Mini-Project (ML for Time Series) - MVA 2022/2023

Sophia Chirrane sophia.chirrane@student-cs.fr
Lucas Saban lucas.saban@ensae.fr

April 30, 2023

1 Introduction and contributions

Time series classification is a task in machine learning that involves predicting the class label of a time series data point based on its temporal sequence of values.This task is challenging because time series data can exhibit complex patterns and variability, and may be affected by noise and other sources of uncertainty.

Shapelets are an important concept in time series classification. A shapelet is a short, representative subsequence of a longer time series that can capture important patterns and features of the data. Shapelets can be learned from the training data using various algorithms, and they can be used to build classifiers for different applications such as disease diagnosis, fraud detection, and activity recognition. By comparing the distance between a shapelet and each subsequence of a longer time series, we can determine whether that time series belongs to a particular class. The use of shapelets can improve the performance of time series classification by reducing the dimensionality of the data and capturing relevant features in a compact and interpretable way. While shapelets have shown promising results in time series classification tasks, their effectiveness is limited by the computational complexity and time required to learn and compare shapelets.

The studied article [3] introduces a new method to overcome this issue while preserving or even improving the accuracy of the classification. The implementation of the method is available. However, as it is in C++ we decided to reimplement it from scratch in Python in order to have a better understanding of the method and more flexibility to test it and improve it. The implementation can be found here. We first propose an analysis of the extracted shapelets and of the stability of the method, which was not presented in the article [3]. In order to improve the method, we noticed that to compare two time series or subsequence of time series (e.g. shapelets with shapelet and each subsequence of a longer time series) the introduced method uses the normalized Euclidean distance. We proposed to analyze the influence of this choice by adding the possibility to use other kind of distances, such as the Dynamic Time Wrapping distance (DTW). We also improved the implementation to include multi class entropy comparison. We shared the work as follows : Implementation of the first part of the method (lines 1 to 10) : Lucas Saban, Implementation of the second part of the method (lines 12 to 24) : Sophia Chirrane. Except for the implementation of the DTW, we built the algorithm ourselves, including computational optimization process (use of JAX, profiling of code snippets, acceleration tricks, multiprocessing, . . .)

2 Method

The paper "Fast Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets" [3] proposes a novel approach to address the computational complexity of learning and comparing shapelets in time series classification tasks. The approach is based on a two-phase algorithm that first learns candidate shapelets using a heuristic method and then prunes the candidates based on their quality and redundancy. By combining these techniques, the algorithm achieves a significant speedup compared to existing shapelet-based algorithms while maintaining a comparable or even better classification performance. The approach has been evaluated on various benchmark datasets and shown to outperform state-of-the-art methods in terms of both speed and accuracy. The pseudocode of the method is presented in Figure 6

The heuristic method in the first part of the proposed method is used to efficiently discover candidate shapelets from a given dataset. The method involves two main steps:

- First, from all possible subsequences of a given length, we extract the different SAX words possible. The Symbolic Aggregate approximation (SAX) technique [4] is used to represent the subsequence windows as symbolic sequences of fixed length. SAX is a symbolic time series representation method that converts a numerical time series into a sequence of symbols using a pre-defined alphabet and a sliding window approach. By using SAX symbols instead of raw numerical values, the heuristic method is able to reduce the dimensionality of the data and focus on the most discriminative features of the time series.This step is not costly, as we can leverage tensor operations and vectorization using JAX [2]. We can remark that a time series usually gives several sax words.
- The next step consists of computing a distinguishing score for each SAX word. This is done by randomly masking some parts of the words, and counting with how many other sax words it matches. By repeating this several times, we are able to build a collision table, from which we can extract distinguishing scores for each SAX word. Those with the highest distinguishing power are then remapped to their raw subsequence to form the shapelet candidates for the second part of the algorithm. This step is the computational bottleneck of our implementation, as there is no clear way to count collision efficiently. Amongst the difficulties there are for example the number of sax words per time series is not constant, which lead to broadcasting issues or the way of computing collision depends on the SAX word and its object of appurtenance, hence adding indexing complexity. The solution would be to use a compiled language like C/C++ that would leverage its speed to accelerate this part.

The heuristic method is designed to be fast and scalable by avoiding the exhaustive search over all possible subsequence windows, which is computationally expensive for high-dimension datasets. Indeed, by searching in the low-dimensionnal sax words space as a proxy for the raw subsequences spaces, the method computational overhead is reduced.

In the second part, the final shapelet is selected from the top-ranked shapelets of part one based on two criteria: information gain and separation gap. Information gain measures the reduction in entropy that results from splitting the dataset based on a given shapelet. It corresponds to the difference between the entropy of the original dataset and the weighted average of the entropies of the two subsets resulting from the split. On one hand, entropy is a measure of the randomness or unpredictability of a dataset, so a high information gain indicates that the shapelet is able to split the data into subsets that are more homogeneous with respect to their class labels. On the other hand, separation gap measures the distance between the two sub dataset resulting from a split.

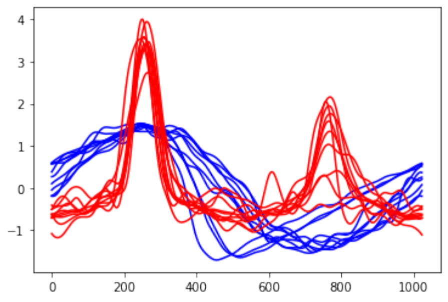
Overall, the information gain and separation gap criteria ensure that the selected shapelets are both informative and distinctive, meaning they can effectively separate the positive and negative classes and contribute to accurate classification.

The algorithm calculates the information gain and separation gap for each shapelet by splitting the dataset into two sub dataset. To find how to split the dataset given a shapelet, the method compute the minimal distance between each Time series of the dataset and the shapelet and then select the split with the higher information gain. The best shapelet, for a given length, is so the one with the highest information gain and separation gap.

3 Data

For the first part of the conducted experiments, we used the StarlightCurves dataset. This dataset was also used by the author of [3]. This dataset is a collection of time series data, specifically light curves of stars, obtained from the MACHO (MAssive Compact Halo Objects) project. The StarlightCurves dataset contains 9,999 light curves, each with 1,024 data points, and each representing the brightness of a star over time. The dataset is labeled with one of three classes: periodic, non-periodic, or eclipsing binary. The periodic class refers to stars that exhibit regular fluctuations in brightness, such as pulsating stars or stars with spots on their surface. The non-periodic class includes stars with irregular variations in brightness, such as variable stars that are undergoing a phase of instability. The eclipsing binary class includes stars that are in binary systems and whose light curves show periodic dips in brightness as one star passes in front of the other.There are several challenges associated with classifying the StarlightCurves dataset.

- The dataset is imbalanced, with the majority of examples belonging to the non-periodic class.
- The light curves in the dataset are noisy, with fluctuations in brightness that may not be directly related to the type of star being observed
- The light curves of stars in different classes can be very similar, making it difficult to distinguish between them based on shape alone.
- Each light curve contains 1,024 data points, resulting in high-dimensional data that can be difficult to work with.


Examples of the two most different class

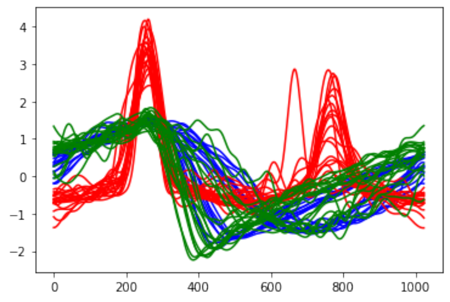
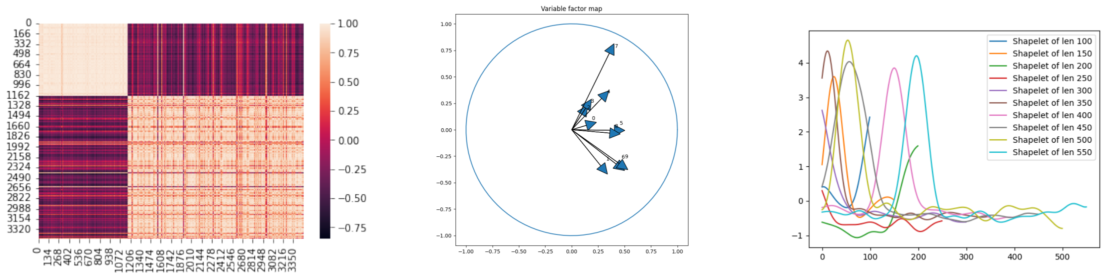

Samples of the 3 class of the dataset

Figure 1: Presentation of the StarlightCurves dataset

4 Results

Binary classification : We first conducted experiments to get a deep understanding of the behavior of the method while dealing with real data. First, we tested the algorithm on the StarlightCurves dataset. The first task we address was a binary classification task on the two most different classes as presented in 1. We first extract the 10 best shapelets of size 100 to 550 (50 by 50) using the whole train set data of each class. We used a cardinality of 4 and a dimensionality of 16. We then define our train set by considering the same one used for the shapelet extraction. For the test set, we used the 1177 examples of the first class and the 2305 of the second one. We transformed the train and test set by computing the minimum distance between the time series and each shapelet as defined in [3]. Thus, shapelet are considered as features that we then used to train a KNN classifier with one neighbor. By doing so, we achieve on the test set a mean accuracy of **0.998**. We also presented the samples' correlation matrix after the shapelet transformation. To have a better understanding of how the algorithm chooses the informative shapelet we also provide the correlation circle after PCA of the shapelets as feature and the chosen shapelets (see Figure 2).



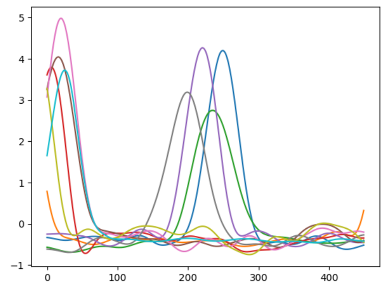
a b c

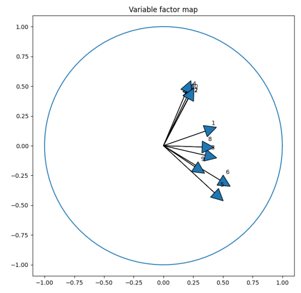
Figure 2: a = Correlation matrix of the test set,b=PCA of the shapelets, c = The selected shapelets

We observed first that the shapelet transformation of the data can pretty well separate the dataset into two class (a). Then, we see that some shapelets are really informative (7) and highly correlated (6,9) : they so share redundant information. This observation is confirmed by the shape of these shapelets (b) which are really close. As expected also the less informative shapelet (0,3) have a shape which does not seem useful for classification (flat shape or too small). Moreover, we can not say the longer, the more informative since the best shapelet is of length 450 but the one of length 150 also performs quite well.

To summarize, the method seems to have a strong generalization power and seems appropriate to reduce drastically the dimension of the light curves initially living in a high dimensional space.

Stability : As the method includes some random operations, we found it relevant to test the stability of the algorithm. To this end, we computed the 450 length best shapelet based on the same light curves samples used in the previous experiment. We used a cardinality of 4 and a dimensionality of 16. We repeated this experiment 10 times. As we can see in Figure 3, for a given set of parameters and data the best shapelet is not always the same and sometimes don't seem to share at all the same information, as we can see both on their visual aspect and their correlation. Therefore, the method is not very stable.

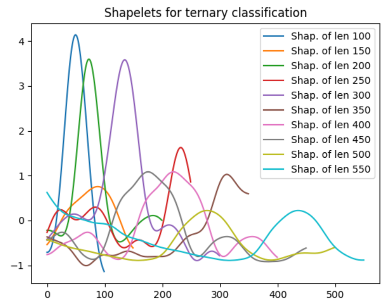

The 450 length best shapelets for ten repetitions



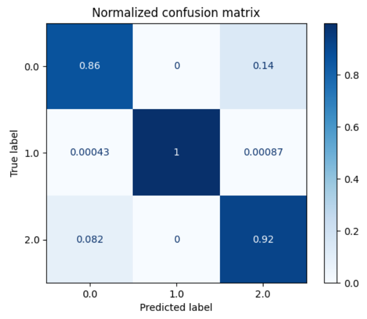
Correlation between these shapelets

Figure 3: Stability analysis

Ternary classification : We also decided to test the performance of the method on the 3 class classification task (see Figure 1). For this task, we extracted only 10 shapelets based on the whole train set. We then proceed the same as for the binary classification task. We achieve a mean accuracy of 0.93 which is similar to the one presented in [3]. As expected, class 0 and 2 more are difficult to distinguish (blue and green in Figure 1).



Extracted shapelets

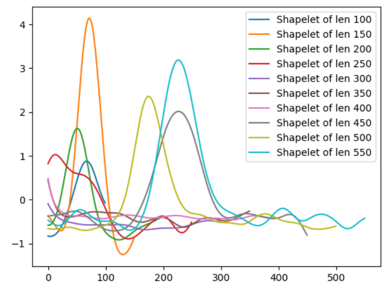


Confusion matrix of the test set

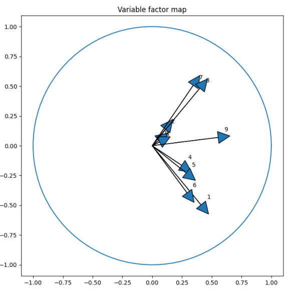
Figure 4: Accuracy of 0.93 when these ten shapelets are selected with the associated confusion matrix

Impact of the chosen distance: One of the fundamental block of the method is the chosen distance to both find the best shapelet and then transform the data to use a classifier. In the reference article [3] they used the normalized Euclidean distance. For the final conducted experiment, we decided to compare the performance on the binary classification task of the method with respect to different distances. For that, we compare the normalized Euclidean distance and the Dynamic Time Warping distance (DTW) [1]. Results obtained with DTW are presented in Figure 5. The achieved accuracy is 0.997 which is just a little less than when the normalized Euclidean distance is used. However, it takes 10 hours to get similar results as with the normalized Euclidean distance, while the latter takes only 10 minutes.

Sample efficiency: finally, it is worth mentioning that the method can be said sample efficient, as we obtained a test accuracy of 0.929 by only using the 10 first train sample for binary classification. Therefore, this method could be made even faster by compromising on accuracy points.



Extracted shapelets



PCA on extracted shapelets

Figure 5: Accuracy of 0.997 while using the DTW as distance

References

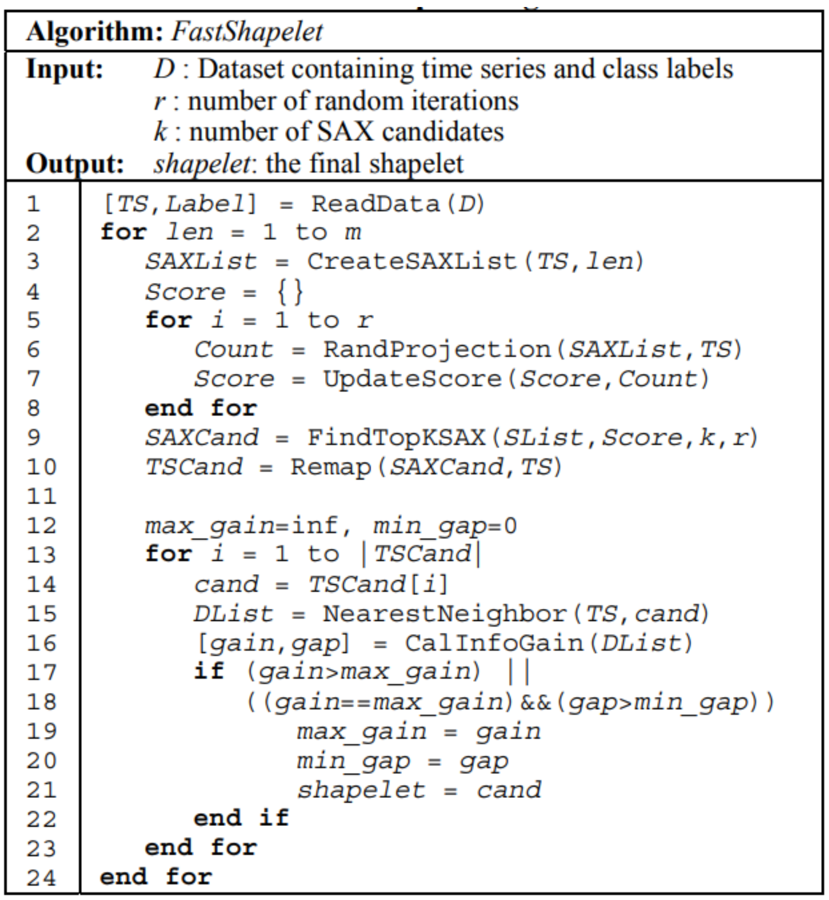
[1] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD Workshop*, 1994.

[2] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[3] Jon Hills, Jason Lines, Edgar Baranauskas, Jeremy Mapp, and Anthony Bagnall. Fast shapelets: A scalable algorithm for discovering time series shapelets. *Data Mining and Knowledge Discovery*, 28(3):872–909, 2014.

[4] Jessica Ke Yi Lin and Yuan Li. Experiencing sax: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, 2007.

5 Appendices



Algorithm: *FastShapelet*

Input: *D* : Dataset containing time series and class labels
r : number of random iterations
k : number of SAX candidates

Output: *shapelet*: the final shapelet

```
[TS,Label] = ReadData(D)
for len = 1 to m
   SAXList = CreateSAXList(TS,len)
   Score = {}
   for i = 1 to r
      Count = RandProjection(SAXList,TS)
      Score = UpdateScore(Score,Count)
   end for
   SAXCand = FindTopKSAX(SList,Score,k,r)
   TSCand = Remap(SAXCand,TS)

   max_gain=inf, min_gap=0
   for i = 1 to |TSCand|
      cand = TSCand[i]
      DList = NearestNeighbor(TS,cand)
      [gain,gap] = CalInfoGain(DList)
      if (gain>max_gain) ||
         ((gain==max_gain)&&(gap>min_gap))
            max_gain = gain
            min_gap = gap
            shapelet = cand
      end if
   end for
end for
```

Figure 6: Algorithm of the fast shapelets method