Using Navigable Small Worlds to Speed Up Time-Series Classification

Vivek Mahato and Pádraig Cunningham

School of Computer Science
University College Dublin
Dublin 4, Ireland
vivek.mahato@ucdconnect.ie, padraig.cunningham@ucd.ie

Abstract. It is not easy to apply standard Machine Learning methods to time-series classification because of the nature of the data. k-Nearest Neighbour (k-NN) can be applied because effective distance measures for time-series are available (e.g. Dynamic Time Warping (DTW)). However, these effective measures are inclined to be quite computationally expensive – DTW is $O(t^2)$ in the length of the time-series. This is a problem because the resultant retrieval time is $O(nt^2)$ when n is the number of training samples. Methods to improve k-NN retrieval performance exist, but some constraints apply. Some methods require that the distance measure is a proper metric; others are only effective in low dimension spaces. In this paper, we evaluate the effectiveness of Navigable Small Worlds (NSW) for time-series classification using k-NN. NSW organises training samples into a small-world network for efficient retrieval. We show that this strategy can significantly speed up time-series classification at the cost of some loss of accuracy.

Keywords: Navigable Small Worlds, Time Series Classification, Nearest Neighbour

1 Introduction

Time-series classification is a particular challenge for supervised machine learning because the data is not in a feature vector format. A comprehensive evaluation of ML methods for time-series classification [3] shows that k-NN is very effective provided a suitable distance measure can be identified. However, k-NN is not a perfect panacea because effective distance measures for time-series data are computationally expensive. Methods to speed up k-NN retrieval (both exact and approximate) that exists have constraints attached. Speedup methods typically require that the distance measure is a proper metric. Some time-series measures (e.g. DTW) do not meet the triangle inequality requirement to be a metric. Also, the curse of dimensionality cannot be avoided; it is hard to speed up k-NN retrieval in high dimension spaces and preserve accuracy.

In this paper, we assess the potential of using Navigable Small Worlds for k-NN in time-series classification. With NSW, the data is organised into a small-

world graph that facilitates efficient retrieval. Small-world networks have the special characteristic that any pair of nodes is likely to be linked by a small number of hops. This property is due to the presence of *hubs* in the network. Hubs are high-degree nodes that serve as a routing backbone that delivers short shortest path distances. With NSW this small-world characteristic arises naturally from the way the network is constructed (see details in section 4.1).

So NSW can be used as a data-structure to provide approximate k-NN retrieval (see Figure 3). While most of the research on NSW for approximate nearest neighbour retrieval is based on proper metrics, the authors do claim that it might also be effective for non-metric spaces [11]. In this paper, we evaluate three time-series distance measures in combination with NSW. We consider DTW, which is not a metric and Time-Warp Edit Distance (TWED) which is. The results in section 5 show that NSW, in combination with DTW, can reduce retrieval time by a factor of four. The price that is paid for this is a reduction in accuracy of roughly 2% to 3%.

In the next section, we provide an overview of k-NN and time-series classification illustrating the computational challenges. In section 3, we review relevant research on k-NN speedup. In section 4, we introduce NSW and show how it can be used for k-NN retrieval. Our results on using NSW to speed up k-NN time-series classification are presented in section 5.

2 *k*-NN and Time-Series Classification

k-NN works by finding the nearest neighbours for a query $\mathbf{q}$ in training set D and then a class is assigned to $\mathbf{q}$ using these neighbours, typically by majority voting. If the data is numeric, D is made up of samples of the form $\mathbf{x} = \{x_1, ..., x_j, ..., x_m\}$. For each $\mathbf{x} \in D$ the Euclidean distance between $\mathbf{q}$ and $\mathbf{x}$ is:

$$d(\mathbf{q}, \mathbf{x}) = \sqrt{\sum_{i=1}^{m} (\mathbf{q}_i - \mathbf{x}_i)^2} \tag{1}$$

If two time-series are the same length, then Euclidean distance *can* be used as a distance measure. However, as we see in Figure 1, if the two time-series are a little out of phase, then the Euclidean distance can be misleading. The vertical lines in the plot on the left indicate this. So there has been extensive research on similarity/distance measures for time-series [10]. In this paper, we consider three measures:

- **DTW** allows the time-axes to be *warped* to allow for more flexible matching [8]. It is $O(t^2)$ in the length of the time-series and it is not a proper metric.
- **TWED** supports inexact matching of time-series by counting edit operations as is done in string matching [12]. It is $O(t^2)$ in the length of the time-series but *is* a proper metric. As such it is a good candidate to incorporate into an NSW retrieval framework.
- **Euclidean Distance** is included as a baseline. It is $O(t)$ and is of course a proper metric.

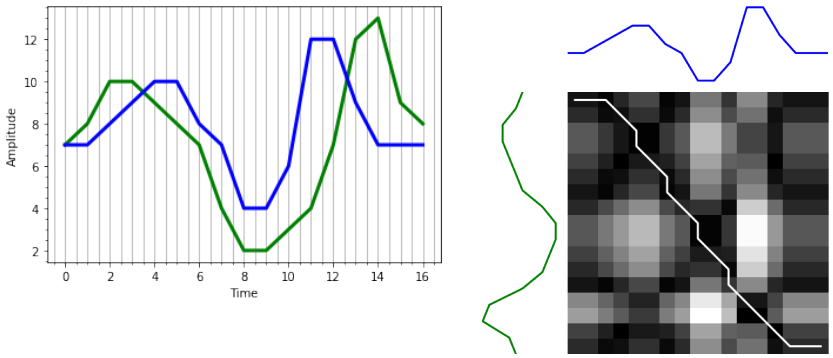


Fig. 1: On the left, we have two time-series that are clearly similar, but the Euclidean distance between them is large (vertical lines in the graph). The image on the right shows the non-linear mapping by DTW between these two time-series (produced using tslearn[14]).

Table 1: 3D printing datasets

Dataset	Train	Test	Total
P_{Large}	1,794	1,793	3,587
P_{Small}	1,384	1,382	2,766

2.1 Time-Series Classification in Manufacturing

In our research, we are particularly interested in the use of time-series classification for process monitoring in manufacturing. We are concerned with the use of time-series classification to detect processing anomalies in metal 3D printing [9]. It is normal to monitor the 3D printing process using sensors. These sensors may monitor conditions in the 3D printing chamber, or they may focus on the melt-pool itself. To evaluate NSW for time-series classification speedup, we consider two datasets collected from an Aconity MINI 3D printer.[1]

The details of the two datasets are summarised in Table 1. Each data sample is a time-series of melt-pool temperature recorded by a pyrometer. The classification task is to detect pores in the part being manufactured. These pores would be rare but would typically show up as dips in the temperature time-series. The longest time-series in both datasets contain 300 time points and the shortest 101. When calculating distance the shorter series is post-padded with zeroes to bring it to the length of the longer one. A trailing zero is added to the longer series to allow matching with this padding.

To compare the time complexity of the three distance measures, we computed the average retrieval time of 10 random queries in a k-NN environment using

[1] https://aconity3d.com/products/aconitymini/

the P_{Large} dataset. The average retrieval time for each measure (given best parameters) per query are as follows:

- Euclidean[2]: 0.02 s
- DTW[3]: 1.46 s
- TWED[4]: 10.16 s

As expected, Euclidean distance is the fastest by two orders of magnitude, DTW the second, and TWED the slowest of them all.

3 *k*-NN Speedup Strategies

These baseline calculations illustrate the run-time problems with k-NN. The performance with Euclidean distance might be acceptable, but the DTW and TWED figures are not adequate for real-time analysis. This is a recognised problem with k-NN and has been the subject of extensive research. This research can be categorised under exact methods and approximate methods. Exact methods will guarantee that the neighbours retrieved by brute force search will be precise. Approximate methods don't meet this requirement.

Exact Methods The two most popular alternatives to brute force search for exact k-NN are Ball Trees and Kd-Trees [4] but their applicability is constrained. Kd-Trees can only be used with feature-vector data and are thus not applicable for time-series data. Ball Trees require a distance measure that is a proper metric. Both strategies cease to be effective in high dimension spaces [1, 13].

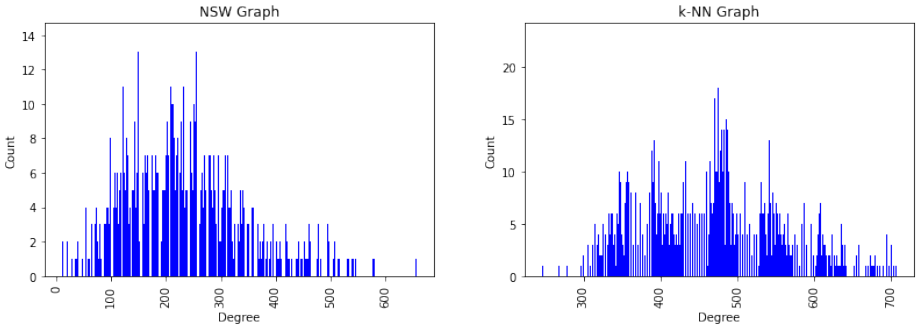
Approximate Methods Efficient k-NN retrieval is an important consideration, particularly in internet applications such as image or music retrieval. As such, the focus has shifted to consider approximate nearest neighbour (ANN) methods [6, 2]. Approximate methods may be adequate where the objective is information retrieval rather than classification. For nearest neighbour search over collections with tens or even hundreds of thousands of objects perfect recall is not essential. ANN strategies can be divided into three broad categories:

- **Random Projection Trees (RPT).** The main drawback with Kd-Trees is that search through the tree entails a lot of backtracking to ensure that the nearest neighbours are retrieved. RPT does not backtrack; instead, the search is repeated multiple times and the initial candidates identified each time are assessed to find good nearest neighbours [7].
- **Locality Sensitive Hashing (LSH)** maps similar items into the same 'buckets' with high probability. The strategy is to use several variants of LSH algorithms to retrieve a candidate set of nearest neighbours. Then the distance metric can be applied to these candidates to find nearest neighbours that will be near-optimal [5].

[2] https://pypi.org/project/numpy/
[3] https://pypi.org/project/tslearn/
[4] https://en.wikipedia.org/wiki/Time_Warp_Edit_Distance

Table 2: Network Statistics

Network	Radius	Diameter	Avg. Hops
k-NN	5	9	3.18
NSW	3	7	2.76



(a) NSW network on P_{Large} dataset. (b) k-NN network on P_{Large} dataset.

Fig. 2: The in-degree distribution of a NSW & k-NN network on P_{Large} dataset

– **Neighbourhood-Based Methods** index the data by building a proximity graph where data points are linked to their neighbours. Retrieval entails navigating this proximity graph to find nearest neighbours for a query. NSW is such a method but with the special characteristic that the graph is a small-world graph (see section 4). A recent evaluation by Li *et al.* [6] shows that these methods are competitive and, because they can be combined with DTW and TWED; this is the ANN strategy we consider.

4 NSW & *k*-NN

NSW is a neighbourhood-based method for ANN that constructs a proximity graph that has small-world characteristics. This is illustrated in Figure 2 and Table 2. We compare an NSW graph with a standard k-NN graph. The k-NN graph is constructed by connecting each node to its $f = 233$ nearest neighbours (parameter setting details are in section 4.2). The NSW graph is constructed using the strategy detailed in section 4.1. The statistics in Table 2 show that the NSW graph is more compact than the k-NN graph; this is due to the high-degree *hub* nodes on the right in Figure 2(a).

4.1 NSW in Operation

NSW uses a small-world network that can be represented by a graph $G(V, E)$, where vertices from set V are the stored elements, and edges from set E are

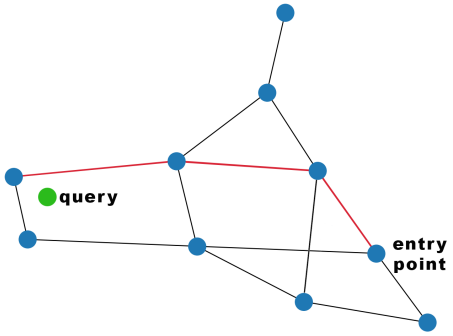


Fig. 3: Navigable Small World (NSW): The nearest neighbours to the query would be found from the entry point along the path shown in red.

connections linking them with other vertices. The NSW structure is constructed by inserting elements into the network consecutively[11]. For each insertion of an element, the algorithm first searches for a set of its nearest neighbours already present in the network (see Algorithm 1).

Algorithm 1 Insertion Algorithm

procedure NN_INSERT($element, f, m$) ▷ m multi-searches to find f neighbours to connect with element
  $neighbours \leftarrow NN\_Search(element, f, m)$
  for $neighbour$ *in* $neighbours$ **do**
    $element.connect(neighbour)$
    $neighbour.connect(element)$
    Store the new and updated elements in the network

NSW employs a variant of the greedy search algorithm as its core k-NN search to fetch approximate neighbours of the object. At first, a random node is selected from the network as an entry point (see Figure 3), and then search progress by traversing the network from one element to its unvisited nearest neighbouring element. The exploration continues within the neighbourhood of the nearest objects in a greedy style until it no longer improves the already known k nearest neighbours (stop condition) or has an unvisited object left to explore (see Algorithm 2). On identifying a set of neighbours, the algorithm inserts the element into the network by connecting it to its neighbours and vice versa. This incremental strategy has the important side-effect that long-distance links are created early on in the construction process because true nearest neighbours are not present. This results in small-world characteristics.

We used ValueSortedDict from the sortedcollections[5]package in the PyPI repository, as the data-structure for our implementation of NSW for storage and re-

trieval of neighbours. This significantly improves performance because there is a lot of sorting required for network traversal.

Algorithm 2 Search Algorithm

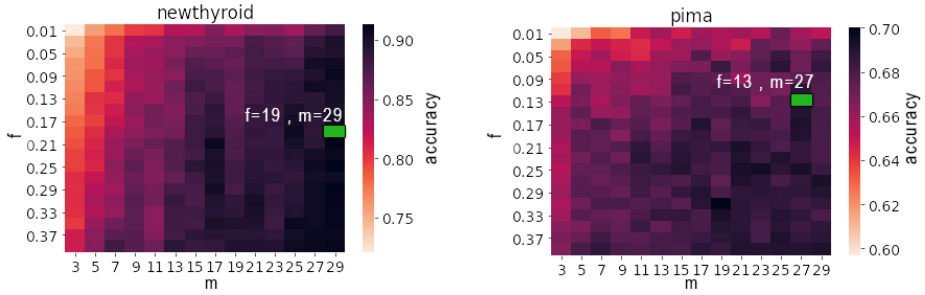
```
procedure NN_Search(query, k, m)  ▷ m searches to find k neighbours of a query
    Set visited_set                                  ▷ maintains list of visited nodes
    ValueSortedDict candidates, result
    for iteration ≤ m do
        v_ep ← random_entry_point                    ▷ Fetch a random node from the network
        visited_set, candidates ← v_ep
        ValueSortedDict temp_result
        while True do
            candidate ← nearest_candidate(candidates)        ▷ from candidates pool
            if stop_condition = False and candidate has neighbours then
                for neighbour in candidate.neighbours do
                    if neighbour not in visited_set then
                        cost ← distance(query, neighbour)
                        visited_set, candidates, temp_result ← neighbour
            else
                break while loop
        result ← temp_result                         ▷ result is updated with temp_result
    Return k nearest neighbours from result
```

Two parameters f and m dictate the performance of the NSW algorithm. f determines the maximum number of "friends" or neighbours, a vertex can have in the network, and m dictates multi-search. m is directly proportional to the recall of the system, where a high enough m results in the perfect recall; however, it is then similar to an exhaustive search. Hence, there is a trade-off between classification accuracy and the speed-up accomplished by the model, governed by its parameters.

4.2 NSW Performance

Before applying NSW to time-series data, we present a baseline analysis on feature vector data. For this, we have selected six public datasets from the KEEL[6] repository for classification tasks. As previously mentioned, selecting optimal values for f and m is essential, owing to the trade-off between accuracy and time complexity of the NSW model. The idea is to obtain a high accuracy while keeping f and m small for a quicker retrieval time. This is achieved through a grid-search, as shown in Figure 4. When supplied with a pair of values, the models undergo 5×5-fold cross-validation over the train set of a dataset, with accuracy as its scoring method. The heatmap generated illustrates the effect of

[5] https://pypi.org/project/sortedcollections/
[6] https://sci2s.ugr.es/keel/datasets.php



(a) Parameter heatmap of NSW over *newthyroid* dataset.

(b) Parameter heatmap of NSW over *pima* dataset.

Fig. 4: These heatmaps show the impact of the f and m parameters on accuracy when using NSW. The selected parameters are highlighted in green.

f and m on the accuracy of the model. Fig. 4 show heatmaps for two datasets. It is clear that higher values of f and m give better accuracy, but it is possible to find lower values that achieve high accuracy while achieving significant speedup.

The feature selection process is mostly automated, with these heatmaps allowing the user to select an optimal pair of parameter values for the dataset. The final evaluation of the model is over the held-out test set of the dataset provided with the selected pair as its parametric values. The evaluation results on the six non-time-series datasets are shown in Figure 5. Over the six datasets, the processing time is reduced to 42% on average. There is no reduction in classification accuracy for three of the datasets. The average reduction in accuracy is $\approx 1.3\%$. The result illustrates NSW to be a promising strategy to achieve accuracy of full k-NN comparably and much quicker.

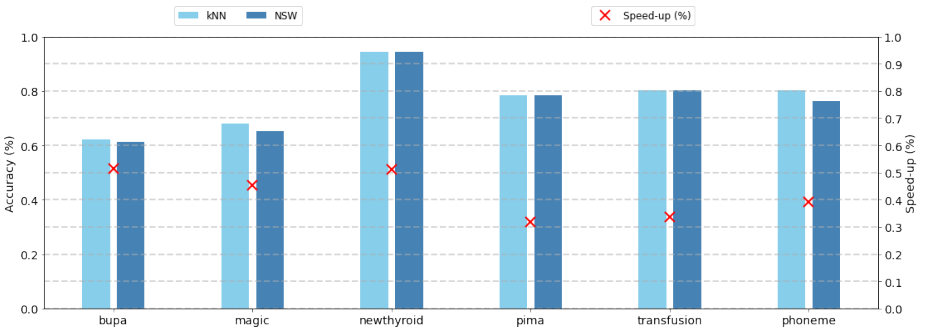
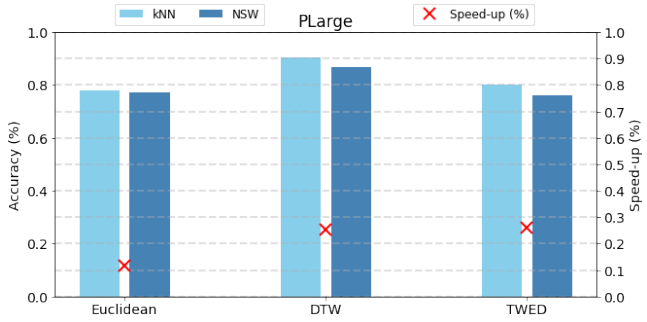


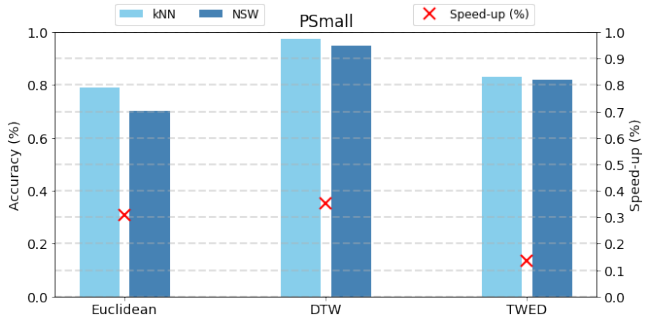
Fig. 5: kNN vs. NSW with Euclidean as their distance metric over six non time-series datasets. The bars represent the accuracy of the model, and the 'red' $\times$'s denotes the speed-up (which is a percentage of the total time taken by k-NN model).

5 NSW in TS Classification

Building upon our preliminary analysis discussed in 4.2, we extend the evaluation of k-NN and NSW models to classification tasks on time-series datasets (P_{Large} and P_{Small}). The final results reported in Figure 6 are on hold-out test sets as described in Table 1. The f and m parameters were tuned using 5×5-fold cross-validation on the training set. The value of k for exact k-NN was set in the same manner. The same k value was used with NSW.



(a) Model evaluation over P_{large} dataset.



(b) Model evaluation over P_{small} dataset.

Fig. 6: k-NN vs. NSW: Evaluation over classification accuracy and time complexity. The "speed-up" gained by NSW is the percentage of total time taken by the k-NN counterpart.

The main conclusions from this evaluation are as follows:

1. Even if TWED is a proper metric, DTW is still the clear winner by achieving the highest accuracy for both exact k-NN and using NSW.
2. Using NSW, the processing time is reduced to 23.5% for P_{large} and 35% for P_{small} of that of exact k-NN-DTW. The speedup is more significant for the larger dataset.

3. NSW achieves accuracy comparable to that of exact k-NN. The drop in accuracy using NSW with DTW measure is 2.7% on P_{large} and 4.2% for P_{small}.

In summary, the speedup is significant – it seems reasonable to expect better than a four-fold speedup on large datasets. However, the impact on accuracy would probably not be acceptable in many situations.

6 Conclusions & Future Work

In this paper, we address the challenge of improving the run-time performance of time-series classification using k-NN. Because effective distance measures for time-series are computationally expensive, we explore the potential of NSW for speeding up k-NN using DTW and TWED. TWED is included because it is a proper metric, whereas DTW is not. Despite its lack of metric pedigree, DTW performs best. NSW delivers significant speedup but at the cost of classification accuracy.

So it seems that NSW does not allow us to avoid the curse of dimensionality. We cannot speed up retrieval without paying the price in terms of accuracy.

In future work, we will explore measures to quantify the intrinsic dimension of time-series data to see if speedup strategies might work for datasets that have a lower intrinsic dimension. Furthermore, these methods need to be evaluated in a realistic context where time-series containing pores would be very rare.

Acknowledgements

This publication has resulted from research supported in part by a grant from Science Foundation Ireland (SFI) under Grant Number 16/RC/3872 and is co-funded under the European Regional Development Fund.

References

1. Arya, S., Mount, D.M., Narayan, O.: Accounting for boundary effects in nearest-neighbor searching. Discrete & Computational Geometry 16(2), 155–176 (1996)
2. Aumüller, M., Bernhardsson, E., Faithfull, A.: Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In: International Conference on Similarity Search and Applications. pp. 34–49. Springer (2017)
3. Bagnall, A., Bostrom, A., Large, J., Lines, J.: The great time series classification bake off: An experimental evaluation of recently proposed algorithms. extended version (2016)
4. Cunningham, P., Delany, S.J.: k-nearest neighbour classifiers 2nd edition (with python examples). arXiv preprint arXiv:2004.04523 (2020)
5. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 604–613 (1998)

6. Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., Lin, X.: Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement. IEEE Transactions on Knowledge and Data Engineering (2019)
7. Liu, T., Moore, A.W., Yang, K., Gray, A.G.: An investigation of practical approximate nearest neighbor algorithms. In: Advances in neural information processing systems. pp. 825–832 (2005)
8. Mahato, V., Johnston, W., Cunningham, P.: Scoring performance on the y-balance test. In: International Conference on Case-Based Reasoning. pp. 281–296. Springer (2019)
9. Mahato, V., Obeidi, M.A., Brabazon, D., Cunningham, P.: An evaluation of classification methods for 3d printing time-series data. arXiv preprint arXiv:2005.09052 (2020)
10. Mahato, V., O'Reilly, M., Cunningham, P.: A comparison of k-nn methods for time series classification and regression. In: 26th Irish Conference on Artificial Intelligence and Cognitive Science (AICS) (2018)
11. Malkov, Y., Ponomarenko, A., Logvinov, A., Krylov, V.: Approximate nearest neighbor algorithm based on navigable small world graphs. Information Systems 45, 61–68 (2014)
12. Marteau, P.F.: Time warp edit distance with stiffness adjustment for time series matching. IEEE transactions on pattern analysis and machine intelligence 31(2), 306–318 (2008)
13. Ponomarenko, A., Malkov, Y., Logvinov, A., Krylov, V.: Approximate nearest neighbor search small world approach. In: International Conference on Information and Communication Technologies & Applications. vol. 17 (2011)
14. Tavenard, R., Faouzi, J., Vandewiele, G., Divo, F., Androz, G., Holtz, C., Payne, M., Yurchak, R., Rußwurm, M., Kolar, K., et al.: Tslearn, a machine learning toolkit for time series data. Journal of Machine Learning Research 21(118), 1–6 (2020)