfect accuracy.

5.8 Mallat

The Mallat dataset (Jeong et al. 2006) consists of eight classes of curves, one piecewise "normal" pattern, and its seven fault cases. One example in each class is presented in Fig. 23. For each of the eight classes, three hundred various replicated curves were generated. These curves were generated from shifting the original curve t time-units ($t = 5, 10, 15, 20, 25, 30$) and adding an Gaussian random noise $N(0, \sigma^2)$ to the entire time series, where $\sigma = 0.1$.

The original creators of this dataset use a CART decision tree which splits the data based on the features derived from wavelet coefficients. They were able to achieve an error rate of 2.25% by the 1/3 training, 2/3 testing split.

In our experiment, we only use 2/15 as training data and leave all the remaining as the testing data and achieve an error rate of 1.5%. In other words, we can use less than half of the data the original authors used and still decrease the error rate by a large margin.

The shapelet decision tree is shown in Fig. 24. Most of the shapelets in our decision can be easily interpreted. The first shapelet covers the first rectangular dip. All the data objects on the left have either no dip or a shallow one, while objects on the right of the tree have a deeper dip. The third shapelet on the decision tree is similar to the first one; it distinguishes the time series objects that are without dip from those with a shallow dip. The fourth shapelet shows that the most critical difference between the objects in class 5 and 6 is whether the second and the third peaks are present. A similar interpretation also exists in class 1 and 7, and the corresponding shapelet is shown as

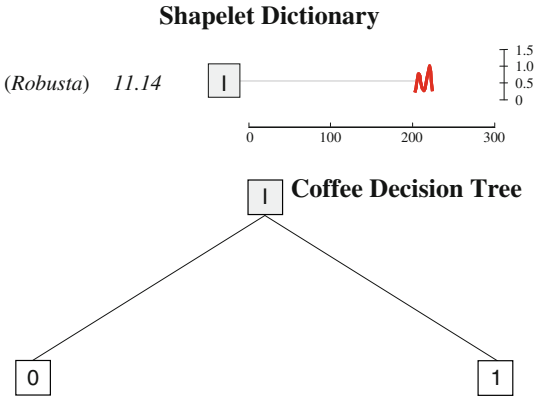

Fig. 22 (*top*) The dictionary of shapelets, together with the thresholds d_{th}. The x-axis shows the position of the shapelet in the original time series. (*bottom*) The decision tree for the coffee spectrography problem

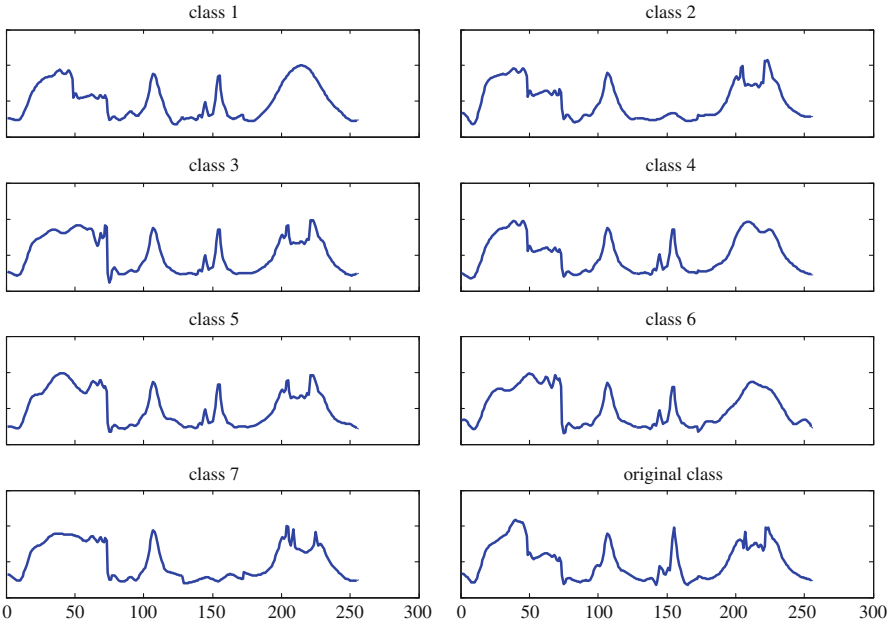

Fig. 23 One sample from each of the eight classes in the Mallat problem

the sixth node. The fifth shapelet presents the difference in the last peak. The objects on the left have a deep dip and those on the right have no dip or a shallow one.

5.9 Gait analysis

Up to this point we have only considered one-dimensional time series for simplicity. However, as the reader will appreciate, only very minor tweaks are required to allow shapelet discovery in k-dimensional time series. In this section we consider a two-dimensional time series problem in gait recognition.

We do gait analysis using *CMU Graphics Lab Motion Capture Database*(CMU). We labeled the motions in the database containing the keyword "walk" in their motion descriptions into two categories; first is the *normal* walk, with only "walk" in the motion descriptions, and the other is the *abnormal* walk, with the motion descriptions containing: "hobble walk," "walk wounded leg," "walk on toes bent forward," "hurt leg walk," "drag bad leg walk", or "hurt stomach walk." In the abnormal walks, the actors are pretending to have difficulty walking normally.

There are several difficulties in gait classification. First, the data items are multivariate time series. The distance measurement for multivariate time series is unclear because individual dimensions are correlated. Second, the time recorded for each walking motion varies a lot. Since the walking motions are collected from different subjects and used for different purposes in the original database, some short walks last only three seconds and the other long walks may last as long as 52 seconds. Third, the intra-class variance is large. The walking time series in the "normal walk" class are collected at least from four actors, everyone walking at their own pace and with their

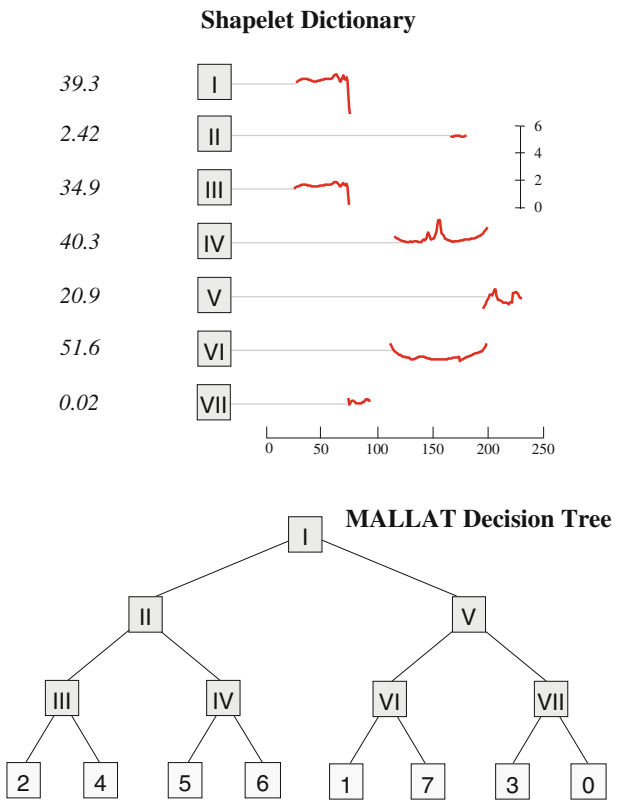

Fig. 24 (*top*) The decision tree for the MALLAT problem. The x-axis shows the position of the shapelets in the original time series. (*bottom*) The corresponding dictionary of shapelets together with the thresholds d_{th}

own walking style. Moreover, there are other motions that intermingle with walking such as stops and turns.

We simplify the multivariate motion data by considering only the z-axis value of the markers on the left and right toes. Sliding the window over a two-variable time series item, we get pairs of subsequences within the *same* sliding window. We use these pairs of subsequences as the shapelet candidates. As an illustration, an item of a walking motion is shown in Fig. 25. The two lines presented the z-axis values of the left and right toes of a walking motion. The pair of subsequences from the two time series extracted by the same sliding window is a shapelet candidate.

The distance between the data item and the shapelet candidate is defined as

$$MultiSubsequenceDist(T, S) = \min\left(\sum_{v=1}^{2} Dist(S_v, S_v^*)\right),$$

where $S^* \in \mathbf{S}_T^{|S|}$, and S_v is time series of the vth variant of S.

$$MultiSubsequenceDist(T,S) = \min(\sum_{v=1}^{2} Dist(S_v, S_v^*)),$$

where $S^* \in \mathbf{S}_T{}^{|S|}$, and S_v is time series of the v^{th} variant of S.

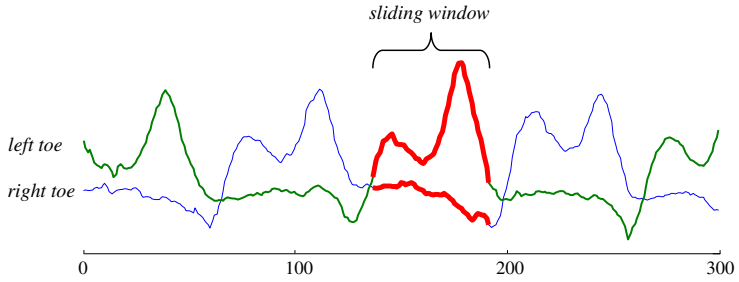

Fig. 25 The illustration of a data item of a walking motion. Each data item consists of two time series in the same time interval. Using a sliding window across the data item, we get the pairs of subsequences as the shapelet candidates

We randomly pick nine walking motions in each class as the training set and leave the others as the testing set. We test with three different segmentations of the time series. In the first segmentation, we use the entire data item unless the time series is too long. If the time series is too long, we cut it into shorter sequences of 500 data points. In the second segmentation, we make each item start at the peak of the time series of the left toe and make the item as long as three cycles. For those data items that are less than three cycles, we make the item as long as they last. In the third segmentation, we make each item start at the wave trough of the time series of the left toe, and make each item exactly two gait cycles long, abandoning those less than two cycles. The accuracy of different segmentation methods is shown in Fig. 26. We compare the accuracy between shapelet classification and *rotation invariant* nearest neighbor. The result shows that if we use merely the raw data, the nearest neighbor achieves an accuracy of 53.5% while the shapelet achieves an accuracy of 86%. Only when very careful segmentation is preprocessed can the nearest neighbor classification perform as well as the shapelet classification in accuracy, both at 90–91%. However, the shapelet still outperforms the nearest neighbor by having the classification done 45 times faster.

Figure 27 shows the shapelet that represents exactly one gait cycle. With shapelet classification, it deals with different time intervals of walking motion effectively, and therefore reduces the amount of human labor needed for segmentation.

5.10 The effect of noise on training and testing a shapelet classifier

In previous case studies, we have considered domains that have noise. For example, as we explicitly show in Sect. 5.3, many of the projectile points are broken, and in Sect. 5.4 many of the heraldic shields are torn. In an image processing context, this would be considered "occlusion noise". In Sect. 5.2 the Lightning EMP Classification dataset is probably one of the nosiest time series classification problems, shown in

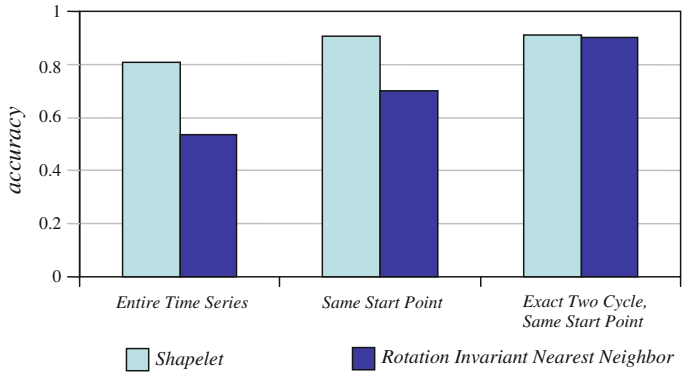

Fig. 26 The accuracy comparison between the shapelet classifier and the rotation invariant nearest neighbor under three segmentation schemes. The shapelet classification method outperforms the nearest neighbor in all three segmentation schemes. However, careful segmentation reduces the accuracy difference between the two methods

Fig. 28, *left*. In Sect. 5.8, the Mallat data in (Jeong et al. 2006) had Gaussian noise added to each of the training data to generate further samples. In Fig. 28, *right*, 17 random samples from the original class (cf. Fig. 23) are shown. The Wheat Spectrography problem, which is known to have class label noise (that is to say, some training class labels are wrong), is presented in Sect. 5.6.

In this section however, we explicitly study how noise affects the accuracy of the shapelet classifier. There are two areas where we may encounter noise; one is in the training data and the other is in the testing data. These two types of noise data are

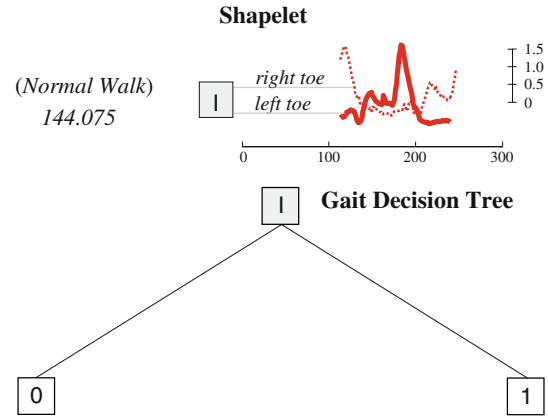

Fig. 27 (*top*) The shapelet, together with the threshold d_{th}. The x-axis shows the position of the shapelet in the original time series. (*bottom*) The decision tree for the 2-class gait analysis problem

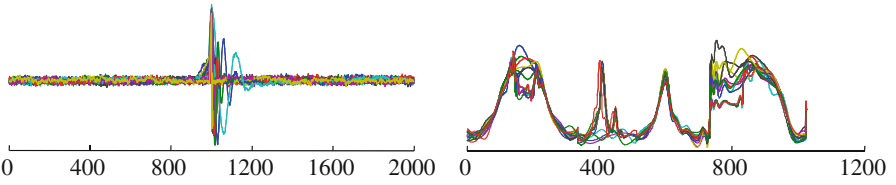

Fig. 28 Data samples from Lightning EMP dataset (*left*) and Mallat dataset (*right*)

evaluated separately in our experiments. That is, when we evaluate how noise influences the training dataset, we only add noise to the training data and keep the testing data unchanged and vice versa. Over several experiments, we add noise to various percentages of a dataset. For each time series object selected, we add a scaled Gaussian random noise $N(0, 1) \times |\max(d) - \min(d)|$ to a single random data point in the time series. The value d is the range of the time series. We evaluate the changes of accuracy on the Gun/noGun dataset in our case studies, since it is perhaps the most studied problem in the literature (Ding et al. 2008; Xi et al. 2006).

We add noise ranging from zero to one-hundred percent of the data, using increments of ten percent. Both tests are averaged over 100 runs, where both the time series to be corrupted and the location of the noisy data point are chosen randomly.

The results are reported in Fig. 29, where the solid line is the average accuracy and the dashed lines bounding it from above and below are its one standard deviation intervals.

We can see that when the training data is corrupted, the shapelet classifier continues to outperform the nearest neighbor classifier. Moreover, the margin of difference increases. However, if the noise occurs in the testing data, the shapelet classifier degrades more quickly than the nearest neighbor classifier. That is because noise could be added randomly to any data point of the time series data. If the noise happens to be added to the original best matching location of the shapelet region, it will greatly affect the accuracy of the shapelet classifier, while for the nearest neighbor classifier, the effect of the noise is mitigated by averaging over more data points (the entire time series). Let us consider an extreme case in the shapelet classifier where each data item of the testing dataset has noise. From the Gun/noGun case study we know that the length of the shapelet is 45 and the total length of the time series 150, so there is a 45/150 probability that noise will appear on the best matching location region. Supposing that half of the induced noise will mislead the sample into the opposite class, the accuracy is

$$1 - (45/150)/2 = 0.85,$$

which is consistent with the results of our experiment.

In general, these results are a mixed bag for us. The positive news is that we can construct shapelets even in the presence of significant noise. However, if the data to

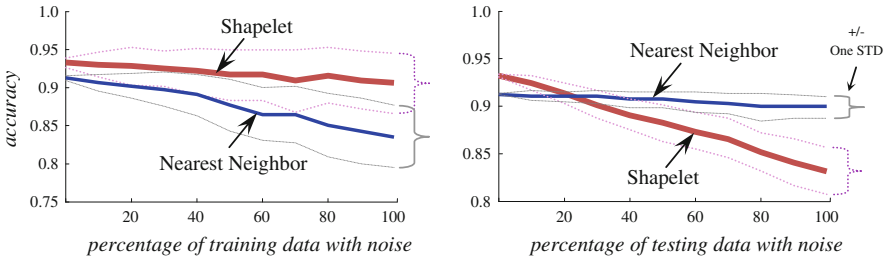

Fig. 29 The change of classification accuracy on Gun/noGun dataset as we add noise to the data. (*left*) Noise is added only to the training dataset. (*right*) Noise is added only to the testing dataset. The lighter lines indicate plus/minus one standard deviation

be classified is very noisy, we may do worse than other methods. One possible way to mitigate this problem is to find multiple shapelets that are indicative of a class and build a disjunctive classifier. For example, an item belongs to Class A if it has a subsequence that is within 5.3 of pattern S_1 **OR** within 2.3 of pattern S_2. We leave such considerations for future work.

6 Conclusions and future work

We have introduced a new primitive for time series and shape mining, the *time series shapelet*. We have shown with extensive experiments that we can find the shapelets efficiently, and that this can provide accurate, interpretable, and much faster 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 motion capture (MOCAP) analyses. In addition, although we used decision trees because of their ubiquity, it is possible that shapelets could be used in conjunction with Associative Classification (Veloso et al. 2006) and other rule-based techniques. Our work does have several limitations that we will also attempt to address in future work. The most obvious weakness is the relatively slow training time. One possible way to mitigate this limitation is by sampling the training set and only using a small subset to build the shapelets. The results in Fig. 13 suggest that this may be a tenable solution. Additionally, our method works best for low frequency data, we may need to either modify our algorithms or preprocess the data, in order to generalize to high frequency data like speech.

Acknowledgements We thank the reviewers for their helpful comments and suggestions. 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.

References

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

Berndt DJ, Clifford J (1994) Using dynamic time warping to find patterns in time series. AAAI-94 workshop on knowledge discovery in databases, Seattle, Washington, 31 July 1994

Breiman L, Friedman J, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth, Belmont, CA

Briandet R, Kemsley EK, Wilson RH (1996) Discrimination of arabica and robusta in instant coffee by fourier transform infrared spectroscopy and chemometrics. Food Chem 44(1):170–174

Chiu B, Keogh E, Lonardi S (2003) Probabilistic discovery of time series motifs. In: Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining, Washington, DC, 24–27 Aug 2003. KDD '03, ACM, New York, NY, pp 493–498

CMU Graphics Lab Motion Capture Database http://mocap.cs.cmu.edu/

Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. In: Proceedings of the VLDB endowment, Aug 2008, vol 1, 2. pp 1542–1552

Faloutsos C, Ranganathan M, Manolopoulos Y (1994) Fast subsequence matching in time-series databases. ACM SIGMOD Record 23, June 1994, vol 2. pp 419–429

Geurts P (2001) Pattern extraction for time series classification. In: Raedt LD, Siebes A (eds) Proceedings of the 5th European conference on principles of data mining and knowledge discovery, Sept

03–05, 2001 (Lecture notes in computer science), vol 2168. Springer-Verlag, London, pp 115–127

Gramm J, Guo J, Niedermeier R (2003) On exact and approximation algorithms for distinguishing substring selection. In: Proceedings of 14th fundamentals of computation theory (Lecture notes in computer science), vol 2751. Springer-Verlag, London, pp 963–971

Jeffery C (2005) http://public.lanl.gov/eads/datasets/emp/index.html

Jeong MK, Lu JC, Huo X, Vidakovic B, Chen D (2006) Wavelet-based data reduction techniques for process fault detection. Technometrics 48(1):26–40

Kadous MW (1999) Learning comprehensible descriptions of multivariate time series. In: Bratko I, Dzeroski S (eds) Proceedings of the sixteenth international conference on machine learning, June 27–30, 1999. Morgan Kaufmann Publishers, San Francisco, CA, pp 454–463

Keogh E, Kasetty S (2002) On the need for time series data mining benchmarks: a survey and empirical demonstration. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining, Edmonton, Alberta, Canada, 23–26 July 2002. KDD '02. ACM, New York, NY, pp 102–111

Keogh E, Ratanamahatana CA (2005) Exact indexing of dynamic time warping knowledge. Inf Syst 7(3):358–386

Keogh E, Wei L, Xi X, Lee S, Vlachos M (2006) LB_Keogh supports exact indexing of shapes under rotation invariance with arbitrary representations and distance measures. In: Dayal U, Whang K, Lomet D, Alonso G, Lohman G, Kersten M, Cha SK, Kim Y (eds) Proceedings of the 32nd international conference on very large data bases, Seoul, Korea, 12–15 Sept 2006. Very large data bases. VLDB Endowment, pp 882–893

Koschorreck W, Werner W (eds) (1981) Facsimile edition with commentary: Kommentar zum faksimile des codex manesse: Die grosse Heidelberger Liederhandschrift

Lang W, Morse M, Patel JM (2009) Dictionary-based compression for long time-series similarity. IEEE transactions on knowledge and data engineering, 15 Oct 2009

Lin J, Keogh E, Wei L, Lonardi S (2007) Experiencing SAX: a novel symbolic representation of time series. Data Min Knowl Discov 15(2):107–144

Martinez AM, Kak AC (2001) PCA versus LDA. IEEE Trans Patt Anal Mach Intel 23(2):228–233

Montagu JA (1840) A guide to the study of heraldry. W. Pickering, London. www.archive.org/details/guidetostudyofhe00montuoft

Rodríguez JJ, Alonso CJ (2004) Interval and dynamic time warping-based decision trees. In: Proceedings of the 2004 ACM symposium on applied computing, Nicosia, Cyprus, March 14–17, 2004. SAC '04. ACM, New York, NY, pp 548–552

Roverso D (2000) Multivariate temporal classification by windowed wavelet decomposition and recurrent neural networks. In: 3rd ANS international topical meeting on nuclear plant instrumentation, control and human-machine interface, 2000

Salzberg SL (1997) On comparing classifiers: pitfalls to avoid and a recommended approach. Data Min Knowl Discov 1:317–327

Veloso A, Meira W, Zaki MJ (2006) Lazy associative classification. In: Proceedings of the sixth international conference on Data mining, Dec 18–22, 2006. ICDM. IEEE Computer Society, Washington, DC, pp 645–654

Wikipedia description of coffee: http://en.wikipedia.org/wiki/Coffee

Wilcoxon F (1945) Individual comparisons by ranking methods. Biometrics 1:80–83

Wilson RH, Goodfellow BG (1994) Mid-infrared spectroscopy. Spectroscopic Techniques for Food Analysis

Xi X, Keogh E, Shelton C, Wei L, Ratanamahatana CA (2006) Fast time series classification using numerosity reduction. In: Proceedings of the 23rd international conference on machine learning, Pittsburgh, Pennsylvania, June 25–29, 2006. ICML '06, vol 148. ACM, New York, NY, pp 1033–1040

Yamada Y, Suzuki E, Yokoi H, Takabayashi K (2003) Decision-tree induction from time-series data based on a standard example split test. In: Proceedings of the 20th international conference on machine learning, pp 840–847

Ye L (2009) The time series shapelet Webpage. www.cs.ucr.edu/~lexiangy/shapelet.html

Ye L, Keogh E (2009) Time series shapelets: a new primitive for data mining. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, Paris, France, June 28 to July 01 2009. KDD '09. ACM, New York, NY, pp 947–956