AFRL-AFOSR-JP-TR-2018-0051

Automatic analysis of satellite image time series

Francois Petitjean
MONASH UNIVERSITY

06/15/2018
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ IOA
Arlington, Virginia 22203
Air Force Materiel Command

DISTRIBUTION A: Distribution approved for public release.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
15-06-2018	Final	24 May 2016 to 23 May 2018

4. TITLE AND SUBTITLE
Automatic analysis of satellite image time series

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA2386-16-1-4023

5c. PROGRAM ELEMENT NUMBER
61102F

6. AUTHOR(S)
Francois Petitjean

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
MONASH UNIVERSITY
WELLINGTON RD
CLAYTON, 3800 AU

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AOARD
UNIT 45002
APO AP 96338-5002

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/AFOSR IOA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-AFOSR-JP-TR-2018-0051

12. DISTRIBUTION/AVAILABILITY STATEMENT
A DISTRIBUTION UNLIMITED: PB Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This project tackles the analysis of series of satellite images by scaling up research into time series classification to large quantities of data. Latest Earth observation satellites (Sentinel-2, Landsat-8, Vens) are now imaging Earth every 2-5 days at high resolution. This opens up incredible opportunities to closely monitor the land-use and land-cover of the planet over time. Current capability cannot make the most of these opportunities, because state-of-the-art research into time series classification does not scale to this amount of data. This project explored how to scale up one of the state-of-the-art algorithms for time series classification NN-DTW (the Nearest Neighbor algorithm coupled with the Dynamic Time Warping similarity measure) - to large quantities of data. In the first year, the research team focused on the classification time and introduced a novel method that allows classification of time series up to 500x faster; while it would have taken 1 CPU-year to create a temporal map of the Houston area at 10m resolution with previous state-of-the-art approaches, the research team showed an ability to do it in less than 4 hours on a single computer. In the second year, they focused on the training time and introduced a novel method to efficiently learn a classification model from data. They showed that our approach allows to gain up to 3 orders of magnitude in training time. This work was described in two publications in the 2017 and 2018 editions of the SIAM International Conference on Data Mining. The work for the second year was awarded Best Paper.

15. SUBJECT TERMS
Dynamic time warping, Satellite image, Time series classification, Temporal land-cover map, Scalability, Hierarchical classification, Expert knowledge, Local constraints

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			ROBERTSON, SCOTT

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Unclassified	Unclassified	Unclassified	SAR	**PAGES**	**19b. TELEPHONE NUMBER** *(Include area code)* +81-042-511-7008

Final Report for AOARD Grant FA2386-16-1-4023

"**Automatic analysis of Satellite Image Time Series**"

May 7, 2018

Name of Principal Investigators (PI and Co-PIs): Dr François Petitjean
- e-mail address : francois.petitjean@monash.edu
- Institution : Monash University
- Mailing Address : Monash University, 25 Exhibition Walk, VIC 3800, Australia
- Phone : +61 3 990 59182
- Fax : +61 3 990 55159

Period of Performance: 05/24/2016 – 05/23/2018

A. Abstract:

This project tackles the analysis of series of satellite images by scaling up research into time series classification to large quantities of data. Latest Earth observation satellites (Sentinel-2, Landsat-8, Venμs) are now imaging Earth every 2-5 days at high resolution. This opens up incredible opportunities to closely monitor the land-use and land-cover of our planet over time. We cannot currently make the most of these opportunities, because state-of-the-art research into time series classification does not scale to this amount of data. This project explores how to scale up one of the state-of-the-art algorithms for time series classification – NN-DTW (the Nearest Neighbor algorithm coupled with the Dynamic Time Warping similarity measure) - to large quantities of data. In the first year, we focused on the classification time and introduced a novel method that allows classification of time series up to 500x faster; while it would have taken 1 CPU-year to create a *temporal* map of the Houston area at 10m resolution with previous state-of-the-art approaches, we are now able to do it in less than 4 hours on a single computer. In the second year, we focused on the training time and introduced a novel method to efficiently learn a classification model from data. There again we show that our approach allows to gain up to 3 orders of magnitude in training time. In terms of publications, the two works were respectively accepted for publication for both the 2017 and 2018 editions of the SIAM International Conference on Data Mining. The work for the second year even obtained **Best Paper Award**.

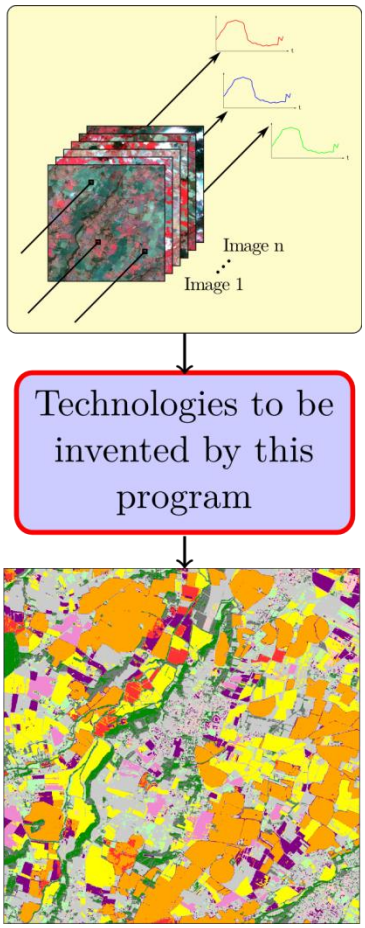

Figure 1: From a series of satellite images to a temporal land-cover map; better seen in color.

B. Introduction:

Earth Observation (EO) satellites have long made it possible to "step back and see the bigger picture", providing indispensable perspective on the status of large-scale agricultural, environmental, and climate problems to inform our decisions. But until recently, the picture we have seen has largely been a static one. This is now changing with satellites such as Sentinel-2, Venμs and Landsat-8 that are frequently imaging Earth at high-resolution, and at no charge to end-users. This introduces unprecedented opportunities to monitor the dynamics of any region of our planet over time.

These satellites produce vast streams of unprecedentedly rich data in the form of time series, enabling the creation of nuanced, temporal land-cover maps that describe the evolution of an area over time (see Figure 1).

Presently, state-of-the-art research into *time series classification* lags behind the demands of the latest space missions, which produce *terabytes* of data each day. This is mostly because most of the research into time series classification has been done with datasets that hold no more than 10 *thousand* time series [10]. In contrast, the Sentinel-2 and Landsat-8[1] satellites gather over 10 *trillion* time series, capturing Earth's land surfaces and coastal waters at resolutions of 10–60m. Existing machine learning methods for *classifying time series* could not scale to such vast volumes of data.

General aims. Land-cover has been declared one of 50 most important information for climate [39]. Land-cover maps associate to each geographic area (x,y) its cover or use information such as 'urban', 'wheat crop', 'deciduous forest' or 'lake'. When learning to accurately and automatically build such maps, it is often critical to use the information about the evolution of the (x,y) area over time. This is why learning to predict land-cover/land-use has become such an important task in remote sensing and machine learning [2-6]. This process is illustrated in Figure 1 where such a map is created from a satellite image series and where the different colors represent different classes of evolution; for instance, in orange is depicted the temporal class of "wheat crops".[2] However, technologies that can effectively perform time series classification at this scale are yet to be invented.

Specific aims. This project seeks to scale one of the leading methods for time series classification – NN-DTW – to the data that is produced by current Earth Observation programs. We developed technologies that make it possible to produce accurate and regular maps about the evolution of its surface. More specifically, we tackled the following scientific challenges:

1. **Scalability.** Sentinel-2 gathers time series over 10 trillion geographic areas (around 130 million square kilometers at 10m resolution – that is 13 trillion time series time series). Quite simply, current technologies for time series classification cannot cope with this scale (further details in the next section). This is the strand of research that has been tackled in the first year of this project; we developed methods that are specifically designed to function at billion-scale.

2. **Accuracy.** To perform best, state-of-the-art methods into time series classification require fine tuning of their parameters. Without this tuning, the system can lose most of its predictive power [14]. All existing learning techniques for these parameters have been developed for datasets with at most $N = 10{,}000$ time series, with time and memory complexities of this learning process that is in $O(N^2)$, which makes it impossible for all satellite image time series datasets. In the second year of the project, we developed techniques that can learn the parameters of state-of-the-art time series classification systems up to 3 orders of magnitude faster.

C. Method/Theory/Experiment: Scalable time series classification

C.1. Background

Time series classification: There is a significant body of research into time series classification, mostly leveraging similarity measures specifically designed for time series. Once

[1] NASA has worked with the ESA to cross-calibrate Sentinel-2 with Landsat-8 – see http://landsat.gsfc.nasa.gov/?p=4268.

[2] Note that images are provided with geometric and radiometric corrections (Level 2A), so that a pixel (x,y) always maps to the same geographic area and radiometric levels are comparable across all images in the series.

a similarity measure has been chosen, the 'Nearest Neighbour' classifier is typically used to perform the classification. There are dozens of similarity measures, of which the most representative are the Euclidean distance, the longest common subsequence (LCSS), as well as Dynamic Time Warping (DTW) [12] and its derivatives DDTW, WDTW, DTWCV, and WDDTW. Comparison studies are available in [13,14]. Other highly-used techniques include Shapelet [15], identifying subsequences of the series that can be used as a query to distinguish between classes; BOP, BOSS, and SAX-VSM [16], which leverage on building a dictionary of common subsequences in each class; Deep Convolutional Networks [17], which learn and leverage on high-order features of the time series; and manual feature engineering. It is increasingly accepted that the Nearest Neighbour (NN) algorithm with DTW (and derivatives) measure is the technique of choice for several time series classification problems [13], and in fact for EO data (see details in the next paragraph). The evidence in [14] strongly suggests that the structure of time series (autocorrelated values, high-apparent dimensionality, distortions of the time-axis) lends itself to the Nearest Neighbour algorithm and to NN-DTW in particular.

Our experience in the classification of satellite image time series: From 2009, in preparation for the then-upcoming Sentinel-2 program, I was part of the French Space Agency and developed time series techniques for one of the first high-resolution satellite image series from which time series could be studied (80 Formosat-2 images). That work gave rise to a series of 12 published papers, including: [4] in which we validated the consistency of the DTW measure and developed techniques to integrate prior knowledge about phenology; [5] in which we showed how to integrate spatial information for time series analysis, and [6] in which we showed how to make the most of multi-resolution series of images. Other important works have followed, with further refinement and validation of DTW for Earth Observation (EO) data [7,8], and wide-scale assessment of out-of-the-box classifiers (decision tree, Random Forest, and SVM) worldwide [2] and in Australia [3]. Here too, the particular characteristics of EO data lend themselves to the use of DTW. This is because (1) the series are irregularly sampled, (2) many phenomena of interest have a behaviour that can be modulated by weather artefacts, and (3) the series are generally too short for methods such as Shapelets, BOSS and SAX-VSM.

Limitations of current approaches for latest generation EO data: The relevance of NN-DTW for several time series classifications has meant that the most recent research has focused on mitigating its oft-lamented drawback: its time complexity. With a training database composed of N time series of length L, each classification requires $O(N \cdot L^2)$ operations. Creating a temporal map of Victoria (238,000 km^2 and 2.3 billion time series) requires a training database holding at least 1 million time series of 50 images in length. My simulations reveal that a direct implementation would require more than 30 years of computation. Even making the most of parallel computation, improvements of three to four orders of magnitude are required to tackle the data from latest EO satellites.

In [29], my collaborators at UC Riverside showed how to pull $O(N \cdot L^2)$ towards $O(N \cdot L)$. However, these technologies require a constant sampling rate of the time series, which is incompatible with the nature of EO data. Moreover, little attention has been paid to mitigating the influence of N (see [27] for details), mostly likely because standard archives have training databases with *a few thousands* time series at most [10]. This is critical for EO data, because it has potential to reduce the complexity of NN-DTW by several orders of magnitude, given that N is in the millions.

Moreover, it is important to note that, to perform best, the NN-DTW algorithm requires to set a parameter w that decides upon the maximum allowed time flexibility when comparing temporal profiles. This constraint corresponds to setting a limit such as: "The state of 2 crops cannot be compared if the sample was taken more than w samples/days/weeks apart." The state-of-the-art method to set this parameter is by cross-validation, which requires $O(N^2 \cdot L^3)$

operations.[3] This is the challenge tackled in the second year of this project.

C.2. Proposed approach

As highlighted above, we have made 2 main contributions in this project, with 2 associated publications. We describe both methods first and then turn to their associated results in Section D.

C.2.1. Proposed approach #1 – Time Series indexing to improve classification time

We aim at making the nearest neighbor algorithm using Dynamic Time Warping (NN-DTW) scalable to large databases of time series. Using this algorithm, classification of a *single* time series requires between $\Omega(N \cdot L)$ and $O(N \cdot L^2)$ operations, where N is the size of the training database and L the length of the time series. For EO data, N is typically greater than 10^6, which makes classification infeasible using this state of the art algorithm. This starkly contrasts with the datasets that researchers have been using to develop time series classification algorithm. Figure 2 illustrates this point: while all datasets of the standard archive of time series [10] can be classified in less than 30 minutes, creating a temporal land-cover map for just a city like Houston (16 million time series) assuming a bare minimum of 1 million training examples would take about a year to complete. To create a land-cover map of Texas (7 billion time series) with a reasonable training dataset of 100 million samples would require 30k years of computation.

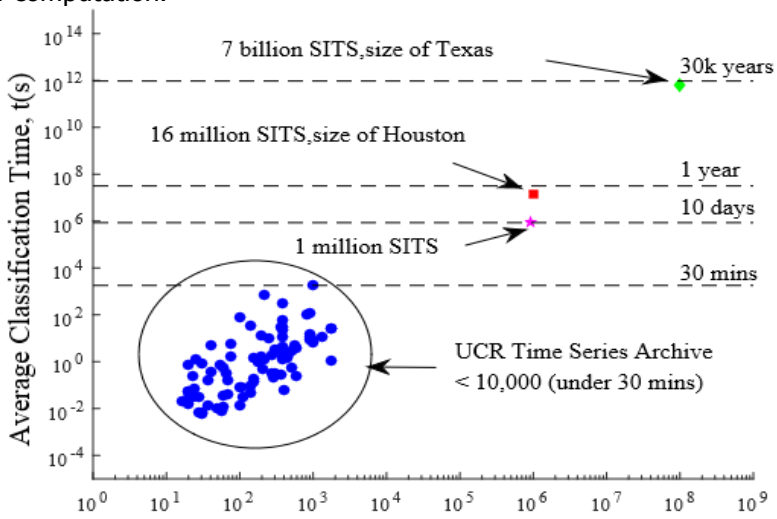

Figure 2: NN-DTW would take 30,000 years to classify the state of Texas. The x-axis represents the size of the dataset to classify (in log-scale).

With these motivations, this work tackles Contract Time Series Classification, where we would like to produce the most accurate classifier under a contracted time (obviously significantly smaller than running the NN-DTW). We propose a new algorithm that efficiently indexes the training database using a hierarchical K-means tree structure specifically designed for DTW. We will show that our algorithm reduces the time per query while retaining similar error to the state of the art, NN-DTW.

Our algorithm has been published at the 2017 SIAM International Conference for Data Mining [40]. This paper is available at http://bit.ly/ScalableTSClassification.

An efficient and popular method of scaling up NN classification on large datasets is to use the

[3] Leave-one-out cross-validation requires N^2 comparisons for each value of $w \in [0, L]$. With a value of w, one comparison has complexity $O(w \cdot L)$. Overall this is $O(N^2 \cdot (L + 2L + 4L + \cdots + L^2)) = O(N^2 \cdot L \cdot (1 + 2 + 4 + \cdots + L)) = O(N^2 \cdot L \cdot L^2) = O(N^2 \cdot L^3)$.

k-d tree structure, where k is the dimensionality of the Euclidean search space [30]. k-d tree allows faster retrieval of the nearest neighbour to the query sample in a short amount of time. It is very efficient in the Euclidean space. However, it is not applicable for DTW-induced space as k-d tree partitions each dimension of the data recursively while DTW makes associations across dimensions. Because of that, k-d tree will not be applicable to NN-DTW for time series classification. Note that most traditional indexing techniques require the space to be induced by a metric, in particular with a dissimilarity measure for which the triangular inequality holds. DTW is however not a metric and does not have this propriety [25].

Our algorithm, *Time Series Indexing (TSI)* is an adaptation of Priority Search K-means Tree (PSKMT) [26] to index time series embedded in a space induced by DTW. The general outline is as follows. At training time, we construct a hierarchy of K-means clusterings over the training dataset; this prepares the indexing data structure that will allow fast querying at testing time. The K-means clustering is performed using DTW as the similarity measure for associating time series to their closest centroids. DBA [27, 31] is used to create and refine the centroids from the associated time series in the expectation phase. At testing time, we maintain three priority queues. The first two queues store the potential branches to explore once exploration of the current branch is complete. The first stores those for which full DTW to the query has been calculated and the second stores those for which only lower bound have been computed. The third queue stores the nearest neighbours. Since we prune off DTW with lower bound (LB), having 2 priority queues ensures that we always traverse from the actual closest branch without having to compute the full DTW distance for all potential branches. We start by descending the tree from the root to the first leaf, at each internal node following the branch closest to the query but pushing the alternatives to the priority queues. Those alternatives that can be excluded just on the lower bound go to the second queue while those for which the full DTW distance is computed go to the first queue. We explore the leaf, and then proceed to the closest branch that was not explored on the path to that leaf (stored as the head of the first queue). We continue in this cycle, stopping when we have exhausted our "contracted time" (or have explored the full tree).

Algorithm 1: build_tree(D, K, I_{max}, w)

Input: D: Time series dataset
Input: K: Branching factor
Input: I_{max}: Maximum k-means iterations
Input: w: Warping window

```
1 if |D| ≤ K then create_leaf(D) ;
2 else
3     P = kmeanspp(D, w);
4     for iterations = 1 : I_max do      // k-means
5         C = assign_to_centroids(P, D, w);
6         for all C_i ∈ C do P_i = DBA(P_i, C_i, w) ;
7     end
      /* recursively build tree                    */
8     for all C_i ∈ C do build_tree(C_i, K, I_max, w) ;
9 end
```

Algorithm 2: assign_to_centroids(P, D, w)

Input: P: Cluster centroids
Input: D: Time series dataset
Input: w: Warping window
Output: $cluster$: Clusters of time series

```
1 cluster = ∅;
2 for all S_i ∈ D do
3     nearest_p = search_nearest_lb(S_i, P, w);
4     cluster[nearest_p].add(S_i);
5 end
6 return cluster;
```

Algorithm 3: search_tree(T, Q, L, w)

Input: T: Hierarchical k-means tree
Input: Q: Query time series
Input: L: Number of time series to examine
Input: w: Warping window
Output: kNN: k nearest neighbours

```
1  Initialize priority queues & seen=0
2  W = envelope(Q, w);
3  traverse_tree(T, Q, W, PQs, L, seen, w);
4  while (PQs not empty) && (seen < L) do
       /* find nearest branch                    */
5      while lb_PQ.top < dtw_PQ.top do
6          lb_branch = lb_PQ.dequeue;
7          d = DTW(Q, lb_branch.Centroid, w);
8          dtw_PQ.enqueue(lb_branch);
9      end
10     if dtw_PQ not empty then
11         T = dtw_PQ.dequeue;
12         traverse_tree(T, Q, W, PQs, L, seen, w);
13     end
14 end
15 kNN = nn_PQ.getAllData;
16 return kNN;
```

We detail the proposed algorithms below.
Algorithm 1 presents the algorithm for building the tree. The root node contains all the training

data. The algorithm recursively clusters the data associated to each node into K clusters. A branch is formed leading to each of the K child nodes, each child node associated with the data in one cluster. The branch is labelled with the DTW average of the time series in the node to which it leads. The data associated with each child node is then clustered into K sub-clusters; and so on recursively. All nodes are labelled with the majority class of the data associated with them. This allows the algorithm to give a plausible class prediction even when we have not yet reach a leaf node. Every recursion is initialized using the standard non-deterministic K-means++ algorithm [32]. The recursion stops when a node contains K or fewer time series. To further speed up the clustering process, LB Keogh is used with NN-DTW in the search nearest LB sub-routine to assign each time series to the nearest cluster centroid, as described in Algorithm 2. Our algorithm is not limited to just LB Keogh. It can be used with any DTW lower bounding functions such as LB Improved [25], depending on the application. In this work, we chose to use LB Keogh because in general, it performs well for most time series datasets and doesn't require computation of an additional projection (as LB-improved does).

Algorithm 4: traverse$(T, Q, W, PQs, L, seen, w)$

```
Input: T, Hierarchical k-means tree
Input: Q, Query time series
Input: W, Envelope for query time series
Input: PQs, dtw_PQ, lb_PQ and nn_PQ
Input: L, Number of time series to examine
Input: seen, Time series seen so far
Input: w, Warping window
1  if N is Leaf then
2      S = T.getAllData;
3      lb_distance = sort_with_lb(W, S);
4      for all S_i ∈ S do
5          worst_d = nn_PQ.firstDistance;
6          if lb_distance_i < worst_d then
7              d = DTW(Q, S_i, w);
8              if d < worst_d then
                   nn_PQ.enqueue(S_i, d) ;
9          end
10         if ++seen == L then break;
11     end
12 else
13     C = T.getChildren;
14     dtw_flag = [false, ...false];
15     best_so_far = inf;
16     distances = sort_with_lb(W, C);
17     for all C_i ∈ C do
18         if distances_i < best_so_far then
19             distances_i = DTW(Q, C_i, w);
20             dtw_flag_i = true;
21             if distances_i < best_so_far then
22                 best_so_far = distances_i;
23                 C_q = C_i;
24             end
25         end
26     end
27     for all C_i ∈ C except C_q do
28         if dtw_flag_i then
               dtw_PQ.enqueue(C_i, d_i) ;
29         else lb_PQ.enqueue(C_i, d_i) ;
30     end
31     traverse_tree(C_q, Q, W, PQs, L, seen, w);
32 end
```

Algorithm 3 describes the tree search algorithm. The tree is searched by first traversing down the tree to the first leaf node, outlined in Algorithm 4. At each level of the tree, the algorithm proceeds with the nearest branch to the query time series. To efficiently search for the nearest branch, we first sort all the branches in ascending lower-bound distance to the query, then use NN-DTW to find the closest branch. This further minimizes DTW computations. The unexplored branches where DTW have been computed will be enqueued into the DTW priority queue while the remaining ones will be enqueued into the LB priority queue. These priority queues are implemented as min-heaps [6], with standard enqueue and dequeue functions. When the query reaches a leaf node, the algorithm searches for the nearest time series using the same method as searching for the nearest branch. The k nearest time series found in the search so far are kept in the nearest neighbour priority queue, implemented as a max-heap. A max-heap allows faster retrieval of the k^{th} nearest neighbour from the query. After exploring the leaf node, the algorithm proceeds to the next branch by dequeuing the DTW priority queue. Both DTW and LB priority queues are compared to ensure that the closest branch to the query is in the DTW priority queue: if the head of the LB priority queue is smaller than the head of the DTW priority queue, we dequeue that branch, compute its DTW distance and enqueue it into the DTW priority queue. The algorithm stops searching the tree when it has seen at least L time series from the leaf nodes. Here, L can also represent the "contracted" classification time.

We have made the source code associated with TSI available at: https://github.com/ChangWeiTan/TSI.

C.2.2. Proposed approach #2 – Efficiently learning the warping window to improve training time

We aim at making the nearest neighbor algorithm using Dynamic Time Warping (NN-DTW) scalable to large databases of time series. NN-DTW is a leading algorithm for time series classification and a component of the current best ensemble classifiers for time series. However, NN-DTW is only a winning combination when its meta-parameter – its warping window – is learned from the training data. The warping window (WW) intuitively controls the amount of distortion allowed when comparing a pair of time series. With a training database of N time series of lengths L, a naïve approach to learning the WW requires $O(N^2 \cdot L^3)$ operations. This often results in NN-DTW requiring days for training on datasets containing a few thousand time series only. In this second work, we introduce FASTWWSEARCH: an efficient and exact method to learn the WW. We show on 86 datasets that our method is always faster than the state of the art, with at least one order of magnitude and up to 1000x speed-up. An example of compared training time is given in Figure 3 where we can exactly learn the window for 100,000 series in 6 hours instead of 7 days, a quantity of data that is infeasible to process with the state of the art.

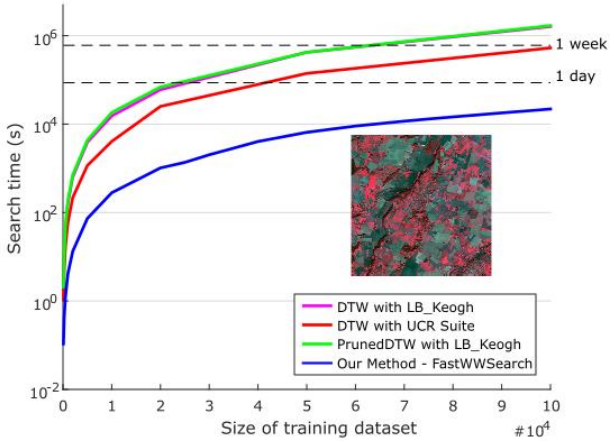

Figure 3: Training time (in log-scale) for NN-DTW where the warping window is learned.

Our algorithm has been accepted for publication at the 2018 SIAM International Conference for Data Mining [41]. It received **Best Research Paper Award.** This paper is available at https://bit.ly/2FGtJXu.

What is a warping window? A warping window w is a global constraint on the re-alignment that DTW finds (originally called Sakoe-Chiba band [12]), such that elements of two time series S and T can be mapped only if they are less than w elements apart, and we note it $DTW_w(S,T)$.
Why should the warping window be learned? The choice of the warping window (WW) has long been known to have a strong influence on accuracy [33,34]. This is exemplified in Figure 4 where we show the influence of WW onto the classification error. This shows both that WW can strongly influence the accuracy of the system, but also that different datasets require different values.

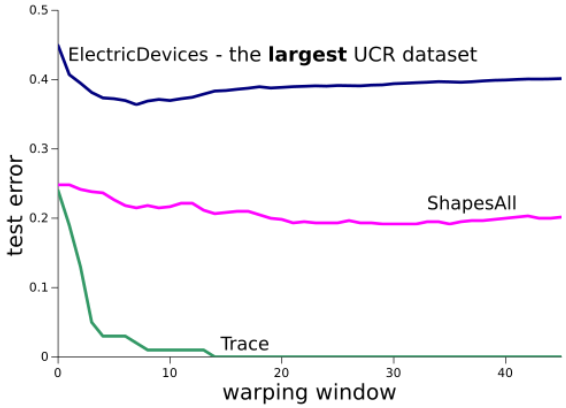

Figure 4: Classification error as a function of the warping window.

How is the warping window learned? Learning of the warping window is not a simple problem; accepted methods in the field are all based on cross-validation: either on leave-one-out (LOO-CV) or on x-fold cross-validation. In this work we focus on LOO-CV for the sake of clarity, with all our algorithms directly usable for x-fold cross-validation. Algorithms for LOO-CV are given in A.1 of our paper [41].

Related work. As learning the warping window involves leave-one-out cross-validation, the task boils down to being able to find the nearest neighbor of each time series in the training dataset (within the training dataset excluding themselves). We review below the state-of-the-art methods to perform this task efficiently.
Silva et al. [35] proposed PrunedDTW to speed up DTW computation itself. They first compute an upper bound and skip the cells of the cost matrix D that are larger. The authors were able to learn the optimal window size faster than the naive method [35]. However,
as we will show in the experiments, the improvement for warping window search is only minimal.
Rakthanmanon et al. [36] proposed the UCR Suite, which includes 4 optimization techniques: early abandoning, reordering early abandoning, reversing query and data roles in LB Keogh (see Section 2.4), and cascading lower bounds. Although they did not directly use their method to learn the warping, it is natural to repurpose it for this task.

DTW Lower Bounds. Learning WW via cross-validation involves being able to efficiently find the neighbor of a time series within the training dataset, i.e. to perform a NN-DTW query for each time series. Lower-bounds to DTW have long been used in this context, they allow us to avoid the (expensive) DTW calculation if it is not needed. Several bounds have been introduced including LB Kim, LB Keogh [37] and LB Improved [38]. Lower bounds can also be used in cascade, starting by the looser (and computationally cheap) one and progressing towards tighter lower bounds [36]. Algorithm A.2 in our paper [41] gives a simple nearest neighbor search with lower bounds and can easily be modified for cascading lower bounds by changing the LB function in line 3.

Our approach – Fast Warping Window Search for DTW
We now introduce our approach: FastWWSearch. In the first part, we start by introducing the mathematical properties that constitute the basis for our algorithm, which we introduce in the second part.

Theorem 1. Let S,T *be two time series,* w_1 *and* w_2 *two warping windows, and* $\overrightarrow{p_{w_1}}$ *and* $\overrightarrow{p_{w_2}}$ *their associated warping paths.* $\overrightarrow{p_{w_1}} = \overrightarrow{p_{w_2}} \Rightarrow DTW_{w_1}(S,T) = DTW_{w_2}(S,T)$*. In other words,* $DTW(S,T)$ *can only differ if the warping path differs.*

Theorem 2. Let S,T *be two time series,* w *a warping window, and* $\overrightarrow{p_w} =$

$\langle (i_1, j_1), \cdots, (i_K, j_K) \rangle$, *then* $(|i_k - j_k| < w)_{\forall k} \Rightarrow DTW_w(S,T) = DTW_{w-1}(S,T)$. *In other words, if no point of a warping path 'touches' the extremity of the warping window at* w *then* $DTW_w(S,T) = DTW_{w-1}(S,T)$.

Theorem 3. Let S,T *be two time series,* w *a warping window, we have* $DTW_w(S,T) \leq DTW_{w-1}(S,T)$.

Figure 5 below illustrates the combination of these 3 theorems by showing the value of $DTW_w(S, T_{\{1,2,3\}})$ as a function of w. It shows that DTW decreases monotically with w (Theorem 3) while the flat sections on the right of the plot illustrate Theorems 1 and 2. This Figure also shows that the path calculated for $w = 24$ remains valid down to $w = 1$ for T_1.

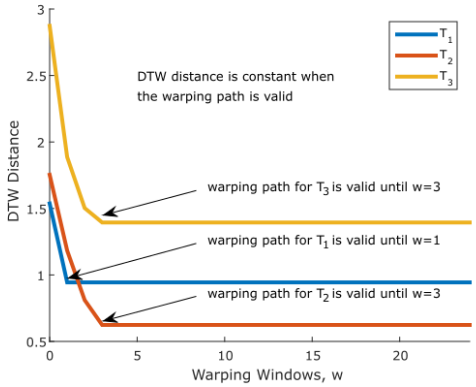

Figure 5: DTW distance for different ***w***

Theorem 4. Let S,T *be two time series,* w *a warping window, we have* $LB\_Keogh_w(S,T) \leq LB\_Keogh_{w-1}(S,T)$.

Intuition behind FastWWSearch. Learning the warping window can be seen as creating a $(N \times L)$-table giving the nearest neighbor (NN) of every time series for all windows. Such a table is illustrated on the right. Once that table is filled, the best value of the window can be learned in one pass over it. We can thus reframe the aim of FastWWSearch as the construction of such a table. Filling it naively, ie by computing DTW for each nearest neighbor $NN(i,w)$ using DTW only requires $O(N^2 \cdot L^3)$ operations. Until recently, most of the research effort had been put onto finding bounds for a fixed value of a warping window. Our method explores bounds across columns of this table, ie from one WW to another. To take advantage of Theorems 3 and 4, we order our computations by *decreasing window size*. Our algorithm iterates from larger values of windows down to smaller ones. This has the consequence of obtaining bounds 'for free' when getting to window w. Moreover, ordering our computations by decreasing window size also allows us to make the most of Theorems 1 and 2. Referring to Figure 5, the long flat tails correspond to large validity windows for DTW_{L-1}. It means that, in that flat section, no bounds are at all necessary, because we already know that the value of DTW (previously computed) has not changed. We now have all the elements necessary to the presentation of our FastWWSearch Algorithm.

	Nearest neighbor at warping windows				
	0	1	$\cdots$	$L-2$	$L-1$
T_1	$T_{24}(2.57)$	$T_{55}(0.98)$	$\cdots$	$T_{55}(0.98)$	$T_{55}(0.98)$
$\vdots$			$\vdots$		
T_N	$T_{60}(4.04)$	$T_{47}(1.61)$	$\cdots$	$T_{47}(1.61)$	$T_{47}(1.61)$

We start by presenting LazyAssessNN in Algorithm 1. LazyAssessNN is a function that, given a pair of time series (S,T), establishes if they can be less than a given distance d apart for a warping window w. It functions in a lazy fashion, by making the most of all possible bounds that we presented above. The algorithm tries lower bounds of increasing complexity until one of two things happen: (1) either a lower bound or $DTW_w(S,T)$ itself is greater than d, in which case the procedure aborts; or (2) we have $DTW_w(S,T) < d$ and we have actually calculated $DTW_w(S,T)$. LazyAssessNN is lazy in that it postpones calculations for as long as it is possible to do so. As we will see next, one has to imagine here that LazyAssessNN will be called several times for the same pair of time series (S,T) for decreasing values of w. When w decreases, any value that was previously calculated for a larger window, becomes a lower bound for the current w.

Algorithm 1: LAZYASSESSNN$(\mathcal{C}_{(S,T)}, w, d, S, T)$

Input: $\mathcal{C}_{(S,T)}$: cache storing the previous measure between S and T
Input: w: the warping window
Input: d: the distance to beat
Input: S, T: the time series to measure
Result: $\mathrm{DTW}_w(S,T)$ if $\geqslant d$, else `pruned`

```
1  if C_(S,T) = ∅ then C_(S,T) ← LB_Kim(S,T)
2  switch C_(S,T).stoppedAt do
       // LB calculated with larger window w'
3      case DTW_w' do
4          if w ∈ C_(S,T).valid ∧ C_(S,T).value < d then
5              return C_(S,T).value
6      case LB_Kim or LB_Keogh_w' do
7          if C_(S,T).value ⩾ d then return pruned
       // Cascading LB_Keogh and DTW
8      otherwise do
9          C_(S,T) ← LB_Keogh_w(S,T)
10         if C_(S,T).value ⩾ d then return pruned
11         C_(S,T) ← LB_Keogh_w(T,S)
12         if C_(S,T).value ⩾ d then return pruned
13         C_(S,T) ← DTW_w(S,T)
14         if C_(S,T).value ⩾ d then return pruned
15         return C_(S,T).value
```

We use a cache to store the previous results of LazyAssessNN obtained for larger windows. We first start by checking if the cache has been initialized; if not we compute LBKim, which is valid regardless of the warping window (line 1). On line 2, we then test where the cache last stopped, i.e. was it computing a lower bound for that target window w, was it computing a lower bound for another window $w' > w$, or was it computing an actual DTW distance (that might be still valid). We then assess if it last stopped having had calculated DTW for a larger window (lines 3-5); if that $DTW_{w'}$ has a path that is still valid and its value is smaller than d, then we return that value of the distance. We then assess if it had calculated $DTW_{w'}$ that is still valid and smaller than d. It is important to observe here that the code terminates when a distance is larger than d or DTW_w is computed. If we cannot prune with $DTW_{w'}$ we proceed and check if we are able to prune using previously computed bounds (lines 6-7). Otherwise from lines 8 to 12, we use cascading lower bounds, testing if we can prune after each of them. Finally, if all the bounds fail to prune T, we compute DTW_w, store the results and if $DTW_w < d$, T is the new NN for S (line 13). We will see in our main algorithm that the next call to LazyAssessNN for the pair (S,T) will be with a smaller w. We take the bounds ordered by computational complexity, which in practice usually correlates with tightness [36]. $DTW_{w'}$ will also generally be tighter than $LBKeogh_w$, especially when w' is close to w.

We now turn to our core algorithm: FastWWSearch. Let us recall that the central aim of our algorithm is to build an $(N \times L)$-table that contains the nearest neighbor of each time series (out of N) and for each value of the warping window (out of L). Once this table has been calculated, one pass over it is sufficient to determine the best value of the warping window.

That pass is the entry point to FastWWSearch and is presented in Algorithm 2. It assumes there exists a method to fill the table and returns the warping window with the lowest leave-one-out cross-validation error on the set of series. Obviously, the core of our approach

Algorithm 2: FASTWWSEARCH

Data: $\mathcal{T}$: training data
Result: $w^\star$: best warping window

```
1  NNs ← FastFillNNTable(T)
2  bestNErrors ← |T| + 1
3  for w ← 0 to L − 1 do
4      nErrors ← 0
5      foreach T_t ∈ T do
6          if NNs[t][w].class ≠ T.class then nErrors++
7      if nErrors < bestNErrors then
8          bestNErrors ← nErrors
9          w* ← w
```

resides in how we calculate this $(N \times L)$-table, which we present in Algorithm 3.

Algorithm 3: FASTFILLNNTABLE($\mathcal{T}$)

Input: $\mathcal{T}$ the set of time series
Result: NNs$[N][L]$ the nearest neighbors table

```
1  Define LANN as LAZYASSESSNN
2  NNs.fillAll(_, +∞)
3  T' ← ∅
4  for s ← 2 to N do
       // We want to update NNs wrt adding S
5      S ← T_s
6      T' ← T' ∪ {T_{s-1}}
7      foreach T ∈ T' do C_{S,T} ← ∅
8      for w ← L − 1 down to 0 do
           // If we already have NN of S for w
9          if NNs[s][w] ≠ ∅ then
               // Update table NNs[t][w]_{1≤t≤s-1}
10             for t ← 1 to s − 1 do
11                 toBeat ← NNs[t][w].distance
12                 res ← LANN(C_{(S,T_t)}, w, toBeat, S, T_t)
13                 if res ≠ pruned then
14                     NNs[t][w] ← (S, res)
15         else
               // Check S against previous T ∈ T'
16             foreach T ∈ T' in asc. order using C do
17                 toBeat ← NNs[s][w].distance
18                 res ← LANN(C_{(S,T)}, w, toBeat, S, T)
19                 if res ≠ pruned then
20                     NNs[s][w] ← (T, res)
                   // Update NNs[t][w] if needed
21                 toBeatT ← NNs[t][w].distance
22                 resT ← LANN(C_{(S,T)}, w, toBeatT, S, T)
23                 if resT ≠ pruned then
24                     NNs[t][w] ← (S, resT)
               // Propagate NN for path validity
25             for w' ∈ NNs[s][w].valid do
                   NNs[s][w'] ← NNs[s][w]
```

At the highest level, our algorithm works by building this table for a subset $\mathrm{T}' \subseteq \mathrm{T}$ of increasing size, until $\mathrm{T}' = \mathrm{T}$. We start by building the table for T' comprising only the 2 first time series T_1 and T_2, and fill this $(2 \times L)$-table as if T' was the entire dataset. At this stage it is trivial that T_2 is the nearest neighbor of T_1 and vice versa. We then add a third time series T_3 from $\mathrm{T} \setminus \mathrm{T}'$ to our growing set T'. At this point, we have to do two things: (a) find the nearest neighbor of T_3 within $\mathrm{T}' \setminus \{T_3\} = \{T_1, T_2\}$ and (b) check if T_3 has become the nearest neighbor of T_1 and/or T_2. We can then add a fourth time series T_4 and so on until $\mathrm{T}' = \mathrm{T}$.

We now describe Algorithm 3 line by line. We start by initializing the NNs $(N \times L)$-table to $(\_, +\infty)$, which means that the table is empty and the distances are thus $+\infty$ (line 2).

We then initialize T' in line 3. Line 4: we start the iteration at 2 as the definition of NN only makes sense if there is at least 2 time series. We then proceed with some initializations (lines 5-7), including for the cache associated with S (line 7). We are then ready to find (a) the NN of S within T', and (b) update the NN for all $T \in \mathrm{T}'$ now that S has been added. We will do this operation for all w in descending order, starting with the largest value $L - 1$ (line 8). Note that to only assess a subset of all possible L values for w, one only need to modify this line; our only requirement is for the values to be taken in descending order. Line 9: we start by checking if we already have a NN for S from previous (i.e. larger) windows. Note that $NNs[s][w]$ here comes compulsorily from $DTW_{w',w'>w}(S, \_)$ for which its path is still valid. We then already have the NN of S and only need to check whether S has also become a NN for other time series in T' (lines 10-14). To this end, we get the previously calculated NN for such $T \in \mathrm{T}'$ (line 11), which we will use as the threshold of distance that we have to 'beat' (i.e. be smaller than) for S to become the NN of T. On line 12 we then call our LazyAssessNN function to assess if S has actually become the NN of T. If LazyAssessNN exits with `pruned`, it then means that S is not the NN of T, and thus that the previous NN of T is still valid, hence nothing has to be done. LazyAssessNN only exits with something else than `pruned` if $DTW_w(S,T) < \mathrm{toBeat}$, which means that S has indeed become the NN of T; we update this information on line 14. The `else` case starting on line 15 is if we didn't already have the NN of S from a previous window. We will then analyze all couple $(S,T)_{T \in \mathrm{T}'}$ and perform the NN update for S and T simultaneously. At this stage, it is possible that we will already have – from previous windows – some information about which $T \in \mathrm{T}'$ might be a better candidate to be the NN of S. This information is stored in the cache, which may contains different types of lower bounds. The number of calculations will be minimized if the very first T is actually the NN of S, because it will give the tightest possible pruning threshold first.

This is why we should first examine the time series that have highest potential to become the

NN of S and order the time series.[4] At line 17, we obtain the distance threshold from $NNs[s][w]$; the first time, we will have $NNs[s][w] = \emptyset$ which is associated with a value of $+\infty$, there will thus be computation of $DTW_w(S,T)$, which will later on be stored in $NNs[s][w]$ (line 20). From the second T, $NNs[s][w].distance$ stores the distance to the so-far NN of S; we use LazyAssessNN to prune candidates or replace the current one if it is better (lines 19-20). We then proceed by checking if S is the NN of T on lines 21--24 (same as lines 11--14). Finally on line 25, having processed all $T \in \mathrm{T}'$, $NNs[s][w]$ contains the actual NN of S at w, and we propagate this information for all $w', w' < w$ for which the warping path is also valid.

We have made the source code associated with FastWWSearch available at: https://bit.ly/2wgn5be.

D. Results and Discussion:

As for the method section, we divide here our results and discussion into two subsections, respectively corresponding C2.1 and C2.2.

D.1 Results and Discussions about time series indexing (TSI)

In this section, we assess the performance of our methods in terms of how quickly it is able to provide a neighbor that is in the close proximity to the actual neighbor of the query. To this end, we assess the classification performance of NN-DTW with increasing "contracted" querying time, and compare:

1. **LB_Keogh NN-DTW**, which is the state-of-the-art approach [24]. Time series are taken one-by-one; if its lower bound distance to the query is greater than the best-so-far neighbor, then the time series is pruned and the method proceeds to the next series, else its actual DTW distance is computed and it becomes the best-so-far neighbor if it is closer than the current best-so-far. To function under a "contract" setting, time series from the training database are sampled randomly without replacement and the class of the best-so-far neighbor is reported at different time. Because this strategy is not deterministic, we report the average results over 10 runs.

2. **Time Series Indexing (TSI)**: This is our proposed method, described in the previous section. When the contracted time is reached, we report the class of the best-so-far neighbor found. If a result is requested before the first leaf is reached, we use the majority class of the cluster associated with the closest centroid explored so far. The non-deterministic part of our algorithm is the initialization of the clustering using K-means++ [32]. We thus also report the average classification error-rate over 10 runs.

We start by presenting the results on the satellite image time series (SITS) case and then on the reference time series classification archive (85 datasets).

D.1.1. Satellite Image Time Series (SITS)

As motivated in the introduction, the new-generation Earth Observation (EO) satellites have begun imaging Earth frequently, completely and in high-resolution. This introduces

[4] Ranking LBKeogh lower bounds has been previously studied in our SDM'17 work (from the first year of this project). Here, we however have different types of lower bounds to interlace. LBKim often has a smaller value than LBKeogh simply because it only looks at 4 elements of series (vs L for LBKeogh). In addition, because $DTW_{w',w'>w}$ has tried to align S and T, it will probably represent a better estimate of DTW_w than LBKeogh. To reflect this, we rank the time series in T' using $LBKim/4$, $LBKeogh/L$, $0.8 \cdot DTW/L$ (the 0.8 factor is used to push DTWs forward when close to the LBKeogh values).

unprecedented opportunities to monitor the dynamics of any regions on our planet over time and revealing the constant flux that underpins the bigger picture of our world.
To the best of our knowledge, there has been little prior research into the classification of time series extracted from satellite image series also called Satellite Image Time Series (SITS). The ability to monitor and classify these time series will have strong impact in many domains especially in the agriculture industry, marine applications and for environment monitoring. In prior work [4], we showed that DTW is a good measure for such time series, because of the non-linear distortions of prototypical ground surfaces, i.e., the fact that two neighbouring surfaces might have slightly different growth rates, although belonging to the same class of crop/tree. In this work, we used 46 geometrically and radiometrically corrected images taken by FORMOSAT-2 satellite. These images are corrected such that every pixel corresponds to the same geographic area on Earth. Each of these images consists of 1 million pixels and each pixel represents a geographic area of $64m^2$, resulting in an area of $64km^2$ per image. Each geographic area (x,y) ($\approx$ pixel) in the image forms a time series with a length of 46, creating a dataset with 1 million time series. The series have been manually collected by experts in geoscience by a mix of photo-interpretation, ground campaigns and urban databases. All time series thus have a label about their temporal class such as *wheat*, *maize*, *broad-leaved tree*, etc (more details in [4]) The formation of Satellite Image Time Series is illustrated in Figure 3, labelled with their temporal classes, represented in different colours.

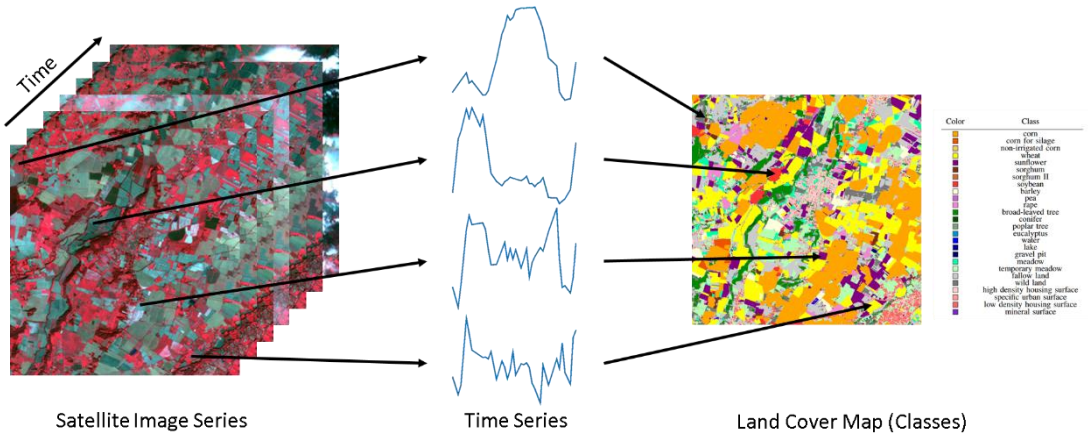

Figure 6: Production of a time series dataset from satellite image series

To ensure reproducibility of our results and encourage researchers to work on large time series datasets, we have obtained permission from the CesBIO and French Space Agency to make our satellite dataset available online at http://bit.ly/SDM2017; note that this is a very high-cost dataset (images are worth more than $100,000 and collecting the labels required months of work) which we hope will be a significant motivation to the field. As there are no pre-defined train/test sets for this dataset, we used 10-fold cross validation results. In this experiment, a warping window of 4 was used: this is aligned with the phenology of observed phenomena for which similar stages of growth cannot be distant by more than about a month [4]. The case study was conducted by varying the contract time from the least to the more permissive, i.e. with more and more time allowed in the contract to perform the classification of each query until we have gone through the whole training dataset. We record the error for

each query as we go through the whole training dataset. We present our results in Figure 4 where the x-axis, representing query time, is in log-scale.

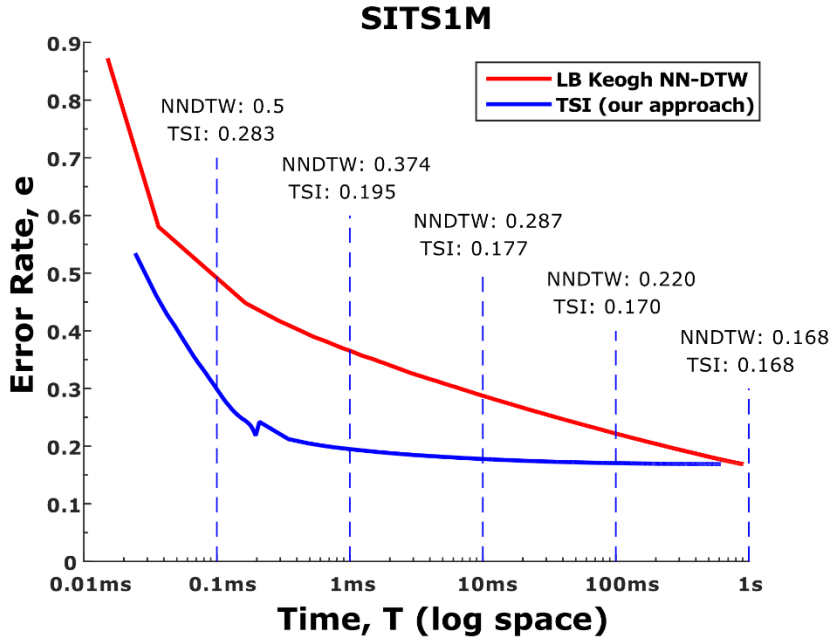

Figure 7: Compared error-rate obtained with our TSI method as compared with the state-of-ther-art LB_Keogh.

The first element to note is that our algorithm, TSI is significantly better than the state of the art; having its curve consistently under the state of the art. Second, we can see that if we had a "contract" to classify each time series in no more than 1ms, then TSI would obtain error rate of 19% as opposed to 37%. The same observation holds for 0.1ms (error of 28% vs 50%), 10ms (error of 18% vs 29%) and 100ms (error of 17% vs 22%).

This is a very important result: let us imagine that one is satisfied with an average error-rate of 20%, then using our approach would classify each query within 1ms, as opposed to 500ms for the state-of-the-art approach. Having a million time series to classify, this would translate to our approach, TSI finishing in **17 minutes** as opposed to **5.75 days for** LB Keogh NN-DTW; a 500x times speed-up. It is only when getting to the far right of the curve that the overheads of TSI (linked to the exploration of the tree and maintenance of the priority queues) would become disadvantageous. This makes sense because if the contract is that you have the time to explore all of the training set, then you might as well just do that rather than using our approximate search. Another point to note in Figure 4 is the slight jump in error for TSI at approximately 0.2ms. This is when the first leaf is expanded and the first class of an actual example is used in place of the majority class of a traversed branch.

D.1.2. On the reference UCR time series classification archive

We perform the same experiment as before for all the 85 time series classification datasets of the UCR archive [10]. Because these datasets have different sizes and length of the time series, we take the time necessary to run the NN-DTW as a reference time and study the performance of our system at different percentage time of this reference. Results are summarized in the table below where we rank the two methods in terms of accuracy, with the associated Wilcoxon Test statistics assessing the significance of the difference between the ranks.

LB_Keogh NN-DTW vs TSI					
	Average Ranks		Wilcoxon Test Statistics		
Intervals	NN-DTW	TSI	R^+	R^-	z
1%	1.529	**1.471**	2034.5	1620.5	-0.907
10%	1.841	**1.159**	3449	206	**-7.105**
20%	1.871	**1.129**	3451	204	**-7.114**
30%	1.806	**1.194**	3219.5	435.5	**-6.099**
40%	1.741	**1.259**	2903	752	**-4.713**
50%	1.671	**1.329**	2616	1039	**-3.455**
Average	1.743	**1.257**			

The average ranking allows us to identify the better performing classifier (emphasized in bold) if we reject the null hypothesis. In this case, our algorithm, TSI performs more accurately than the state of the art under a contracted time. The results show that TSI is more accurate than LB Keogh NN-DTW at all time intervals except for 1%, where we are unable to reject the null hypothesis. As the majority of datasets tested are small, the algorithms are unable to and even an approximate nearest neighbour at the 1% time interval, as a result of which they predict the majority class. Note that if the classes are very similar and the clusters do not well separate the classes, TSI can underperform LB Keogh NN-DTW. This is, for example, the case for the Computers dataset; the associated plot is available at http://bit.ly/SDM2017.

D.2 Results and Discussions about learning the warping window (FastWWSearch)

We compare FastWWSearch's ability to learn the warping window compared to the state of the art:

- **LB_Keogh** [37]: It searches for the best warping window as described in [41, Algorithm A.1] using LBKeogh as LB.
- **UCR Suite** [36]: It is the state of the art for fast NN-DTW and uses cascading lower bounds to replace LB in Algorithm A.1.
- **LB_Keogh-PrunedDTW**: The PrunedDTW algorithm [35] was introduced to speed up the calculation of DTW using upper bounds (instead of lower bounds). It assesses warping windows in ascending order and uses the results for a smaller w as the upper bound for the larger w. Note that we actually improve here on the original paper [35] by adding LB_Keogh to the search mechanism.
- **UCR Suite-PrunedDTW**: To make the comparison as fair as possible, we propose to combine the power of UCR Suite's lower bounding and the one of PrunedDTW's upper bounding. Again, this is an improvement on both methods.

We performed our experiments using all of the 85 freely available benchmark UCR time series datasets [10]. All methods are exact: they all learn the exact same value of the warping window and thus all return the same accuracy, which is why we only present running time experiments.

D.2.1. Speed-up on UCR datasets

Figure 8 shows the scatter plot of learning time for our FastWWSearch method (x-axis) vs all competitors (y-axis). This is a clear-cut and significant result: our method is faster than the state of the art for all datasets (above the line means that our method is faster).

There is also a slight upwards trend: as the task becomes more and more complicated, it seems that so is the improvement of our method over the state of the art. For easy task requiring less than 10 seconds with the state of the art, FastWWSearch gains one order of magnitude and performs in less than 1 seconds. This makes sense and this is not where the gain is the most interesting. However, as the task becomes harder, so is the advantage of our method over others getting more important. This even reaches up to 3 orders of magnitude

speed-up for very demanding tasks. For dataset HandOutlines for instance, the fastest state-of-the-art method requires 100 days ($9.10^6 s$), while FastWWSearch only needs 2.5 hours ($9.10^3 s$).

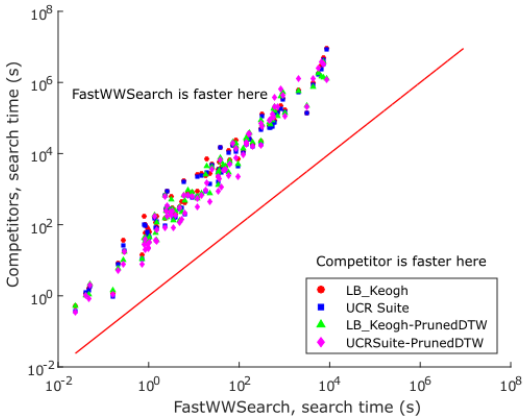

Figure 8: Average 10 runs results on the benchmark UCR datasets (better seen in color)

It is also interesting to summarize the results depending on size of the data and length of the series. We calculated the average speed-up for datasets with smaller training size ($N \leq 200$) and larger training size ($N > 200$). The average speed-up for smaller datasets was of 106x, while it was of 184x for larger ones. Regarding length, we calculated the average speed-up for datasets with shorter series ($L \leq 300$) and longer ones ($L > 300$). The average speed-up for datasets with shorter series was of 67x, while it was of 250x for longer ones. These results confirm that our algorithm is tackling both the N and L terms.

D.2.2. Scalability to 100,000 time series

It is now interesting to comment again on Figure 3 that was presented earlier. This dataset corresponds to 100,000 time series, associated to 100,000 'pixels' observed over time (over a series of satellite images - more details in [4]). In this task, the aim is to establish the land-use of a geographic area based on its radiometric evolution over a series of satellite images. Time series are required because, for instance, one needs the temporal dynamic to distinguish between types of crops (when they grow and are harvested, how fast they grow, etc).

This dataset is also particularly interesting because the time series are short with $L = 46$, which tends to isolate the influence of N on the scalability. We can see in Figure 3 that when $N = 100$, FastWWSearch makes it possible to gain 1.5 orders of magnitude in running time over UCR Suite and 2 orders over the other state-of-the-art methods; and those gains seem very stable as N grows. For $N = 100{,}000$, FastWWSearch only requires 6 hours to complete, while the fastest state-of-the-art method – in this case UCR Suite – requires 7 days (and more than 18 days for the others).

Note finally that we studied the question of whether incorporating PrunedDTW in our method would make it faster; our results show in [41, Appendix C] that it has little influence on the results.

E. Conclusions

In this project, we have made significant improvements to the scalability of one of the leading methods for time series classification: the nearest neighbour algorithm coupled with the dynamic time warping similarity measure (NN-DTW).

In the first part of this work, we have proposed the first algorithm to index DTW-induced space and showed that it is essential for the classification of time series when a large amount of data is available. We demonstrated that on a large remote sensing data where time series classification is critical, we are able to obtain the same accuracy 500x faster than the state of the art, NN-DTW search; we can thus classify the entire 1 million dataset in 17 minutes instead of 5 days. This is extremely promising for larger remote sensing datasets that contain hundreds of millions of examples [2].

In the second part of this work, we have proposed a novel algorithm to learn the warping window of NN-DTW. The warping window is the only parameter of NN-DTW – learning it is critical in order to obtain state-of-the-art results but the time complexity of this learning is extremely high, rendering it very difficult for large datasets. Our FastWWSearch algorithm makes it possible to gain up to three orders of magnitude in running time to learn the warping window, with the benefit increasing with the size of the dataset and lengths of the series. Our method is the first one able to learn a warping window for datasets with 100,000 time series.

F. References

1. A Allen, "The Value of Earth Observations from Space to Australia," *CRC-SI report*, December 2015, *http://bit.ly/ValueEO*
2. J Inglada *et al.*, "Assessment of an Operational System for Crop Type Map Production Using High Temporal and Spatial Resolution Satellite Optical Imagery." *Remote Sensing*, 2015.
3. Sheffield, K., Morse-McNabb, E., Clark, R., Robson, S. and Lewis, H., 2015. Mapping dominant annual land cover from 2009 to 2013 across Victoria, Australia using satellite imagery. *Scientific data*, *2*.
4. F. Petitjean, J. Inglada and P. Gançarski, "Satellite Image Time Series Analysis under Time Warping," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 8, pp. 3081-3095, 2012.
5. F. Petitjean, C. Kurtz, N. Passat & P. Gançarski, Spatio-Temporal Reasoning for the Classification of Satellite Image Time Series *Pattern Recognition Letters*, 2012, Vol. 33, Num. 13, pp. 1805-1815.
6. F. Petitjean, J. Inglada & P. Gançarski, "Assessing the quality of temporal high-resolution classifications with low-resolution satellite images time series," *International Journal of Remote Sensing*, 2014, Vol. 35, Num. 7, pp. 2693-2712.
7. V Maus, et al. "Open boundary dynamic time warping for satellite image time series classification." *IEEE IGARSS*, 2015.
8. X Guan et al. "Mapping Rice Cropping Systems in Vietnam Using an NDVI-Based Time-Series Similarity Measurement Based on DTW Distance." *Remote Sensing*, 2016.
9. R. Flamary et al. "Analysis of Multitemporal Classification Techniques for Forecasting Image Time Series," *IEEE Geoscience and Remote Sensing Letters*, 12.5 (2015): 953-957.
10. Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen and G. Batista (2015). *The UCR Time Series Classification Archive*. URL www.cs.ucr.edu/~eamonn/time_series_data/
11. M. Shokoohi-Yekta, J. Wang and E. Keogh. On the Non-Trivial Generalization of Dynamic Time Warping to the Multi-Dimensional Case. *SIAM SDM*, 2015.
12. H. Sakoe and and S. Chiba (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49.
13. X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh, "Experimental comparison of representation methods and distance measures for time series data," *Data Mining and Knowledge Discovery*, vol. 26, no. 2, pp. 275–309, 2013.
14. A. Bagnall and J. Lines, "An experimental evaluation of nearest neighbour time series classification. technical report #CMP-C14-01," Department of Computing Sciences, University of East Anglia, Tech. Rep., 2014.
15. Lexiang Ye and Eamonn Keogh (2009) Time Series Shapelets: A New Primitive for Data Mining, *ACM SIGKDD*, 2009
16. P Senin and S Malinchik, SAX-VSM:Interpretable Time Series Classification Using SAX and

Vector Space Model, *IEEE ICDM*, 2013.
17. Yi Zheng, Qi Liu, Enhong Chen, Yong Ge and J. Leon Zhao, "Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks," *WAIM*, 2014.
18. Barbedo JGA, Lopes A, Automatic genre classification of musical signals. *EURASIP Journal on Advances in Signal Processing*, 2007.
19. Eisner R, Poulin B, Szafron D, Lu P, Greiner R (2005) Improving protein function prediction using the hierarchical structure of the gene ontology. In: Proc. of the IEEE Symp. on Computational Intelligence in Bioinformatics and Computational Biology, pp 1–10
20. Secker A, Davies M, Freitas AA, Clark E, Timmis J, Flower DR (2010) Hierarchical classification of g-protein-coupled-receptors with data-driven selection of attributes and classifiers. International Journal of Data Mining and Bioinformatics 4(2):191–210
21. Clare A, King RD (2003) Predicting gene function in saccharomyces cerevisiae. Bioinformatics 19:ii42–ii49, suppl. 2
22. Kiritchenko S, Matwin S, Famili AF (2005) Functional annotation of genes using hierarchical text categorization. In ACL BioLINK.
23. Silla Jr, Carlos N., and Alex A. Freitas. "A survey of hierarchical classification across different application domains." Data Mining and Knowledge Discovery 22.1-2 (2011): 31-72.
24. E. Keogh. Exact indexing of dynamic time warping. Int. Conf. on Very Large Data Bases, pp 406-417, 2002.
25. D Lemire, Faster retrieval with a two-pass dynamic-time-warping lower bound, Pattern Recognition, Volume 42, Issue 9, 2009.
26. M Muja and DG Lowe, Scalable NN algorithms for high dimensional data, *Pattern Analysis and Machine Intelligence*, 2014.
27. F. Petitjean, G. Forestier, G. Webb, A. Nicholson, Y. Chen and E. Keogh, "Dynamic Time Warping Averaging of Time Series allows Faster and more Accurate Classification," *IEEE ICDM*, 2014.
28. P Kasarapu and L Allison. "Minimum message length estimation of mixtures of multivariate Gaussian and von Mises-Fisher distributions," *Machine Learning*, 2015.
29. T Rakthanmanon et al. "Searching and mining trillions of time series subsequences under dynamic time warping" *ACM SIGKDD*, 2012
30. Jon Louis Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM, 18 (1975), pp. 509-517.
31. François Petitjean, Alain Ketterlin, and Pierre Gançarski, A global averaging method for dynamic time warping, with applications to clustering, Pattern Recognition, 44 (2011), pp. 678{693.
32. David Arthur and Sergei Vassilvitskii, k-means++: The advantages of careful seeding, in Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007, pp. 1027-1035.
33. CA Ratanamahatana and E Keogh, Three myths about DTW data mining, in SIAM SDM, 2005, pp. 506-510.
34. X Xi, E Keogh, C Shelton, L Wei, and CA Ratanamahatana, Fast time series classification using numerosity reduction, in ICML, 2006, pp. 103-1040.
35. DF Silva and GE Batista, Speeding up all-pairwise dynamic time warping matrix calculation, in SIAM SDM, 2016, pp.837-845.
36. T Rakthanmanon, B Campana, A Mueen, G Batista, B Westover, Q Zhu, J Zakaria and E Koegh, Searching and mining trillions of time series subsequences under DTW, in ACM SIGKDD, 2012, pp. 262-270.
37. E Keogh and CA Ratanamahatana, Exact indexing of dynamic time warping, Knowledge and information systems, 7 (2005), pp. 357-386.
38. D Lemire, Faster retrieval with a two-pass dynamic-time-warping lower bound, Pattern recognition, 42 (2009), pp. 2169-2180.
39. Bojinski, S., M. Verstraete, T.C. Peterson, C. Richter, A. Simmons, and M. Zemp, 2014: The

Concept of Essential Climate Variables in Support of Climate Research, Applications, and Policy. Bull. Amer. Meteor. Soc., 95, 1431–1443, https://doi.org/10.1175/BAMS-D-13-00047.1
40. CW Tan, GI Webb and F Petitjean, "Indexing and classifying gigabytes of time series under time warping," SIAM International Conference on Data Mining, 2017. http://bit.ly/ScalableTSClassification
41. CW Tan, M Herrmann, G Forestier, GI Webb and F Petitjean, "Efficient search of the best warping window for Dynamic Time Warping," SIAM International Conference on Data Mining, 2018. https://bit.ly/2FGtJXu

List of Publications and Significant Collaborations that resulted from your AOARD supported project:

The core publications for this project are the two papers published in the SIAM SDM Conference, respectively in 2017 and 2018. Our SDM'18 paper was awarded **Best Research Paper**. Other papers are works where work carried out for this grant led to ideas supporting subsidiary works. Note Chang-Wei TAN is my PhD student.

In standard format showing authors, title, journal, issue, pages, and date, for each category list the following:

a) papers published in peer-reviewed journals,

- H.A. Dau, D.F. Silva, **F. Petitjean**, G. Forestier, A. Bagnall, A. Mueen, E. Keogh, Optimizing dynamic time warping's window width for time series data mining applications, *Data Mining and Knowledge Discovery, in press, 2018.*
- N.A. Zaidi, G.I. Webb, M.J. Carman, **F. Petitjean**, W. Buntine, M. Hynes, H. de Sterck, Efficient Parameter Learning of Bayesian Network Classifiers, *Machine Learning*, Vol. 106, Num. 9-10, pp. 1289-1329, 2017.
- G. Forestier, **F. Petitjean**, L. Riffaud, P. Jannin, Automatic matching of surgeries to predict surgeons' next actions, *Artificial Intelligence in Medicine*, Vol. 81, pp. 3-11, 2017.
- G. Forestier, **F. Petitjean**, P. Senin, L. Riffaud, P.-L. Henaux and P. Jannin, Finding discriminative and interpretable patterns in sequences of surgical activities, *Artificial Intelligence in Medicine*, Vol. 82, pp. 11-19, 2017.

b) papers published in peer-reviewed conference proceedings,

- C.W. Tan, G.I. Webb and **F. Petitjean**. Indexing and classifying gigabytes of time series under time warping. *SIAM International Conference on Data Mining*, 2017.
- G. Forestier, **F. Petitjean**, H.A. Dau, G.I. Webb and E. Keogh. Generating synthetic time series to augment sparse datasets. *IEEE International Conference on Data Mining*, 2017.
- H.A. Dau, D.F. Silva, **F. Petitjean**, G. Forestier, A. Bagnall, A. Mueen, E. Keogh, Judicious Setting of Dynamic Time Warping's Warping Window Width allows more Accurate Classification of Time Series, *IEEE Big Data, 2017.*
- C.W. Tan, M. Herrmann, G. Forestier, G.I. Webb and **F. Petitjean**. Efficient search of the best warping window for Dynamic Time Warping. *SIAM International Conference on Data Mining*, 2018, Best Paper Award.

c) papers published in non-peer-reviewed journals and conference proceedings,
d) conference presentations without papers,
e) manuscripts submitted but not yet published, and
f) provide a list any interactions with industry or with Air Force Research Laboratory scientists or significant collaborations that resulted from this work.

Attachments: Publications a), b) and c) listed above if possible.