Binary Shapelet Transform for Multiclass Time Series Classification

Aaron Bostrom(✉) and Anthony Bagnall

University of East Anglia, Norwich NR47TJ, UK
a.bostrom@uea.ac.uk

Abstract. Shapelets have recently been proposed as a new primitive for time series classification. Shapelets are subseries of series that best split the data into its classes. In the original research, shapelets were found recursively within a decision tree through enumeration of the search space. Subsequent research indicated that using shapelets as the basis for transforming datasets leads to more accurate classifiers. Both these approaches evaluate how well a shapelet splits all the classes. However, often a shapelet is most useful in distinguishing between members of the class of the series it was drawn from against all others. To assess this conjecture, we evaluate a one vs all encoding scheme. This technique simplifies the quality assessment calculations, speeds up the execution through facilitating more frequent early abandon and increases accuracy for multi-class problems. We also propose an alternative shapelet evaluation scheme which we demonstrate significantly speeds up the full search.

1 Introduction

Time series classification (TSC) is a subset of the general classification problem. The primary difference is that the ordering of attributes within each instance is important. For a set of n time series, $\mathbf{T} = \{T_1, T_2, ..., T_n\}$, each time series has m ordered real-valued observations, $T_i = \{t_{i1}, t_{i2}, ..., t_{im}\}$ and a class value c_i. The aim of TSC is to determine a function that relates the set of time series to the class values.

One recently proposed technique for TSC is to use shapelets [1]. Shapelets are subseries of the series $\mathbf{T}$ that best split the data into its classes. Shapelets can be used to detect discriminatory phase independent features that cannot be found with whole series measures such as dynamic time warping. Shapelet based classification involves measuring the similarity between a shapelet and each series, then using this similarity as a discriminatory feature for classification. The original shapelet-based classifier [1] embeds the shapelet discovery algorithm in a decision tree, and uses information gain to assess the quality of candidates. A shapelet is found at each node of the tree through an enumerative search. More recently, we proposed using shapelets as a transformation [2]. The shapelet transform involves a single-scan algorithm that finds the best k shapelets in a set of n time series. We use this algorithm to produce a transformed dataset,

A. Hameurlain et al. (Eds.): TLDKS XXXII, LNCS 10420, pp. 24–46, 2017.
DOI: 10.1007/978-3-662-55608-5_2

where each of the k features is the distance between the series and one shapelet. Hence, the value of the i^{th} attribute of the j^{th} record is the distance between the j^{th} record and the i^{th} shapelet. The primary advantages of this approach are that we can use the transformed data in conjunction with any classifier, and that we do not have to search sequentially for shapelets at each node. However, it still requires an enumerative search throughout the space of possible shapelets and the full search is $O(n^2m^4)$. Improvements for the full search technique were proposed in [1,3] and heuristics techniques to find approximations of the full search were described in [4–6].

Our focus is only on improving the exhaustive search. One of the problems of the shapelet search is the quality measures assess how well the shapelet splits all the classes. For multi-class problems measuring how well a shapelet splits all the classes may confound the fact that it actually represents a single class. Consider, for example, a shapelet in a data set of heartbeat measurements of patients with a range of medical conditions. It is more intuitive to imagine that a shapelet might represent a particular condition such as arrhythmia rather than discriminating between multiple conditions equally well. We redefine the transformation so that we find shapelets assessed on their ability to distinguish one class from all, rather than measures that separate all classes. This improves accuracy on multi-class problems and allows us to take greater advantage of the early abandon described in [1].

A further problem with the shapelet transform is that it may pick an excessive number of shapelets representing a single class. By definition, a good shapelet will appear in many series. The best way we have found to deal with this is to generate a large number of shapelets then cluster them [2]. However, there is still a risk that one class is generally easier to classify and hence has a disproportionate number of shapelets in the transform. The binary shapelet allows us to overcome this problem by balancing the number of shapelets we find for each class.

Finally, we describe an alternative way of enumerating the shapelet search that facilitates greater frequency of early abandon of the distance calculation.

2 Shapelet Based Classification

The shapelet transform algorithm described in [2] is summarised in Algorithm 1. Initially, for each time series, all candidates of length *min* to *max* are generated (i.e. extracted and normalised in the method *generateCandidates*). Then the distance between each shapelet and the other $n-1$ series are calculated to form the order list, D_S. Distance between a shapelet S of length l and a series T is given by

$$sDist(S,T) = \min_{w \in W_l} (dist(S,w)) \tag{1}$$

where W_l is the set of all l length subseries in T and $dist$ is the Euclidean distance between the equal length series S and w. The order list is used to determine the quality of the shapelet in the *assessCandidate* method. Quality can be assessed by information gain [1] or alternative measures such as the F, moods median or

rank order statistic [7]. Once all the shapelets for a series are evaluated they are sorted and the lowest quality overlapping shapelets are removed. The remaining candidates are then added to the shapelet set. By default, we set $k = 10n$ with the caveat that we do not accept shapelets that have zero information gain.

Algorithm 1. FullShapeletSelection($\mathbf{T}$, min, max, k)

Input: A list of time series $\mathbf{T}$, min and max length shapelet to search for and k, the maximum number of shapelets to find)
Output: A list of k shapelets

```
1: kShapelets ← ∅
2: for all T_i in T do
3:    shapelets ← ∅
4:    for l ← min to max do
5:       W_{i,l} ← generateCandidates(T_i, l)
6:       for all subseries S in W_{i,l} do
7:          D_S ← findDistances(S, T)
8:          quality ← assessCandidate(S, D_S)
9:          shapelets.add(S, quality)
10:   sortByQuality(shapelets)
11:   removeSelfSimilar(shapelets)
12:   kShapelets ← merge(k, kShapelets, shapelets)
13: return kShapelets
```

Once the best k shapelets have been found, the transform is performed with Algorithm 2. A more detailed description can be found in [8].

Extensions to the basic shapelet finding algorithm can be categorized into techniques to speed up the average case complexity of the exact technique and those that use heuristic search. The approximate techniques include reducing the dimensionality of the candidates and using a hash table to filter [4], searching the space of shapelet values (rather than taking the values from the train set series) [5] and randomly sampling the candidate shapelets [6]. Our focus is on improving the accuracy and speed of the full search. Two forms of early abandon described in [1] can improve the average case complexity. Firstly, the Euclidean distance calculations within the $sDist$ (Eq. 1) can be terminated early if they exceed the best found so far. Secondly, the shapelet evaluation can be abandoned early if $assessCandidate$ is updated as the $sDist$ are found and the best possible outcome for the candidate is worse than the current top candidates.

A speedup method involving trading memory for speed is proposed in [3]. For each pair of series T_i, T_j, cumulative sum, squared sum, and cross products of T_i and T_j are pre-calculated. With these statistics, the distance between subseries can be calculated in constant time, making the shaplet-discovery algorithm $O(n^2m^3)$. However, pre-calculating of the cross products between all series prior to shapelet discovery requires $O(n^2m^2)$ memory, which is infeasible for most problems. Instead, [3] propose calculating these statistics prior to the start of the scan of each series, reducing the requirement to $O(nm^2)$ memory, but increasing

the time overhead. Further refinements applicable to shapelets were described in [9], most relevant of which was a reordering of the sequence of calculations within the *dist* function to increase the likelihood of early abandon. The key observation is that because all series are normalized, the largest absolute values in the candidate series are more likely to contribute large values in the distance function. Hence, if the distances between positions with larger candidate values are evaluated first, then it is more likely the distance can be abandoned early. This can be easily implemented by creating an enumeration through the normalized candidate at the beginning, and adds very little overhead. We use this technique in all experiments.

Algorithm 2. $FullShapeletTransform$(Shapelets **S**,**T**)

```
1: T' ← ∅
2: for all T in T do
3:    T' ←<>
4:    for all shapelets S in S do
5:       dist ← sDist(S, T)
6:       T' ← append(T', dist)
7:    T' ← append(T', T.class)
8:    T' ← T' ∪ T'
9: return T'
```

3 Classification Technique

Once the transform is complete we can use any classifier on the problem. To reduce classifier induced variance we use a heterogenous ensemble of eight classifiers. The classifiers used are the WEKA [10] implementations of k Nearest Neighbour (where k is set through cross validation), Naive Bayes, C4.5 decision tree [11], Support Vector Machines [12] with linear and quadratic basis function kernels, Random Forest [13] (with 100 trees), Rotation Forest [14] (with 10 trees) and a Bayesian network. Each classifier is assigned a weight based on the cross validation training accuracy, and new data are classified with a weighted vote. The set of classifiers were chosen to balance simple and complex classifiers that use probabilistic, tree based and kernel based models. With the exception of k-NN, we do not optimise parameter settings for these classifiers via cross validation. More details are given in [8].

4 Alternate Shapelet Techniques

4.1 Fast Shapelets

The fast shapelet (FS) algorithm was proposed in 2013 [4]. The algorithm is a refinement of original decision tree shapelet selection algorithm. It employs a

number of techniques to speed up the finding and pruning of shapelet candidates at each node of the tree [15]. The major changes made to the enumerative search is the introduction of symbolic aggregate approximation (SAX) [16] as a means for reducing the length of each series as well as smoothing and discretising the data. The other major advantage of using the SAX representation is that shapelet candidates can be pruned by using a collision table metric which highly correlates with Information Gain to reduce the amount of work performed in the quality measure stage. In the description in Algorithm 3 the decision tree has been omitted to improve clarity.

The first stage of the shapelet finding process is to create a list of SAX words [16]. The basic concept of SAX is a two stage process of dimension reduction and discretisation. SAX uses piece-wise aggregate approximation (PAA), to transform a time series into a number of smaller averaged sections before z normalization. This reduced series is discretised into a given alphabet size. Breakpoints are defined by equally likely areas of a standard normal distribution and each series forms a single string of characters. These strings are much smaller in length compared to the original series, so finding shapelets is faster. To increase the likelihood of word collisions a technique called random projects is employed. Given some SAX words random projection reduces their dimensionality by masking a number of their letters. The SAX words are randomly projected a number of times, the projected words are hashed and a frequency table for all the SAX words is built.

From this frequency table a new set of tables can be built which represent how common the SAX word is with respect to each class. A score for each SAX word can be calculated based on these grouping scores, and this value is used for assessing the distinguishing power of each SAX word. From this scoring process a list of the top k SAX shapelets can be created. These top k SAX shapelets are transformed back into their original series, where the shapelets information gain can be calculated. The best shapelet then forms the splitting rule in the decision tree.

Algorithm 3. FindBestShapelet(Set of time series **T**)

```
1: bsfShapelet, shapelet
2: topK = 10
3: for length ← 5 to m do
4:     SAXList =FindSAXWords(T, length)
5:     RandomProjection(SAXList)
6:     ScoreList =ScoreAllSAX(SAXList)
7:     shapelet =FindBestSAX(ScoreList, SAXList, topK)
8:     if bsfShapelet < shapelet then
9:         bsfShapelet = shapelet
10: return bsfShapelet
```

4.2 Learn Shapelets

Learn shapelets (LS) was proposed by Grabocka *et al.* in 2014 [5]. Rather than uses subseries in the data as candidate shapelets, LS searches the space of all possible shapelets using a gradient descent approach on an initial set of shapelets found through clustering. An set of series is taken from the data and clustered using k-Means. The resulting centroids are refined with a two stage gradient descent model of shapelet refinement and logistic regression assessment. The learning is continued until either the model has converged or the number of iterations has exceeded a hard limit ($maxIter$). We show a high level overview of the algorithm presented in [5] in Algorithm 4.

Algorithm 4. LearnShapelets(Set of time series $\mathbf{T}$)

```
Parameters: K,R,L_min,η,λ
S ← InitKMeans(T,K,R,L_min)
W ← InitWeights(T,K,R)
for i ← maxIter do
    M ← updateModel(T,S,α, L_min, R)
    L ← updateLoss(T,M,W)
    W,S← updateWandS(T,M,W,S, η, R, L_min, L, λ_W, α)
    if diverged() then
        i = 0
        η = η/3
```

5 Shapelet Transform Refinements

5.1 Binary Shapelets

The standard shapelet assessment method measures how well the shapelet splits up all the classes. There are three potential problems with this approach when classifying multi-class problems. The problems apply to all possible quality measures, but we use information gain to demonstrate the point. Firstly, useful information about a single class may be lost. For example, suppose we have a four class problem and a shapelet produces the order line presented in Fig. 1, where each colour represents a different class.

The first shapelet groups all of class 1 very well, but cannot distinguish between classes 2, 3 and 4 and hence has a lower information gain than the split produced by the second shapelet in Fig. 1 which separates class 1 and 2 from class 3 and 4. The more classes there are, the more likely it is that the quantification of the ability of a shapelet to separate out a single class will be overwhelmed by the mix of other class values. We can mitigate against this potential problem by defining a binary shapelet as one that is assessed by how well it splits the class of the series it originated from all the other classes. The second problem with

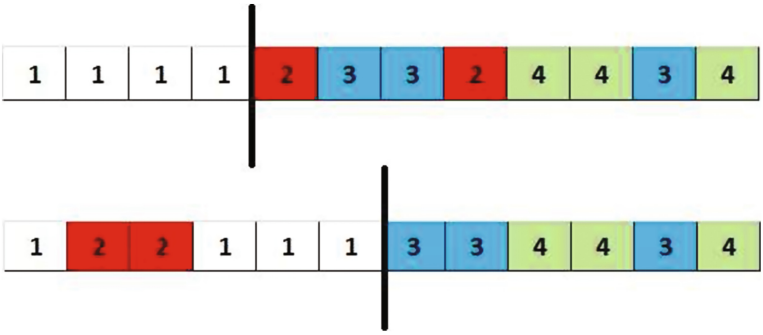

Fig. 1. An example order line split for two shapelets. The top shapelet discriminates between class 1 and the rest perfectly, yet has lower information gain than the orderline shown below it.

searching all shapelets with multi-class assessment arises if one class is much easier to classify than the others. In this case it is likely that more shapelets will be found for the easy class than for the other classes. Although our principle is to find a large number of shapelets (ten times the number of training cases) and let the classifier deal with redundant features, there is still a risk that a large number of similar shapelets for one class will crowd out useful shapelets for another class. If we use binary shapelets we can simply allocate a maximum number of shapelets to each class. We adopt the simple approach of allocating a maximum of k/c shapelets to each class, where c is the number of classes. The final problem is that the shapelet early abandon described in [1] is not useful for multi-class problems. Given a partial orderline and a split point, the early abandon works by upper bounding the information gain by assigning the unassigned series to the side of the split that would give the maximum gain. However, the only way to do this with multi-class problems is to try all permutations. The time this takes quickly rises to offset the possible benefits from the early abandon. If we restrict our attention to just binary shapelets then we can take maximum advantage of the early abandon. The binary shapelet selection is described by Algorithm 5.

5.2 Changing the Shapelet Evaluation Order

Shapelets are phase independent. However, for many problems the localised features are at most only weakly independent in phase, i.e. the best matches will appear close to the location of the candidate. Finding a good match early in *sDist* increases the likelihood of an early abandon for each *dist* calculation. Hence, we redefine the order of iteration of the *dist* calculations within *sDist* so that we start with the index the shapelet was found at and move consecutively left and right from that point. Figure 2 demonstrates the potential benefit of this approach. The scan from the beginning is unable to early abandon on any of the subseries before the best match. The scan originating at the candidates location finds the best match faster an hence can early abandon on all the distance calcu-

Algorithm 5. BinaryShapeletSelection($\mathbf{T}$, min, max, k)

Input: A list of time series $\mathbf{T}$, min and max length shapelet to search for and k,the maximum number of shapelets to find)
Output: A list of k Shapelets

```
1: numClasses ← getClassDistribution(T)
2: kShapeletsMap ← ∅
3: prop ← k/numClasses
4: for all T_i in T do
5:    shapelets ← ∅
6:    for l ← min to max do
7:       W_{i,l} ← generateCandidates(T_i, l)
8:       for all subseries S in W_{i,l} do
9:          D_S ← findDistances(S, T)
10:         quality ← assessCandidate(S, D_S)
11:         shapelets.add(S, quality)
12:   sortByQuality(shapelets)
13:   removeSelfSimilar(shapelets)
14:   kShapelets ← kShapeletsMap.get(T.class)
15:   kShapelets ← merge(prop, kShapelets, shapelets)
16:   kShapeletsMap.add(kShapelets, T.class)
17: return kShapeletsMap.asList()
```

lations at the beginning of the series. Hence, if the location of the best shapelet is weakly phase dependent, we would expect to observe an improvement in the time complexity. The revised function $sDist$, which is a subroutine of $findDistances$ (line 9 in Algorithm 5), is described in Algorithm 6.

6 Results

We demonstrate the utility of our approach through experiments using 74 benchmark multi-class datasets from the UCR Time Series Classification archive [17]. In common with the vast majority of research in this field, we present results on the standard train/test split. The min and max size of the shapelet are set to 3 and m (series length). As a sanity check, we have also evaluated the binary shapelets on two class problems to demonstrate there is negligible difference. On 25 two class problems, the full transform was better on 6, the binary transform better on 19 and they were tied on 1. All the results and the code to generate them are available from [18,19].

6.1 Accuracy Improvement on Multi-class Problems

Table 1 gives the results for the full shapelet transform and the binary shapelet transform on problems with 2-50 classes. Overall, the binary shapelet transform is better on 48 data sets, the full transform better on 21 and on 4 they are equal. On multi class problems, the binary shapelet transform is better on 29 problems

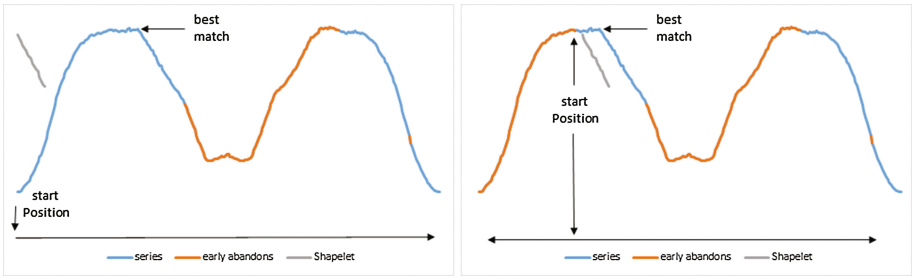

Fig. 2. An example of Euclidean distance early abandon where the *sDist* scan starts from the beginning (a) and from the place of origin of the candidate shapelet (b). For the scan from the beginning, there are no early abandons until the scan has passed the best match. Because the best match is close to the location of the candidate shapelet, starting from the shapelets original location allows for a greater number of early abandons.

Algorithm 6. $sDist$(shapelet S, series T_i)

```
1: subSeq ← getSubSeq(T_i, S.startPos, S.length)
2: bestDist ← euclideanDistance(subSeq, S)
3: i ← 1
4: while leftExists || rightExists do
5:    leftExists ← S.startPos − i ≥ 0
6:    rightExists ← S.startPos + i < T_i.length
7:    if rightExists then
8:       subSeq ← getSubSeq(T_i, S.startPos + i, S.length)
9:       currentDist ← earlyAbandonDistance(subSeq, S, bestDist)
10:      if currentDist > bestDist then
11:         bestDist ← currentDist
12:   if leftExists then
13:      subSeq ← getSubSeq(T_i, S.startPos − i, S.length)
14:      currentDist ← earlyAbandonDistance(subSeq, S, bestDist)
15:      if currentDist > bestDist then
16:         bestDist ← currentDist
17:   i ← i + 1
18: return bestDist
```

compared with the full shapelet transform being better on 15. The difference between the full and the binary shapelet transform is significant at the 5% level using a paired T test.

Figure 3 shows the plot of the difference in accuracy of the full and binary shapelet transform plotted against the number of classes. There is a clear trend of increasing accuracy for the binary transform as the number of classes continues. This is confirmed in Table 2, which presents the same data grouped into bins of ranges of number of classes.

Table 1. Full shapelet transform vs. Binary shapelet transform.

Dataset	#Classes	FST	binaryST
Adiac	37	0.565	**0.783**
ArrowHead	3	**0.771**	0.737
Beef	5	0.833	**0.9**
BeetleFly	2	**0.75**	0.6
BirdChicken	2	0.75	**0.8**
Car	4	0.733	**0.917**
CBF	3	**0.997**	0.974
ChlorineConcentration	3	**0.7**	0.7
CinCECGtorso	4	0.846	**0.954**
Coffee	2	**1**	0.964
Computers	2	0.7	**0.736**
CricketX	12	**0.782**	0.772
CricketY	12	0.764	**0.779**
CricketZ	12	0.772	**0.787**
DiatomSizeReduction	4	0.876	**0.925**
DistalPhalanxOutlineAgeGroup	3	0.741	**0.77**
DistalPhalanxOutlineCorrect	2	0.736	**0.775**
DistalPhalanxTW	6	0.633	**0.662**
Earthquakes	2	0.734	**0.741**
ECGFiveDays	2	**0.999**	0.984
FaceAll	14	0.737	**0.779**
FaceFour	4	**0.943**	0.852
FacesUCR	14	**0.913**	0.906
fiftywords	50	**0.719**	0.705
fish	7	0.977	**0.989**
FordA	2	0.927	**0.971**
FordB	2	0.789	**0.807**
GunPoint	2	0.98	**1**
Haptics	5	0.477	**0.523**
Herring	2	**0.672**	0.672
InlineSkate	7	**0.385**	0.373
ItalyPowerDemand	2	**0.952**	0.948
LargeKitchenAppliances	3	**0.883**	0.859
Lightning2	2	0.656	**0.738**
Lightning7	7	**0.74**	0.726
MALLAT	8	0.94	**0.964**

(Continued)

Table 1. *(Continued)*

Dataset	#Classes	FST	binaryST
MedicalImages	10	0.604	**0.67**
MiddlePhalanxOutlineAgeGroup	3	0.63	**0.643**
MiddlePhalanxOutlineCorrect	2	0.725	**0.794**
MiddlePhalanxTW	6	**0.539**	0.519
MoteStrain	2	0.891	**0.897**
NonInvasiveFatalECGThorax1	42	0.9	**0.95**
NonInvasiveFatalECGThorax2	42	0.903	**0.951**
OliveOil	4	**0.9**	0.9
OSULeaf	6	0.715	**0.967**
PhalangesOutlinesCorrect	2	0.748	**0.763**
Plane	7	**1**	1
ProximalPhalanxOutlineAgeGroup	3	**0.854**	0.844
ProximalPhalanxOutlineCorrect	2	**0.9**	0.883
ProximalPhalanxTW	6	0.771	**0.805**
RefrigerationDevices	3	0.557	**0.581**
ScreenType	3	**0.533**	0.52
ShapeletSim	2	0.919	**0.956**
SmallKitchenAppliances	3	0.773	**0.792**
SonyAIBORobotSurface1	2	**0.933**	0.864
SonyAIBORobotSurface2	2	0.885	**0.934**
StarLightCurves	3	0.976	**0.979**
SwedishLeaf	15	0.907	**0.928**
Symbols	6	**0.886**	0.882
SyntheticControl	6	**0.983**	0.983
ToeSegmentation1	2	0.956	**0.965**
ToeSegmentation2	2	0.854	**0.908**
Trace	4	0.98	**1**
TwoLeadECG	2	0.996	**0.997**
TwoPatterns	4	0.941	**0.955**
UWaveGestureLibraryX	8	0.784	**0.803**
UWaveGestureLibraryY	8	0.697	**0.73**
UWaveGestureLibraryZ	8	0.727	**0.748**
wafer	2	0.998	**1**
WordSynonyms	25	**0.597**	0.571
Worms	5	0.701	**0.74**
WormsTwoClass	2	0.766	**0.831**
Yoga	2	0.805	**0.818**
Total wins		21	48

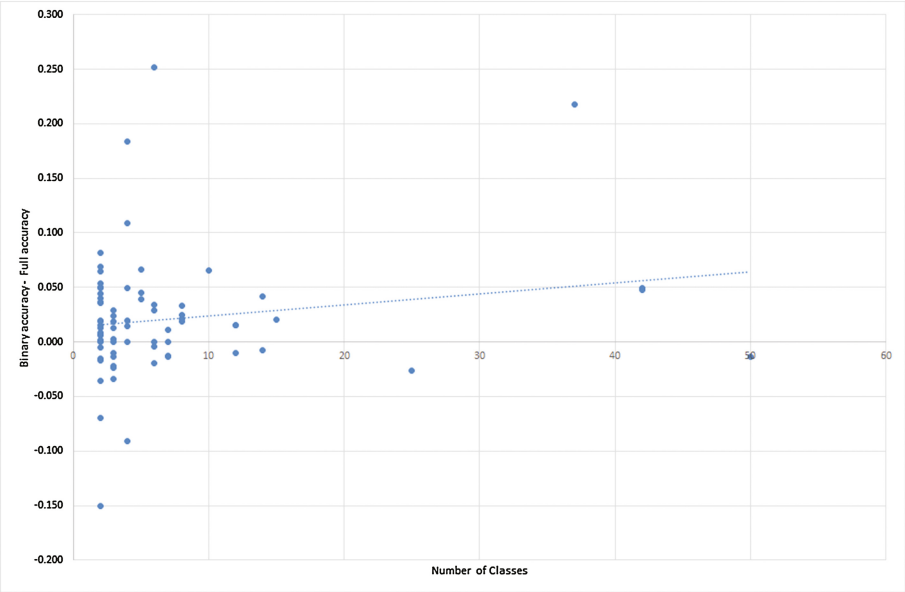

Fig. 3. Number of classes plotted against the difference in error between the full shapelets and the binary shapelets. A positive number indicates the binary shapelets are better. The dotted line is the least squares regression line.

6.2 Accuracy Comparison to Other Shapelet Methods

To establish the efficacy of the binary shapelet approach we wanted to thoroughly compare it to other shapelet approaches within the literature, Fast Shapelets (FS) [4] and Learn Shapelets (LS) [5]. The 85 datasets from the updated repository [18] were stratified and resampled 100 times, to produce 8500 problems. All folds and results are reproducible within our common Java WEKA [10] framework [19]. Where possible we tried to match the experimental procedure for parameter searching outlined in the original work. We performed very extensive tests and analysis on the algorithms implemented to make sure they were

Table 2. Number of data sets the binary shapelet beats the full shapelet split by number of classes.

Number of classes	Full better	Binary better
2	6	19
3 to 5	7	13
6 to 9	4	8
10 and above	4	8
All	21	48

Table 3. The average accuracies for the ShapeletTransform, LearnShapelets and Fast-Shapelets averaged over a 100 resamples for the 85 UCR datasets

Datasets	ST	LS	FS
Adiac	**0.768**	0.527	0.555
ArrowHead	**0.851**	0.841	0.675
Beef	**0.736**	0.698	0.502
BeetleFly	**0.874**	0.861	0.795
BirdChicken	**0.927**	0.863	0.862
Car	**0.902**	0.856	0.736
CBF	**0.986**	0.977	0.924
ChlorineConcentration	**0.682**	0.586	0.566
CinCECGtorso	**0.918**	0.855	0.741
Coffee	**0.995**	0.995	0.917
Computers	**0.785**	0.654	0.5
CricketX	**0.777**	0.744	0.479
CricketY	**0.762**	0.726	0.509
CricketZ	**0.798**	0.754	0.466
DiatomSizeReduction	0.911	**0.927**	0.873
DistalPhalanxOutlineCorrect	**0.829**	0.822	0.78
DistalPhalanxOutlineAgeGroup	**0.819**	0.81	0.745
DistalPhalanxTW	**0.69**	0.659	0.623
Earthquakes	0.737	0.742	**0.747**
ECG200	0.84	**0.871**	0.806
ECG5000	**0.943**	0.94	0.922
ECGFiveDays	0.955	0.985	**0.986**
ElectricDevices	**0.895**	0.709	0.262
FaceAll	**0.968**	0.926	0.772
FaceFour	0.794	**0.957**	0.869
FacesUCR	0.909	**0.939**	0.701
FiftyWords	**0.713**	0.694	0.512
Fish	**0.974**	0.94	0.742
FordA	**0.965**	0.895	0.785
FordB	**0.915**	0.89	0.783
GunPoint	**0.999**	0.983	0.93
Ham	0.808	**0.832**	0.677
HandOutlines	**0.924**	0.837	0.841
Haptics	**0.512**	0.478	0.356

(Continued)

Table 3. *(Continued)*

Datasets	ST	LS	FS
Herring	**0.653**	0.628	0.558
InlineSkate	**0.393**	0.299	0.257
InsectWingbeatSound	**0.617**	0.55	0.488
ItalyPowerDemand	**0.953**	0.952	0.909
LargeKitchenAppliances	**0.933**	0.765	0.419
Lightning2	0.659	**0.759**	0.48
Lightning7	0.724	**0.765**	0.101
Mallat	**0.972**	0.951	0.893
Meat	**0.966**	0.814	0.924
MedicalImages	0.691	**0.704**	0.609
MiddlePhalanxOutlineCorrect	0.815	**0.822**	0.716
MiddlePhalanxOutlineAgeGroup	**0.694**	0.679	0.613
MiddlePhalanxTW	**0.579**	0.54	0.519
MoteStrain	**0.882**	0.876	0.793
NonInvasiveFatalECGThorax1	**0.947**	0.6	0.71
NonInvasiveFatalECGThorax2	**0.954**	0.739	0.758
OliveOil	**0.881**	0.172	0.765
OSULeaf	**0.934**	0.771	0.679
PhalangesOutlinesCorrect	**0.794**	0.783	0.73
Phoneme	**0.329**	0.152	0.173
Plane	**1**	0.995	0.97
ProximalPhalanxOutlineCorrect	**0.881**	0.793	0.797
ProximalPhalanxOutlineAgeGroup	**0.841**	0.832	0.797
ProximalPhalanxTW	**0.803**	0.794	0.716
RefrigerationDevices	**0.761**	0.642	0.574
ScreenType	**0.676**	0.445	0.365
ShapeletSim	0.934	0.933	**1**
ShapesAll	**0.854**	0.76	0.598
SmallKitchenAppliances	**0.802**	0.663	0.333
SonyAIBORobotSurface1	0.888	0.906	**0.918**
SonyAIBORobotSurface2	**0.924**	0.9	0.849
StarlightCurves	**0.977**	0.888	0.908
Strawberry	**0.968**	0.925	0.917
SwedishLeaf	**0.939**	0.899	0.758

(Continued)

Table 3. *(Continued)*

Datasets	ST	LS	FS
Symbols	0.862	**0.919**	0.908
SyntheticControl	0.987	**0.995**	0.92
ToeSegmentation1	**0.954**	0.934	0.904
ToeSegmentation2	**0.947**	0.943	0.873
Trace	**1**	0.996	0.998
TwoLeadECG	0.984	**0.994**	0.92
TwoPatterns	0.952	**0.994**	0.696
UWaveGestureLibraryX	**0.806**	0.804	0.694
UWaveGestureLibraryY	**0.737**	0.718	0.591
UWaveGestureLibraryZ	**0.747**	0.737	0.638
UWaveGestureLibraryAll	**0.942**	0.68	0.766
Wafer	**1**	0.996	0.981
Wine	**0.926**	0.524	0.794
WordSynonyms	**0.582**	0.581	0.461
Worms	**0.719**	0.642	0.622
WormsTwoClass	**0.779**	0.736	0.706
Yoga	0.823	**0.833**	0.721
Total wins	71	14	4

identical to the available source code, and provide statistically similar results as published. This often required working with the original authors to help replicate and fix errors.

We present the results of the mean accuracy for the binary shapelet transform, FS and LS in Table 3. We found that binary shapelets was better than FS and LS on 67 problems, and on a pair wise t-test was significantly better at the 5% level. We show the critical difference of the three approaches in Fig. 4.

6.3 Accuracy Comparison to Standard Approaches

Using the same methodology for comparing shapelet methodsm we compared the shapelet approach to more standard approaches. These were 1-nearest neighbour using Euclidean distance, 1 nearest neighbour with dynamic time warping and lastly rotation forest. We present the results in Table 4 showing that on the 85 datasets, the binary shapelet wins on 55 problems and is significantly better than other the standard approaches. We show the comparison of these classifiers in the critical difference diagram in Fig. 5.

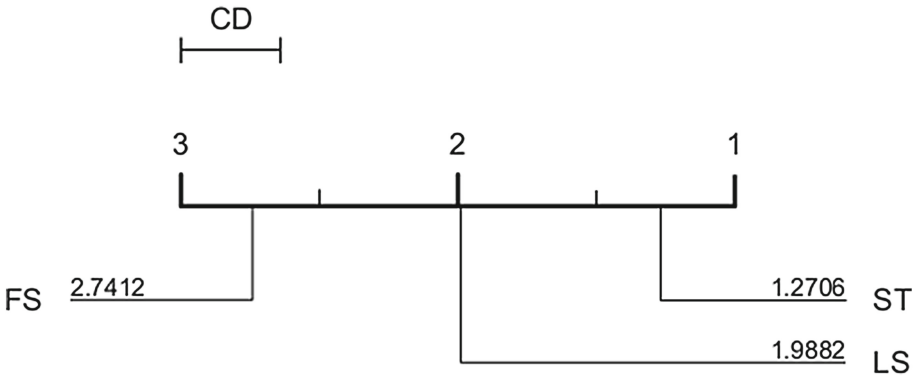

Fig. 4. The critical difference diagram of the Shapelet Transform, Fast Shapelets and Learn Shapalets, the data is presented in 3

Table 4. The average accuracies for the BinaryShapeletTransform, 1NN with Euclidean distance, 1NN with DTW setting window size through cross-validation and Rotation Forest, averaged over a 100 resamples for the 85 UCR datasets

Datasets	ST	ED	DTW	RotF
Adiac	**0.768**	0.617	0.615	0.754
ArrowHead	**0.851**	0.841	0.829	0.789
Beef	0.736	0.533	0.532	**0.819**
BeetleFly	**0.875**	0.686	0.785	0.791
BirdChicken	**0.927**	0.693	0.823	0.747
Car	**0.902**	0.724	0.714	0.788
CBF	**0.986**	0.870	0.974	0.898
ChlorineConcentration	0.682	0.652	0.651	**0.846**
CinCECGtorso	0.918	0.891	**0.928**	0.712
Coffee	**0.995**	0.981	0.981	0.995
Computers	**0.785**	0.575	0.688	0.666
CricketX	**0.777**	0.579	0.774	0.620
CricketY	**0.762**	0.545	0.749	0.599
CricketZ	**0.798**	0.589	0.779	0.626
DiatomSizeReduction	0.911	0.943	**0.944**	0.881
DistalPhalanxOutlineCorrect	**0.829**	0.744	0.756	0.812
DistalPhalanxOutlineAgeGroup	**0.819**	0.731	0.733	0.807
DistalPhalanxTW	0.690	0.628	0.621	**0.692**
Earthquakes	0.737	0.682	0.696	**0.759**
ECG200	0.840	**0.879**	0.872	0.851

(Continued)

Table 4. *(Continued)*

Datasets	ST	ED	DTW	RotF
ECG5000	**0.943**	0.927	0.927	0.942
ECGFiveDays	**0.955**	0.811	0.835	0.860
ElectricDevices	**0.895**	0.695	0.783	0.788
FaceAll	**0.968**	0.878	0.958	0.905
FaceFour	0.794	0.778	0.851	**0.853**
FacesUCR	0.909	0.764	**0.918**	0.784
FiftyWords	0.713	0.659	**0.770**	0.675
Fish	**0.974**	0.802	0.814	0.859
FordA	**0.965**	0.682	0.684	0.837
FordB	**0.915**	0.648	0.661	0.808
GunPoint	**0.999**	0.891	0.947	0.924
Ham	0.808	0.758	0.749	**0.822**
HandOutlines	**0.924**	0.853	0.857	0.912
Haptics	**0.512**	0.386	0.409	0.469
Herring	**0.653**	0.496	0.545	0.608
InlineSkate	0.393	0.326	**0.400**	0.340
InsectWingbeatSound	0.617	0.553	0.555	**0.633**
ItalyPowerDemand	0.953	0.954	0.951	**0.967**
LargeKitchenAppliances	**0.933**	0.524	0.788	0.622
Lightning2	0.659	0.716	**0.831**	0.760
Lightning7	0.724	0.618	**0.744**	0.701
Mallat	**0.972**	0.933	0.943	0.946
Meat	0.966	0.981	0.980	**0.994**
MedicalImages	0.691	0.701	0.748	**0.756**
MiddlePhalanxOutlineCorrect	0.815	0.776	0.775	**0.820**
MiddlePhalanxOutlineAgeGroup	**0.694**	0.583	0.570	0.669
MiddlePhalanxTW	**0.579**	0.493	0.496	0.568
MoteStrain	**0.882**	0.866	0.862	0.859
NonInvasiveFatalECGThorax1	**0.947**	0.822	0.820	0.899
NonInvasiveFatalECGThorax2	**0.954**	0.889	0.884	0.928
OliveOil	0.881	0.877	0.876	**0.889**
OSULeaf	**0.934**	0.573	0.634	0.587
PhalangesOutlinesCorrect	0.794	0.768	0.766	**0.833**
Phoneme	**0.329**	0.104	0.230	0.127

(Continued)

Table 4. *(Continued)*

Datasets	ST	ED	DTW	RotF
Plane	**1.000**	0.967	0.994	0.986
ProximalPhalanxOutlineCorrect	**0.881**	0.818	0.816	0.875
ProximalPhalanxOutlineAgeGroup	0.841	0.770	0.765	**0.847**
ProximalPhalanxTW	0.803	0.713	0.732	**0.808**
RefrigerationDevices	**0.761**	0.426	0.573	0.570
ScreenType	**0.676**	0.432	0.465	0.466
ShapeletSim	**0.934**	0.505	0.652	0.488
ShapesAll	**0.854**	0.754	0.804	0.760
SmallKitchenAppliances	**0.802**	0.370	0.674	0.714
SonyAIBORobotSurface1	**0.888**	0.785	0.804	0.814
SonyAIBORobotSurface2	**0.924**	0.855	0.855	0.846
StarlightCurves	**0.977**	0.853	0.908	0.970
Strawberry	0.968	0.956	0.955	**0.974**
SwedishLeaf	**0.939**	0.772	0.842	0.884
Symbols	0.862	0.876	**0.920**	0.842
SyntheticControl	0.987	0.903	**0.989**	0.967
ToeSegmentation1	**0.954**	0.612	0.722	0.578
ToeSegmentation2	**0.947**	0.781	0.851	0.646
Trace	**1.000**	0.778	0.993	0.932
TwoLeadECG	**0.984**	0.735	0.881	0.928
TwoPatterns	0.952	0.906	**0.999**	0.928
UWaveGestureLibraryX	**0.806**	0.740	0.777	0.779
UWaveGestureLibraryY	**0.737**	0.665	0.695	0.717
UWaveGestureLibraryZ	**0.747**	0.661	0.687	0.728
UWaveGestureLibraryAll	0.942	0.943	**0.960**	0.946
Wafer	**1.000**	0.995	0.995	0.995
Wine	**0.926**	0.893	0.891	0.919
WordSynonyms	0.582	0.615	**0.730**	0.586
Worms	**0.719**	0.491	0.569	0.605
WormsTwoClass	**0.779**	0.624	0.661	0.657
Yoga	0.823	0.840	**0.858**	0.854
Total wins	55	1	13	16

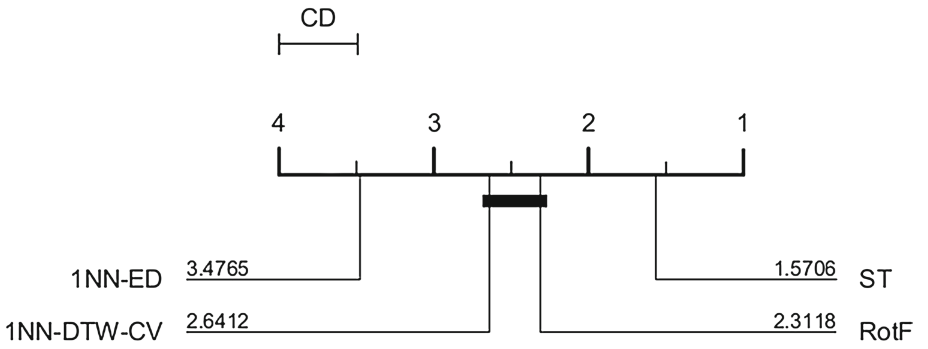

Fig. 5. The critical difference diagram showing the comparison of the new Shapelet Transform compared with a number of standard benchmark classifiers; 1 Nearest Nieghbour with Euclidean Distance, 1 Nearest Neighbour with Dynamic Time Warping setting the windows size through cross-validation and Rotation Forest, results are shown in Table 4

6.4 Average Case Time Complexity Improvements

One of the benefits of using the binary transform is that it is easier to use the shapelet early abandon described in [3]. Early abandon is less useful when finding the best k shapelets than it is for finding the single best, but when it can be employed it can give real benefit. Figure 6 shows that on certain datasets, using the binary shapelet discovery means millions of fewer $sDist$ evaluations.

We assess the improvement from using Algorithm 6 by counting the number of point wise distance calculations required from using the standard approach, the alternative enumeration, and the state of art the enumeration in [2].

For the datasets used in our accuracy experiments, changing the order of enumeration reduces the number of calculations in the distance function by 76% on average. The improvement ranges from negligible (e.g. Lightning7 requires 99.3% of the calculations) to substantial (e.g. Adiac operations count is 63% of the standard approach). This highlights that the best shapelets may or may not be phase independent, but nothing is lost from changing the evaluation order and often substantial improvements are achieved. This is further highlighted in Fig. 7 where the worst dataset Synthetic control does not benefit from the alternate enumeration. For the average and the best case we see a reduction of approx. 15% fewer operations required. On the Olive Oil dataset we see a 99% reduction in the number of distance calculations required.

Full results, all the code and the datasets used can be downloaded from [18,19]. The set of experiments, and results are constantly being expanded and evaluated against the current state of the art.

We define the opCounts as the number of operations performed in the Euclidean distance function when comparing two series. This enables us to estimate the improvements of new techniques. We show in Table 5 the op counts in millions for 40 datasets. The balancing in some rare cases can increase the number of

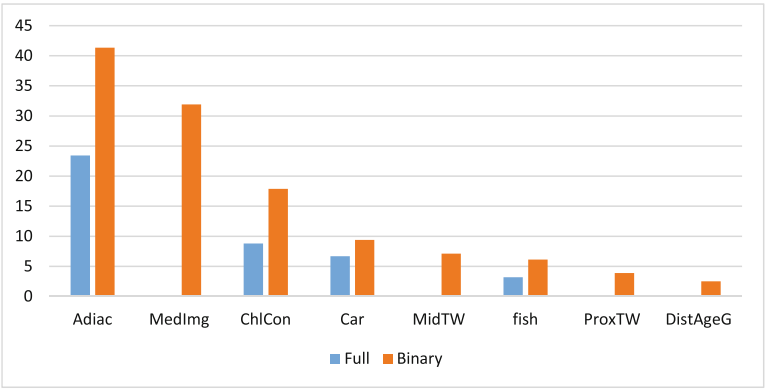

Fig. 6. Number of *sDist* measurements that were not required because of early abandon (in millions) for both full and binary shapelet discovery on seven datasets.

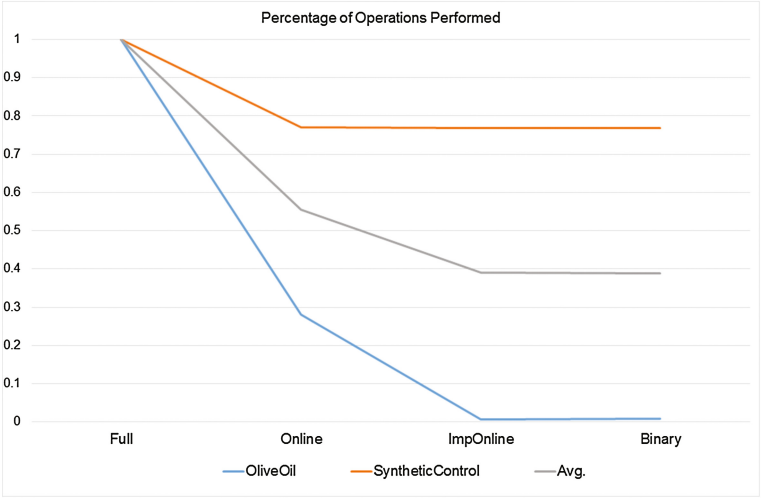

Fig. 7. Percentage of operations performed of Full search, compared with previous sDist optimisations, and our two new speed up techniques.

operations performed. Typically these datasets have a large number of a class swamping the shapelet set. The balancer has individual lists for each class, and thus the shapelet being compared to for early abandon entropy pruning is different for each class, and in some cases can mean more work is done. In these experiments we wanted to show the progression of op count reduction as we combined techniques. When comparing to the previous best early abandon technique, the combination of the new methods proposed has seen a reduction of on average 35% of the required operations, to find and evaluate the same shapelets.

Table 5. A table to show the opCounts in millions for the Full method, and the current and new techniques as a proportion of the calculations performed

Datasets	Full ST	Online	ImpOnline	Binary	New ST
ArrowHead	423196	0.496	0.307	0.298	0.301
Beef	3567380	0.357	0.234	0.266	0.285
BeetleFly	2192860	0.576	0.555	0.555	0.557
BirdChicken	2192860	0.475	0.398	0.398	0.403
CBF	20033	0.768	0.726	0.726	0.726
Coffee	427243	0.449	0.090	0.090	0.089
DiatomSizeReduction	286551	0.364	0.088	0.088	0.088
DistalPhalanxOutlineAgeGroup	569421	0.475	0.191	0.191	0.187
DistalPhalanxOutlineCorrect	1282270	0.489	0.218	0.218	0.218
DistalPhalanxTW	569421	0.474	0.192	0.192	0.178
ECG200	72759	0.568	0.448	0.448	0.448
ECG5000	8203050	0.608	0.512	0.512	0.487
ECGFiveDays	14826	0.580	0.326	0.326	0.326
FaceAll	7903390	0.717	0.692	0.692	0.692
FaceFour	698003	0.686	0.561	0.566	0.566
FacesUCR	1004840	0.740	0.676	0.676	0.670
GunPoint	105975	0.585	0.364	0.364	0.364
ItalyPowerDemand	136	0.529	0.397	0.397	0.397
Lightning7	4219010	0.732	0.712	0.706	0.696
MedicalImages	1202180	0.640	0.622	0.622	0.534
MiddlePhalanxOutlineAgeGroup	569421	0.456	0.156	0.156	0.153
MiddlePhalanxOutlineCorrect	1282270	0.458	0.161	0.161	0.161
MiddlePhalanxTW	566573	0.453	0.159	0.159	0.158
MoteStrain	1645	0.694	0.576	0.576	0.578
OliveOil	7706080	0.280	0.007	0.009	0.009
Plane	401574	0.567	0.455	0.455	0.455
ProximalPhalanxOutlineAgeGroup	569421	0.441	0.138	0.138	0.136
ProximalPhalanxOutlineCorrect	1282270	0.447	0.140	0.140	0.140
ProximalPhalanxTW	569421	0.443	0.138	0.138	0.137
ShapeletSim	1994760	0.690	0.673	0.673	0.679
SonyAIBORobotSurface1	799	0.579	0.397	0.397	0.397
SonyAIBORobotSurface2	1101	0.654	0.561	0.561	0.561
SwedishLeaf	5745210	0.550	0.393	0.393	0.389
Symbols	1266960	0.632	0.601	0.601	0.601
SyntheticControl	102522	0.771	0.768	0.768	0.768
ToeSegmentation1	776099	0.620	0.611	0.611	0.607
ToeSegmentation2	1469900	0.555	0.556	0.556	0.554
Trace	4785000	0.691	0.634	0.634	0.634
TwoLeadECG	1991	0.497	0.203	0.203	0.203
Wine	810711	0.383	0.023	0.023	0.023

7 Conclusion

Shapelets are useful for classifying time series where the between class variability can be detected by relatively short, phase independent subseries. They offer an alternative representation that is particularly appealing for problems with long series with recurring patterns. The downside to using shapelets is the time complexity. The heuristic techniques described in recent research [4,5] offer potential speed up (often at the cost of extra memory) but are essentially different algorithms that are only really analogous to shapelets described in the original research [1]. Our interest is in optimizing the original shapelet finding algorithm within the context of the shapelet transform. We describe incremental improvements to the shapelet transform specifically for multi-class problems. Searching for shapelets assessed on how well they find a single class is more intuitive, faster and becomes more accurate than the alternative as the number of classes increases. We demonstrated that the binary shapelet approach is significantly more accurate than other shapelet approaches and is significantly more accurate than conventional approaches to the TSC problem.

References

1. Ye, L., Keogh, E.: Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data Min. Knowl. Disc. **22**(1–2), 149–182 (2011)
2. Hills, J., Lines, J., Baranauskas, E., Mapp, J., Bagnall, A.: Classification of time series by shapelet transformation. Data Min. Knowl. Disc. **28**(4), 851–881 (2014)
3. Mueen, A., Keogh, E., Young, N.: Logical-shapelets: an expressive primitive for time series classification. In: Proceeding 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2011)
4. Rakthanmanon, T., Keogh, E.: Fast-shapelets: a fast algorithm for discovering robust time series shapelets. In: Proceeding 13th SIAM International Conference on Data Mining (SDM) (2013)
5. Grabocka, J., Schilling, N., Wistuba, M., Schmidt-Thieme, L.: Learning time-series shapelets. In: Proceeding 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2014)
6. Gordon, D., Hendler, D., Rokach, L.: Fast randomized model generation for shapelet-based time series classification. arXiv preprint arXiv:1209.5038 (2012)
7. Lines, J., Bagnall, A.: Alternative quality measures for time series shapelets. In: Yin, H., Costa, J.A.F., Barreto, G. (eds.) IDEAL 2012. LNCS, vol. 7435, pp. 475–483. Springer, Heidelberg (2012). doi:10.1007/978-3-642-32639-4_58
8. Hills, J.: Mining time-series data using discriminative subsequences. PhD thesis, School of Computing Sciences, University of East Anglia (2015)
9. Rakthanmanon, T., Bilson, J., Campana, L., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E.: Addressing big data time series: mining trillions of time series subsequences under dynamic time warping. ACM Trans. Knowl. Disc. Data, **7**(3) (2013)
10. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The WEKA data mining software: an update. SIGKDD Explor. **11**(1), 10–18 (2009)

11. Quinlan, J.R., et al.: Bagging, boosting, and c4.5. In: AAAI/IAAI, vol. 1, pp. 725–730 (1996)
12. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
13. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
14. Rodriguez, J.J., Kuncheva, L.I., Alonso, C.J.: Rotation forest: a new classifier ensemble method. IEEE Trans. Pattern Anal. Mach. Intell. **28**(10), 1619–1630 (2006)
15. Lines, J., Davis, L., Hills, J., Bagnall, A.: A shapelet transform for time series classification. In: Proceeding of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2012)
16. Lin, J., Keogh, E., Li, W., Lonardi, S.: Experiencing SAX: a novel symbolic representation of time series. Data Min. Knowl. Disc. **15**(2), 107–144 (2007)
17. Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The UCR time series classification archive (2015). http://www.cs.ucr.edu/~eamonn/time_series_data/
18. Bagnall, A., Lines, J., Bostrom, A., Keogh, E.: The UCR/UEA TSC archive. http://timeseriesclassification.com
19. Bagnall, A., Bostrom, A., Lines, J.: The UEA TSC codebase. https://bitbucket.org/TonyBagnall/time-series-classification