Master research Internship

Bibliographic report

Indexing Time Series

Domain: Machine learning, Time series, Retrieval task, Classification task

Author:
Arnaud LODS

Supervisor:
Simon MALINOWSKI
Romain TAVERNARD
Laurent AMSALEG
Linkmedia

Abstract

In this report we are interested to study papers related to the time series domain. More specifically we intend to perform retrieval tasks on datasets which contains million of times series that are themselves both in high dimension and with multiple variables. We would like to see if existing methods could be used to fit our needs. Thus we will present works depicted in different papers to manipulate those time series and we will see how researchers have been using them to either classify or retrieve time series on a large scale. Eventually, as we might not find methods appropriate we will see how we could extend the state of the art.

Contents

Introduction

Time Series (TS) are one of the most important data types as they represent what can be collected in many fields such as medicine, speech, finance, information retrieval, computer vision etc. This type of data is different from the others as its values are structured over time, thus creating the need to get special methods to manipulate them. Among the several operations carried out on these data, the retrieval task is one of the most frequent. Basically given a TS we try to find in a dataset one or several TS that are likely to be relevant to the query, in other words we want to retrieve TS that are highly similar to our query.

The works of the team Linkmedia focuses on the analysis of multimedia data (either audio or video). Thus some specificities stand out while working on this data :

1. We have to work with large dataset (million of TS)
2. Each TS itself can hold a lot of data, thus each object is represented in high dimension
3. Each TS possess several variables. These TS are called multivariate time series (MTS)

Early works focusing on TS tried to manipulate TS in a way that would enable us to make accurate similarity search. At that time these methods could be applied on the entirety of databases as we didn't need real time results and the amount of data was handy. However in recent years the collected data increased exponentially and it has become impossible to make the calculations on whole databases. Thus recent works focused on improving the manipulation of TS with efficient computation time in order to carry out either retrieval or classification tasks on large scale. These works have proved to hold convincing results, though with circumstances different than ours. The goal of this bibliography is to study the works produced to assess them and define whether or not we could use them with our issues.

Through this study we will not only consider papers related to the retrieval task but also works that have been produced as a way to solve classification tasks. In contrary to our initial task, a classification task intends to define the belonging class of a query. In [17] the authors presented the idea that even though classification and retrieval are two different tasks they are still equivalent. Therefore, we might be able to adapt the mechanisms from one task to the other. Thus we will also study classification works in order to see whether or not their methods could satisfy our needs.

To prepare this study we will begin in Section 1 with the description of methods related to similarity measures between TS. In Section 2 we will study papers that focus on the retrieval task using methods presented earlier. We will then present in the conclusion the ideas that raised from our observations.

1 Similarity measures for time series

This section will be divided in two parts. In a first part we will present methods that use the raw data to compute the similarity between two TS. Then methods that transform these TS to do the computation are introduced in the second part.

For the rest of this study, we define both TS and MTS as follow. A TS X is a sequence of values $X = \{x_1, x_2, ..., x_n\}$ which are associated with timestamps $\{t_1, t_2, ..., t_n\}$ with $t_i = t_{i-1} + \tau$ where τ is the sampling interval and n is the number of points in our TS. A MTS is a TS that contains m variables noted y. We represent them with a vector Y_t where the i^{th} row of Y_t is y_it. Thus for any time t, $Y_t = (y_{1t}, y_{2t}, ..., y_{3t})$. A TS that has only one variable is also called an univariate time series (UTS).

1.1 Raw time series

1.1.1 Euclidean distance

The common way to measure the distance between two objects is to use the Euclidean distance (ED). This is a method that can be used with almost every tope of data that has real values. Considering two TS $X = \{(x_1; t_1), (x_2; t_2), ..., (x_n; t_n)\}$ and $Y = \{(y_1; t_1), (y_2; t_2), ..., (y_n; t_n)\}$ of size n, the Euclidean distance is computed by adding the squared difference between each value of the TS at the same timestamp :

$$ED(X,Y) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$

With multivariate series, we would do the same computation for each of its variable and take its sum. It has been proved that this method can compute a real distance between two TS and its computational complexity is rather low ($O(n)$). Even though this is a basic method that has not been adapted to TS, several recent works either use the ED with their method or compare their results to the results you could get with ED alone.

However the main problem that we confront while using ED to differentiate TS is that it does not evaluate shift in times. If we try to compare two TS slightly shifted over the same timestamps this algorithm will not recognize the high similarity. Though it is still able to differentiate between pretty similar and far TS, and can be used as a pretty cheap pruning, it has been recognized that ED does not yield the best results.

1.1.2 Dynamic Time Warping

Sakoe proposes in [13] the Dynamic Time Warping (DTW), which was designed to correct the first problem of ED and thus can mesure the similarity between two sequences that are of different sizes and may have temporal shifts. The idea is to calculate an optimal match between two TS. To compute the DTW we construct a matrix W where $W(i, j)$ denotes the squared distance between the i^{th} element of X and j^{th} element of Y. To find the best match we retrieve a path through this matrix that minimizes the total cumulative distance between them. The optimal path is the one minimizing the cost :

$$DTW(X,Y) = min\sqrt{\sum_{k=1}^{K} w_k}$$

where w_k is the k-th element of a warping path P, which is an adjacent set of matrix elements. Usually the warping path is computed using dynamic programming to evaluate :

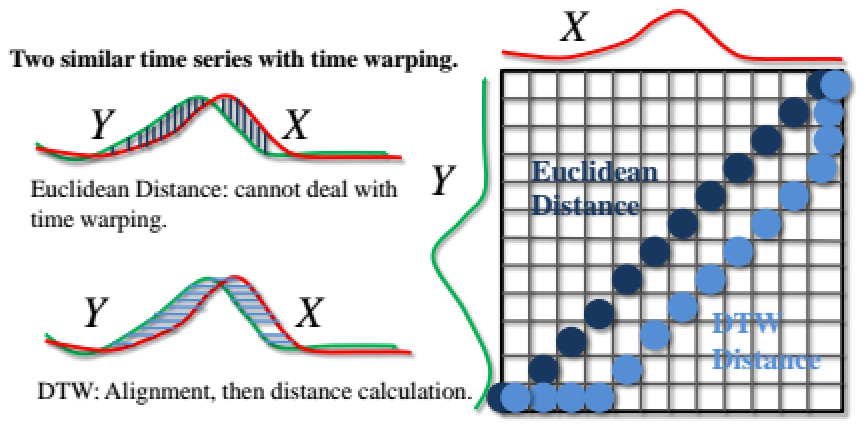
Two similar time series with time warping.
Y
X
Euclidean Distance: cannot deal with time warping.
Y
X
DTW: Alignment, then distance calculation.
X
Y
Euclidean Distance
DTW Distance

Figure 1: Left-Top: ED between X and Y. Left-Bottom: DTW between X and Y. Right: Matrix with the squared distance between each elements of X and Y, by default the ED will sum theses values for the same timestamp while DTW find an other path that minimizes the cost

$$\gamma(i,j) = d(x_i, y_j) + min\{\gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1)\}$$

where $d(x_i, y_j)$ is the distance found in the current cell and $\gamma(i,j)$ is the cumulative distance of $d(i,j)$ and the minimum cumulative distances from the adjacent matrix elements.

When dealing with MTS, we need to compute the DTW for every sequence and then get the sum of them. While this approach has been accepted as the best way to differentiate two TS, it does require a computational cost of $O(n^2)$, which is much more expensive to compute than ED. Even though a lower bound with a computation cost of $O(n)$ has been developed (and we still need to compute the full DTW on several candidates), this method can't be used on too many TS. As this method suffers from the same problem as the ED, recent works focused on working on other methods to prune the dataset and apply DTW on a subgroup of the dataset.

1.1.3 Global Alignment Kernel

The idea behind kernel methods is to transform the input data into a higher dimension space that will then be convenient to use (mostly with classifier like SVM). In [5] the authors try to suggest the use of a specific kernel method for TS as the existent ones are not adapted to this data representation. Using a standard kernel method was a no-go as it couldn't cope with TS of different length and it would not be able to capture the dependencies between two consecutive values.

Here the authors take into account all the possible alignments between two TS instead of just using the best alignment. Thus the score values for every possible alignment computed for the TS is used to calculate the kernel value. Used on small dataset of a few thousands of TS for a classification task the results were convincing with a misclassification of 5.4% (9 classes). The problem here is that the computation for this method is as high as computing the DTW (though it has been improved in [4]) thus we can't use this on a large dataset. Moreover if we were to use this method on a MTS we would need to calculate the alignment for each TS for each of its variable increasing as much the computation time as the MTS has variables.

1.2 Transform based

As seen in the last subsection, there exist methods used on the raw values of TS to evaluate the distance between two TS. However it is inefficient to apply these methods on a huge number of TS, thus other works focused on the transformation of TS. The first focus is to reduce the dimensionality of the TS to reduce the needed computation. The second focus is inspired from the computer vision tasks, where we can carry out efficient classification tasks on images by describing them in an other dimension. The idea is to find a better description of the TS that can be easily manipulated.

1.2.1 Symbolic Aggregate approXimation

Lin [7] shows the problem of working on TS in a streaming environment as it is and try to provides a new representation that would allow to reduce as much as possible the dimensionality of TS while still keeping the original form of the TS. The idea is to discretize the data into symbolic strings while guaranteeing that the distance between two TS representations is a lower bound of the original distance. Both challenges are discussed and solved in this work.

First each TS is normalized to have a mean of zero. Their goal is to have the same probability for each symbol to appear and they made the observation that the distribution of their data follows a Gaussian distribution. Then they use the Piecewise Aggregate Approximation (PAA) to reduce the dimensionality of the TS. Simply said, to reduce the dimensionality of a TS from n dimensions to m dimensions, the data is divided into m equal sized frames and the mean value of the values falling within this frame is used to represent the frame. Formally a frame $\overline{x_i}$ is calculated with :

$$\overline{x_i} = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}j} x_j$$

When the TS are transformed using the PAA, the SAX discretization is applied. In this case their data follow a Gaussian distribution, but other kind of distribution could be applied depending on the data. The breakpoints $B = \beta_1, \beta_2, ..., \beta_{a-1}$ are chosen such that the area under a $N(0,1)$ Gaussian curve from β_i to $\beta_{i+1} = \frac{1}{a}$, a being the number of areas we want to produce. Points produced by PAA that are bellow the smallest breakpoint are mapped on the symbol "a", then the values that are greater than this breakpoint but smaller than the next are mapped on the symbol "b" etc.. See Figure 2.

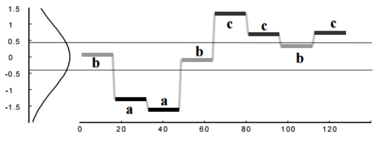

Figure 2: Example of SAX

Eventually once the TS have been transformed following this algorithm, we can use a specific similarity measure to evaluate the distance between two TS, which is based on the ED in this paper.. To calculate the distance between two points, that are now discretized, we look into a lookup table (that has the value of the distance between 'a' and 'b', 'a' and 'c'..). Thus the number of calculus are greatly reduced and we get a time computation complexity of $O(\frac{n}{w})$ to get a lower bound of the normal ED.

$$Dist\_Sax(\widehat{X}, \widehat{Y}) = \sqrt{\sum_{i=0}^{w}(dist(\widehat{x}_i, \widehat{y}_i))^2}$$

This paper has been approved by the community working on TS. Though one of the drawbacks of this method is that we have to set two parameters that might change the accuracy depending on our dataset. The more we reduce the dimensionality of the data, the less the computation time but then the pruning would be too low and we'd still have to use the original distance measure on a large part of the dataset. However this representation can be used in a lot of data mining tasks, for example as a basis for pattern extraction [15]. The experiments presented show that SAX holds pretty good results compared to other methods.

Thus using this method like that we could solve our first two problems sinceit has the ability to reduce by a large margin the dimensionality of the data. However this method is not adapted to MTS. First the authors make the assumption that all the values follow a Gaussian distribution but we can't suppose the same thing for MTS because the multiple variables of a MTS are likely to be correlated. Moreover there is two ways to adapt this method for MTS. The first is to use the algorithm on each variable. Thus we would have a sequence for each variable as the output, and that is still tough to manipulate. The second is to somehow, for each timestamp, take all the symbols generated and create a symbol for this combination of symbols to get only one sequence as an output. But in this case the result would be really sensitive to noise and it would be really hard to find two MTS highly similar with the representation.

1.2.2 Set representation

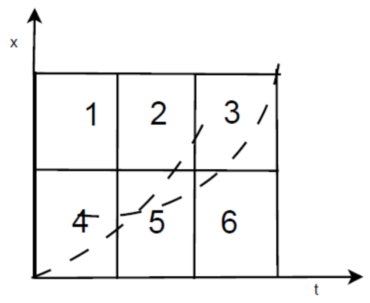

Figure 3: Example of Set representation

An other work by Peng [11] presented a method which is very similar to SAX. Instead of mapping "frames" extracted from PAA depending on a horizontal separation; here values are going to be mapped depending on the cells they belong to. Given a set of time series, they have a $bound(D)$ such that $bound(D) = [(t_{min}, x_{min}), (t_{max}, x_{max})]$ is the minimum bounding rectangle in the plane that covers the set of time series. This rectangle is then divided into equal cells. Then each point $p_i = (t_i, x_i)$ of the time series belong to a cell if $down_x < x_i < up_x$ and $left_t < t_i < right_t$ where the coordinates of the left-down and right-up of the cell are $(left_t, down_x)$ and $(right_t, up_x)$ respectively.

Each cell is assigned an unique ID, and then the algorithm computes the cells' IDs of all points. As this method doesn't use a mean for every vertical frame, the output will most likely have higher dimension than those of SAX, but it will still reduce the dimensionality. However it is not possible to determine the size of a transformed TS. Moreover, this should result in higher computation time but the output is supposed to give better lower bound results for the distance.

One more interesting thing about this paper is that they give a convincing idea to extend this method for MTS. For a MTS with two variables, instead of constructing rectangles we can construct cubes with one more parameter, and estimate in which cube each data point is present with these parameters. While the computation time would increase drastically with a higher number of variables, we could still be able to transform a MTS into a UTS thus resulting in easier distance measure and comparison. The problem with this is that manipulating hypercube would give out bad results as we either use "large" hypercube that will take too much datapoint and thus that will group MTS that have nothing to do in common, or use "small" hypercube that won't be able to group similar data because of the noise. Thus we might observe two similar MTS that have some noise on one (or more) of their variables, resulting in the datapoint going into another hypercube.

1.2.3 Piecewise Linear Approximation

We have seen earlier the use of PAA for the SAX transformation that is designed to reduce the dimensionality of TS. The Piecewise Linear Approximation (PLA) in an other method proposed and that has been used for a lot of papers as a basis. As the name of this method suggests it, the idea is to approximate the TS using linear segments $s'_t = a.t + b$. a and b are determined such that the reconstruction error of the TS is minimized. This error corresponds to the distance between the approximated and the actual time series. a and b can be obtained with :

$$a = \frac{12\sum_{i=1}^{m}(t - \frac{n+1}{2})x_i}{n(n+1)(n-1)} \qquad \text{and} \qquad b = \frac{6\sum_{i=1}^{m}(t - \frac{2n+1}{3})x_i}{n(1-n)}$$

where x_i is the value at the timestamp i. The lines segments that result from these parameters can well approximate the TS as they are selected as to achieve the minimum reconstruction error. As we do for SAX, we divide the TS into m segments of equal length and we calculate both a and b for each segment. The new TS X_{PLA} is now defined as follow : $X_{PLA} = a_1, b_1; a_2, b_2; ...; a_m, b_m$. With this transformed TS, it is now needed to define a distance measure that can give a lower bound of the ED.

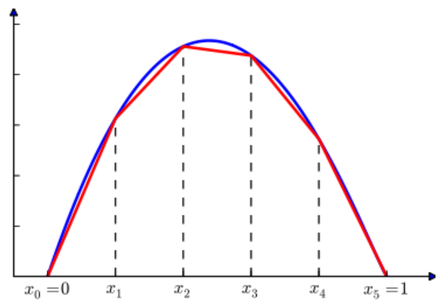

Figure 4: Example of PLA approximation

Chen [3] proposed a lower bound distance function based on the data we have presented above that calculate the sum of their distance between each segment :

$$dist_{PLA}(S,Q) = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{l}(a_{i,j} + b_i)^2}$$

where l is the length of each segment, $a_{Si} = a_{Qi} - a_{2i}$ and $b_{Si} = b_{Qi} - b_{2i}$. This function is then used to prune the dataset. As it is based on the same idea as PAA, it gives out pretty good results but there is still the need to define the parameter. Same as the methods above, with MTS with can either work with multiple sequences or try to have a "mean" segment calculated from all the segments for each position, but in this case we most likely would match MTS that have nothing in common.

1.2.4 Shapelet transformation

The idea behind this method, presented by Lines [8] among other, is to generate "shapelets" from TS that are small subsequences that can occur at any point in a series. Then once shapelets are extracted from a dataset of TS, we can compute the distance between those shapelets and a query. Thus depending on the size of the extracted shapelets we can reduce the dimensionality of the TS in our database by a large margin.

We define S, a subsequence of length l of a TS X of length n, where $l < n$. A TS contains up to $(n - l) + 1$ distinct subsequences of length l. One of the problems is that the generation of all possible shapelets, from a minimal to a maximal defined length, take up a lot of time and several papers work on pruning this step. When the candidates shapelets are generated, several methods exist to extract the shapelets that best represent the TS. For example, with a classification task in mind, the distance between each shapelet of a TS and the TS of the same class are computed, and the shapelets with the minimal distance are kept. The distance is usually calculated with the ED as we want to compare a certain shapelet with every subsequence of the same length of an other TS and take the minimal value. It is also possible to extract random shapelets and use them directly. When the shapelets are generated, the distance between a query and each shapelet is computed in a list $[d_1, d_2, ..., d_n]$ with d_i being the distance between the query Q and the shapelet i. Thus the similarity is based on these distances.

One of the main problem of this approach is the shapelets' extraction that would take too much time on a large dataset even though it is an offline work that can be done only once. Moreover we can't extract too many shapelets as we need to stock them somewhere and it would be unreasonable that the shapelets take more space than the original dataset. Enventually there is still the problem with our MTS as we would need to extract shapelets for every variable. However as shapelets represent vectors in high dimension being able to extract a single shapelet that would represent multiple variables could give out interesting results.

1.2.5 Bag of words

The bag of words (BoW) or bag of patterns (BoP) is an approach that has been widely used in computer vision works. The idea is to represent documents using a histogram of word occurrences and, because of its simplicity and performances with images, there has been several papers that focused on extracting words from TS. Intuitively a word is a local feature that is extracted from a TS and that is supposed to represent the TS. For example one of the way to do so is to transform a TS using the SAX algorithm, then the n-grams of this TS are extracted to form a BoP [15]. As a reminder a n-gram represents all the combination of juxtaposed values of length n. For example for $X = [A, D, C, D, B]$ the $3 - grams$ are $[(A, D, C), (D, C, D), (C, D, B)]$.

This approach is mostly used for classification tasks, as TS from a same class are supposed to have highly similar histograms. An interesting thing to note is that it is quite intuitive to use this representation for MTS. Indeed we just need to extract one histogram by variable for a MTS, then compute all of its histograms to get a "global" histogram that represents this TS.

However one of the drawbacks from the BoW approach is that these words represent a fixed interval extracted from the TS. Thus we lose a lot of information about the time. Moreover it would be hard to keep the correlation between the different variables as the time information is partly lost.

2 Retrieval of time series

We've presented in the last section several methods that might be used as a basis for data mining tasks. Indeed it would not be possible to use these methods on whole datasets everytime we receive a query thus we will now present in this section works that focus on reducing the complexity of the retrieval task while keeping good accuracy.

Unless we specify it in the next subsections, the data is first normalized using the z-normalization that is used in most works, given by the equation $z = \frac{x - \mu}{\sigma}$ where x is the current value, μ is the mean of the TS and σ is the standard deviation of the TS.

We want to point out that there might be times where we don't want the data to be normalized (for example with temperature data we might not want to match a low temperature reading with a high one that has somehow the same variation and thus the same sequence once normalized) but we are usually looking for matching patterns than matching values, thus this is why most works normalize the data before processing it.

2.1 UCR suite

The first work presented by Rakhtanmanon [12] tried to find a workaround to the computation time of DTW. DTW has been proved to be really efficient in most cases to evaluate the similarity between two TS but its complexity is too high to use it on a large scale. This is why the authors propose here the UCR suite, which is a compound of four ideas to calculate efficient lower bound of the DTW to prune most of the candidates with a really fast computation time.

Early Abandoning Z-normalization : In this case we are treating TS in a dataset that have a size larger than our query. Thus the goal is to calculate the distance between all subsequences of a candidate; these subsequences being of the query's length. Usually we would take the first subsequence, normalize it, process it then take the second subsequence, normalize it, process it, etc.. It is pretty inefficient thus their first proposal is to keep in memory two sums while processing the query : the sum of the values from the start and the sum of the values from the start minus the n last values (n being the query's length). Thus we only need to go through each candidate once while performing the other step to get the normalization of each subsequence.

Reordering Early Abandoning : They note that the number of computation we need to do before exceeding the best-so-far value can vary according to the values we are processing. If we always start from the first values of the TS we might make more computation than starting from an other value. The idea is to start the computation where the absolute values of the query are the highest because this is where we might observe the larger difference. Indeed as all the data is normalized; the mean is zero and we observe that, on average, the highest values (be it in positive or negative) will have the largest contributions to the distance measure.

Reversing the Query/Data Role in LB$_{\text{-}Kheogh}$: LB$_{\text{-}Kheogh}$ is one of the distance measure that try to compute a lower bound for the DTW. Simply said an envelope is built around the query and the ED is calculated between the candidate and this envelope. This method enables to get a fast to compute lower bound, which we call here LB$_{\text{-}Kheogh}EQ$. This method being much faster to compute than DTW, they decided to also use the same method but reversed. If the first one didn't prune the candidate then before using the full DTW the candidate's envelope is built to compute the distance between the query and the candidate's envelope. This lower bound is called LB$_{\text{-}Kheogh}EC$.

Cascading lower bounds : Eventually the main contribution of the UCR suite is the use of lower bounds in cascade to prune a lot of TS early on with a low computation time. First the $LB_{\text{-}Kim}FL$, which compare the first and the last values of the query and the candidate, is used. This is a very weak lower bound but it allows to prune a lot of candidates. Then if the candidate has not been eliminated, we compute the $LB_{\text{-}Kheogh}EQ$. Whenever the best-so-far value is exceeded, we abandon this candidate and take the next one. If the lower bound has been completed without exceeding the best-so-far, they reverse the query and the data (as seen above) and compute the $LB_{\text{-}Kheogh}EC$ with the same idea. Finally if this does not prune the candidate, then the full DTW is computed.

Using these methods allow to drastically reduce the computation time for each query. Even though the computation is made on the whole database, a large part of it is pruned with a computation time between $O(1)$ and $O(n)$ making this really fast. However there are a few problems with their approach. In this paper, they present their results for a similarity research to find the best match to the query. If we wanted to get more than one match in the database, we could expect that there would be less early abandoning than when we are only seeking the most similar TS. With an increase of the number of requested matches, we wouldn't be able to prune as much as depicted in this paper and the performances would worsen. Moreover, this work is presented with UTS. We could extend their algorithm on MTS but we would need to perform their computation for each variable contained in a MTS, thus increasing many times the computation time. Furthermore when we use the lower bound LB_{Kheogh} we create an envelope of the TS. This means that with MTS we are going to make an envelope for each variable of our MTS, resulting in a less efficient lower bound as it is computed over multiple dimensions.

2.2 indexable SAX

In this paper, Shieh [16] focused on the use of the SAX representation to produce an efficient way of indexing terabytes of TS. Instead of using an alphabet as symbols this time each symbol is computed as a binary string with their cardinality : $iSAX(X, n, c) = \{n_1^c, n_2^c, n_3^c, n_4^c\})$ where n represents the number of vertical cuts and c the cardinality - meaning that with a cardinality of 8 we can go up to 3 digits, thus there are two horizontal cuts. Using the same lookup table as SAX we can calculate the lower bound distance between two iSAX words.

Now for the indexing, a tree is constructed with a root node, several internal nodes and terminal nodes. A terminal nodes is a pointer to an index file on the disk which contains the raw TS. These TS are represented by the terminal node's representative iSAX word. Basically if a terminal node is identified by the iSAX word $S = \{01, 01, 11\}$ then all the TS contained in this file have that same word. A maximum number of TS that a file can contain is defined beforehand as to not have files that contain too many TS. When we try to insert a TS in a file that's already full, we split the node to add additional information. With our example above we might add a

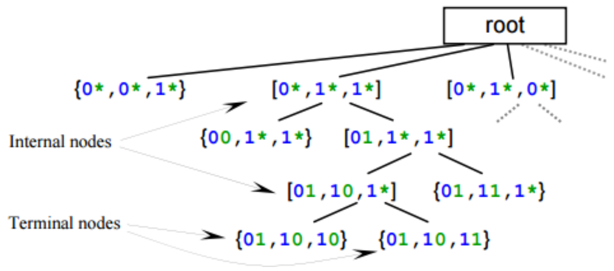

Figure 5: Example of an iSAX tree

digit to the first symbol and create as many nodes as there are TS with different values with the new cardinality. Then the old terminal node become an internal node whose purpose is to split the space with the promotion of cardinal values that were necessary to separate the TS. The root node has the same function as an internal node.

When this tree is created and a query is received it is possible to do two different searches. The approximate search is based on the fact that two similar TS will be represented by the same node. Thus the query is transformed into an iSAX word and the tree is scanned. If it finds a terminal node with the exact same word, the pointed file is returned. Otherwise it takes the first terminal node pointed by the last internal node with all matching values. For the exact search, iSAX will first make an approximate research to get a best-so-far (BSF) result. With this result the nodes will be examined to see whose distance is potentially less than the BSF, starting from the root node. If the distance to the node is less than the BSF answer, then the node's descendants are inserted into a priority queue. If the node is a terminal node then the file is fetched and the distances are computed for each entry and the BSF is updated accordingly. Eventually when there is no more node in the list, the search is done.

For a few million of TS; the approximate search allows to retrieve in 91.5% of the case one of the top 100 of the true nearest neighbor and in 14% of the case the true nearest neighbor. Considering that the approximate search needs only one access to the disk the results are convincing. As for the exact search for 100 million TS the algorithm took an average of 90 minutes while a sequential scan needs 1800 minutes. However as this method uses SAX as its base it suffers from the same setback we indicated in the Section 1.

2.3 Set based indexing and pruning

The method presented earlier [11] didn't focus solely on developing a new representation for TS but the authors also propose three different algorithms that can be used for retrieval tasks. Theses algorithms are an improved version of a naive Jaccard similarity (JS) over the query's cells and every candidate's cells. In our case the JS corresponds to the number of matching cells over the number total of unique cells that both TS contain.

Index based : For this algorithm an inverted list is used. Each entry of this list has a pair $(c, X*)$ where c represents a cell's id, and $X*$ the set of TS containing c. Thus it is easy to compute the number of intersections with each TS. Then we compute the JS between our query and each TS that has matches.

Pruning based : Here the JS is computed on an "approximated" representation of the TS. We reduce even more the dimensionality of the TS by reducing the number of cells. We browse the dataset and if a candidate is not pruned by the value calculated with the approximated TS, the JS is calculated on the original set of cells.

Approximate : This algorithm is similar to the last one, but this time instead of dividing once the TS in a lower dimension, we will move from the lowest dimension (2x2) and increase it until we reach the original scale. Then for each step we compute the JS and keep the candidates with the maximal similarity, while discarding the others.

The experiments shows promising results in term of computation time, the dimension reduction allows for faster retrieval of the nearest candidates in the database. However the results presented are below the state of the art. The reason is that with a small number of cells, TS far away are considered similar and with a high number of cells, similar TS tend to not match. We were particularly interested in this method because it could offer an interesting way to manage MTS, but these results would get even worse if we use hypercubes instead of sets. Indeed we would have few hypercubes thus a lot of different TS would be considered the same and with a high number of hypercubes we would not have any matches. When we see that this method doesn't hold the results we could expect in 2D, it would become worst in higher dimension.

2.4 indexable PLA

Using as a starting point the data that we get from PLA, Chen [3] introduced a way to use this approximation to index TS. As a recall once transformed, a TS is a list of tuples $a_1, b_1; ...; a_n, b_n$ that are the coefficients associated to each segment representing the TS. This PLA representation is then inserted into a R-tree, which is a structure that tries to group nearby objects with their minimum bounding rectangle (MBR). Here each entry of nodes in the R-tree consists of a MBR that contains both the reduced data and a pointer pointing to its corresponding subtree.

Though a lot of work has been made as to find an efficient way to compute the lower bound, the experiments have been made on databases of a few thousands TS. If we consider the efficiency of the R-tree when the number of data increases as well as the dimensionality of the data, we can't expect the results to be consistent. Moreover it suffers from the same setback as SAX and can't manage MTS.

2.5 SSH

An other recent work which proposes a new way to extract information from TS is presented in [9]. Their idea is to generate a weighted set of n-grams from TS and use these as hash value.

The main point of this method is to find matching n-grams between TS. Thus before extracting them we discretize the value as finding matching n-grams with continuous real values would be near impossible. A randomly generated gaussian filter of size m slides over the TS with a step size δ. For each slide the filter is multiplied with the current subsequence of the TS and we store a bit that indicate the sign of the output.

As we now have a representation of small subsequences of the TS, we can use them as a key for the hash. The idea is that if two TS share a common n-gram then we can expect that those two TS match. Thus for each TS generated we take the n-grams (n being defined as a parameter) and we create a set that take as the key the n-grams and as the value the number of occurrences of these n-grams.

Eventually as we want to capture the similarity between the sets, the weighted minwise hashing is used to index these weighted sets. When a query is received, it is transformed following the same steps and we keep the sets that are similar to our query. The most likely candidates are returned, and then the UCR suite is used on them to speed up the pruning of theses candidates; the full DTW being computed on the final candidates.

One of the problem of this algorithm is that it needs to define three parameters that can increase the computation time by a large margin and also vary the results. In this experiment they have a few millions data points and the results presented look convincing enough. They can prune 95% of the dataset with TS of size 2048, the method is faster than the URC suite (up to 20 times) and when searching for the top-10 most similar TS, they get a 100% accuracy. The problem is that the accuracy varies by a large margin depending on these parameters (we can get as low as 20%) and we can expect that on datasets that are heterogeneous we might not be able to get a good accuracy. Moreover it would take too much time to identify the best parameters on datasets that contain million of TS. As n-grams are extracted without keeping the information about their timestamp, we also lose some information here and with MTS we could not keep the information of correlated variables. Moreover working on MTS would increase by a lot the computation time to transform these TS.

3 Conclusion

As a reminder, our goal is to be able to manage retrieval tasks on large datasets of time series in high dimension, themselves being multivariate. This is a real challenge as the space we are dealing with is on a scale that makes it impossible to treat the raw data for real life tasks. We have seen that the need to manage them in different spaces is a must but though there is ideas that could be improved or used to cope our problematics; all the current state of the works presented here are not able to deal with at least one of our goal. Thereby there is a possibility that the ideas presented for theses works could be improved or altered with further investigation.

We would like to focus on other solution and the second task that researchers have worked on is the classification task. In the introduction we raised an idea developed in [17] that methods used with classification in mind can also be used for retrieval tasks and vice versa. Thus we extended the study to classification works to see whether theses papers would use methods that we could transpose to our task. Several works [1, 14, 2] focused on the extraction of features from time series with BoW approaches. As presented in 1.2.5, the main problem from this representation is that it extracts local descriptor, thus we lose the time information which doesn't cope with our main point. In [2] the authors were able to take into account the relation between variables but it was tested on low sized database and we do not expect that this method could be extended to millions of time series.

Though we can keep one lesson from the works focusing on classification : each of these works are transforming the data into suitable vectors that are then fed to classifier. We would like to see how we could transform the TS into vectors that could be manipulated with indexing methods as we know there are efficient methods out there that can manage large sets of high dimensional vectors.

There is several way to manage them, though as it is not our main concern we intend to use the library FLANN [10] that has been recognized as a really efficient way to index a lot of high number of vectors using randomized kd-tress and k-means. If we end up satisfied with the vector representation we could even try to use an other method presented in [6] that has performed really well on several millions of vectors and might be one of the best method of indexing high dimensional vectors to increase our performance.

From what we have still not ruled out with the methods manipulating time series, the shapelet transform looks promising. Indeed this method allows for a good representation of a time series in high dimension while carrying good information. In [8] the authors use this transformation for a classification task on UTS and try to extract shapelets that would represent classes, thus they manage few vectors. The extraction of a shapelet representing in itself a multivariate time series for each time series would be challenging but we could expect great results from this. Though like the representation with BoW we still loose a part of the time information that is not encoded in the shapelet.

Eventually while taking into account everything we have seen so far with the similarity measures presented in this study, the one that corresponds the best to the approach we are trying to use is the global alignment kernel. Though the method presented in the paper would not be convenient as we want to generate vectors; using kernel methods looks really promising and could give us the opportunity to meet our needs. What we would like to explore is a way to represent times series through high dimensional vectors where the scalar product between them would give us a kernel approximation (currently these vectors are represented by the alignments between TS). These vectors would have all the variable of a multivariate time series encoded and would also keep the time information though it would not be something we have to manage afterwards. We could then use them with the indexing methods that we referred to above.

From this study we can conclude that representing TS with vectors looks promising for our needs as we could benefit from indexing methods that are really efficient even on a scale of several million of TS. We will need to focus on the way to represents the TS that would best describe them. Both shapelet transform and kernel approximation are interesting methods to deepen.

References

[1] Adeline Bailly, Simon Malinowski, Romain Tavenard, Thomas Guyet, and Laetitia Chapel. Dense bag-of-temporal-sift-words for time series classification. 2016.

[2] Mustafa Gokce Baydogan and George Runger. Learning a symbolic representation for multivariate time series classification. *Data Mining and Knowledge Discovery*, 29(2):400–422, 2015.

[3] Qiuxia Chen, Lei Chen, Xiang Lian, Yunhao Liu, and Jeffrey Xu Yu. Indexable pla for efficient similarity search. In *Proceedings of the 33rd international conference on Very large data bases*, pages 435–446. VLDB Endowment, 2007.

[4] Marco Cuturi. Fast global alignment kernels. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 929–936, 2011.

[5] Marco Cuturi, Jean-Philippe Vert, Oystein Birkenes, and Tomoko Matsui. A kernel for time series based on global alignments. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 2, pages II–413. IEEE, 2007.

[6] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128, 2011.

[7] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11. ACM, 2003.

[8] Jason Lines, Luke M Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 289–297. ACM, 2012.

[9] Chen Luo and Anshumali Shrivastava. Ssh (sketch, shingle, & hash) for indexing massive-scale time series. *arXiv preprint arXiv:1610.07328*, 2016.

[10] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[11] Jinglin Peng, Hongzhi Wang, Jianzhong Li, and Hong Gao. Set-based similarity search for time series. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2039–2052. ACM, 2016.

[12] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270. ACM, 2012.

[13] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, 26(1):43–49, 1978.

[14] Patrick Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, 2015.

[15] Pavel Senin and Sergey Malinchik. Sax-vsm: Interpretable time series classification using sax and vector space model. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 1175–1180. IEEE, 2013.

[16] Jin Shieh and Eamonn Keogh. i sax: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631. ACM, 2008.

[17] Lingxi Xie, Richang Hong, Bo Zhang, and Qi Tian. Image classification and retrieval are one. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 3–10. ACM, 2015.