EasyChair Preprint

№ 14238

Enhancement of the Local Outlier Factor Algorithm for Anomaly Detection in Time Series

Daniel Barrish and Jan van Vuuren

August 1, 2024

Enhancement of the local outlier factor algorithm for anomaly detection in time series

Daniel Barrish[1][0009−0007−3528−5869] and
Jan van Vuuren[2][0000−0003−4757−5832]

[1] Stellenbosch Unit for Operations Research in Engineering, Department of Industrial Engineering, Stellenbosch University, South Africa
22941096@sun.ac.za

[2] Stellenbosch Unit for Operations Research in Engineering, Department of Industrial Engineering, Stellenbosch University, South Africa
vuuren@sun.ac.za

Abstract. A plethora of methodological approaches have been proferred in recent years within the topical field of time series anomaly detection. Despite many advancements in adjacent fields, the well-known local outlier factor algorithm has stood the test of time in this domain, with recent comparative studies indicating that it remains competitive with newer approaches in benchmark tests. In this paper, we enhance this algorithm by leveraging ensembling techniques and GPU (graphics processing unit) acceleration. To the best of our knowledge, our ensembling approach establishes a new state-of-the-art accuracy of 66.8% in respect of the well-known UCR Time Series Anomaly Detection benchmark, while our GPU implementation was approximately eleven times faster than our CPU (central processing unit) baseline.

Keywords: Anomaly detection · Local outlier factor algorithm · Time series · Machine learning

1 Introduction

Time series anomaly detection is a topical field, with diverse applications in domains such as medicine, finance, predictive maintenance, and intrusion detection, to name but a few. Every year, dozens of new approaches are proposed for this pivotal task, but the opacity of benchmark data sets, metrics, and methodologies has made their fair comparison a complex task.

In a recent study [4], it was shown that the well-established local outlier factor algorithm is at least competitive with newer approaches to time series anomaly detection, even without extensive hyperparameter tuning. This means that it should be possible to improve its performance, at least marginally, with a few simple enhancements. In order to verify this claim, we carried out a sensitivity analysis on the two key hyperparameters of the sub-sequence local outlier factor algorithm (namely window size and number of neighbours), and determined that the accuracy of the algorithm depends highly on the hyperparameter values

selected. The diverse predictive behaviour of various hyperparameter configurations indicated that a simple ensembling approach might be a promising avenue for improving accuracy.

Unfortunately, the computational demands of a large ensemble proved excessive, especially when considering that anomalies in time series typically must be detected in real time. As a result, we explored ways to speed up the local outlier factor algorithm. We settled for a simple, yet highly effective, approach of accelerating the k-nearest neighbours search (the computationally most expensive step of the local outlier factor algorithm) using a *graphics processing unit* (GPU), resulting in a speed-up of up to 1050% for large data sets.

This significant speed-up facilitated the testing of our ensemble model. Results indicated that our proposed ensemble is significantly more accurate than the best configuration of the local outlier factor algorithm, without being prohibitively time-consuming. Moreover, our ensembling approach also outperforms the state-of-the-art *discord aware matrix profile* (DAMP) algorithm (with sharpening) in terms of accuracy. We anticipate that additional modifications to the local outlier factor algorithm may improve its accuracy further, and we proffer a handful of avenues for future work in this regard.

This paper is structured as follows. Section 2 contains a description of the working of the local outlier factor algorithm, and this is followed in Section 3 by an outline of how it may be applied to time series data. In Section 4, we analyse and compare the performance of various hyperparameter configurations to motivate the rationale behind our ensembling approach. Thereafter, we demonstrate in Section 5 how GPUs may be used to accelerate the algorithm, while Section 6 is devoted to an explanation of our proposed ensembling approach towards improving prediction accuracy. Our experimental results are presented and discussed in Section 7, and some conclusions are finally drawn in Section 8, which also contains some suggestions for future work.

2 Background

In this section, a brief overview is provided of the field of time series anomaly detection. This is followed by an exposition of the local outlier factor algorithm and its descendants, as well as how the algorithm may be applied to time series data.

2.1 Anomaly detection in time series

The goal of time series anomaly detection is, simply put, to identify patterns or events in time-indexed data that deviate significantly from the expected or normal behavior. More formally, we define a time series as an array of the form

$$T = [t_1, t_2, \ldots, t_n]$$

where t_i is a real-valued number (in the case of univariate time series) or a real-valued vector (in the case of multivariate time series) at timestep $i \in \{1, \ldots, n\}$.

An anomaly detection algorithm should ingest the time series T and output a subset of the indices $\{1, \ldots, n\}$ corresponding to the most anomalous timesteps. While our focus in this paper is on *static univariate* time series anomaly detection for illustrative purposes, the general approach that we propose should conceivably be generalisable to the multivariate case.

One of the main challenges in the field of anomaly detection is that there is no universal definition of what is meant by the notion of an anomaly. More specifically, in order for a point to be considered an anomaly, we do not know *by how much* it should deviate from the norm, and based on *which measure*. Moreover, since anomalies are inherently scarce, labelled examples are usually few and far between. This means that we rarely have examples of anomalies when training our models, making this an unsupervised learning task.

The anomaly detection process is generally the same for both tabular and time series data, and consists of two steps as articulated by Keogh [10]. These steps are as follows:

1. The most anomalous points in the data set are identified. This involves employing an algorithm which calculates an *anomaly score* for each data point—typically ranging from 0 (representing the least anomalous data points) to 1 (representing the most anomalous data points).
2. A threshold is determined which segregates normal data points from anomalies. Data points with an anomaly score greater than the threshold are flagged as anomalies.

The first step (anomaly scoring) is generally considered to be the trickier of the two, while the second step (thresholding) depends heavily on the context or use case. As a result, most time series anomaly detection research has focused on developing improved approaches towards anomaly scoring. To this end, hundreds of algorithms have been proposed in the literature, rendering time series anomaly detection a minefield for practitioners, as there are an abundance of approaches to choose from without having many consistent, high-quality benchmarks in terms of which to compare them.

2.2 The original local outlier factor algorithm

One of the most prominent, and celebrated, anomaly scoring approaches is the local outlier factor algorithm, developed in 2000 by Breunig *et al.* [6]. Although it is a relatively uncomplicated algorithm, especially relative to more recent deep learning methods, it has stood the test of time. Not only is the algorithm relatively fast and highly parallelisable, but it also enjoys the major advantage of not requiring any training. In other words, the algorithm may simply be applied to test data.

The local outlier factor algorithm is based upon the intuitive notion that one may be able to identify anomalies by comparing the density of a data point with those of neighbouring data points. The crux of the local outlier factor algorithm lies in finding an appropriate set of k nearest neighbours, as these neighbours

are used to calculate the density of the point in question as well as the so-called local outlier factor. Points that exhibit a significantly lower density than those of surrounding points may be considered anomalous.

The algorithm may be summarised in three steps:

1. A so-called *reachability distance* is calculated between a given data point and the points in its neighbourhood.
2. These reachability distances are used to compute the *local reachability density* of the given data point.
3. The local reachability density of the given data point is compared with the local reachability densities of its neighbours in order to obtain the local outlier factor of the given data point.

The local outlier factor algorithm requires one key parameter to be set, which is the number of nearest neighbours to be considered, denoted by k. Using this parameter, $\texttt{kDistance}(x)$ is defined as the distance (typically Euclidean) between a given data point x and its kth nearest neighbour. Based on the definition of this k-distance, the *reachability distance* between x and another point y is defined by

$$\texttt{RD}_k(x, y) = \max\{\texttt{kDistance}(y),\ \texttt{dist}(x, y)\}, \tag{1}$$

where $\texttt{dist}(x, y)$ denotes the distance between x and y.

All points at most $\texttt{kDistance}(x)$ away from x are considered to be within its neighbourhood, denoted by $\mathcal{K}$. Note that this definition means that the neighbourhood may consist of more than k data points. The *local reachability density* of a data point x is defined as

$$\texttt{LRD}_k(x) = \frac{|\mathcal{K}|}{\Sigma_{y\in\mathcal{K}}\texttt{RD}_k(x, y)}, \tag{2}$$

which is simply the inverse of the average reachability distance between x and points in its neighbourhood. The value in (2) is compared with the local reachability densities of the neighbours of x by computing the local outlier factor

$$\texttt{LOF}_k(x) = \frac{1}{|\mathcal{K}|}\Sigma_{y\in\mathcal{K}}\frac{\texttt{LRD}_k(y)}{\texttt{LRD}_k(x)} = \frac{\Sigma_{y\in\mathcal{K}}\texttt{LRD}_k(y)}{|\mathcal{K}|\ \texttt{LRD}_k(x)}, \tag{3}$$

which represents its density relative to these neighbouring points. The value in (3) may be interpreted as follows:

- A value close to 1 indicates that the point has a similar density to those of its neighbours.
- A value greater than 1 indicates that the point has a lower density than those of its neighbours, meaning that it is potentially an anomaly.
- Conversely, a value smaller than 1 indicates that the point has a higher density than those of its neighbours, meaning that it is an inlier.

Once calculated, the local outlier factor values in (3) can be mapped to the range $[0, 1]$ in order to align the anomaly scores with those of other popular anomaly detection algorithms.

It should be noted that the local outlier factor algorithm, as described here, was designed for tabular data, and is not suitable for application to time series data. This is due to the fact that the algorithm makes no allowance for a temporal relationship between successive timesteps. In Section 3, we discuss how the original local outlier factor algorithm may be adapted for time series data.

2.3 Descendants of the original algorithm

Numerous modifications and enhancements of the original local outlier factor algorithm have been proposed. In this section, we present a broad overview of some of the most prominent variants, although there are far too many for our coverage to be comprehensive.

Kriegel *et al.* [12] introduced the notion of *local outlier probabilities*, or LoOP for short. They sought to standardise the output of the local outlier factor algorithm to the range $[0, 1]$ in such a way that the score could be interpreted as the probability of a point being an anomaly. This does, however, come at the cost of increased computation time.

While the scope of this paper is limited to static data, it is useful to provide a brief perspective into the streaming context. A handful of proposed approaches were aimed at tailoring the local outlier factor algorithm for streaming data and enhance its efficiency. These approaches may be found in the work of Pokrajac *et al.* [22], Salehi *et al.* [24], and Li *et al.* [13], who introduced versions of the local outlier factor algorithm which were designed to work incrementally. Algushairy *et al.* [1], as well as Mishra and Chawla [17], have provided an overview and comparison of these streaming variants.

Alshawabkeh *et al.* [2] and Zhao *et al.* [28] recognised the value in accelerating the local outlier factor algorithm by using GPUs. Alshawabkeh *et al.* [2] discussed their broad approach and experimental results, while Zhao *et al.* [28] made their implementation available as well.

Other approaches have focused on enhancing the accuracy of the local outlier factor algorithm. For instance, Xu *et al.* [26] introduced an automated hyperparameter tuning method, while Cheng *et al.* [7] outlined a two-step ensemble method which uses the local outlier factor algorithm in conjunction with another popular anomaly detection algorithm, called isolation forest.

The local outlier factor algorithm has been applied to countless domains and use cases. Examples may be found in the work of Auskalnis *et al.* [3] (detecting intrusions in computer networks), Ma *et al.* [16] (detecting anomalies in traffic data), and Oehmcke *et al.* [20] (detecting special events in marine time series data).

2.4 Related work

The matrix profile, introduced by Yeh *et al.* [27] in 2016, is a data structure and family of algorithms which shares many similarities with the local outlier factor algorithm. The data structure consists of two vectors. The first is used to

store the z-normalised Euclidean distances between all sub-sequences in a time series and each sub-sequence's nearest neighbour, and the second is used to store the index of each nearest neighbour. Despite its apparent simplicity, the matrix profile may be used to solve a wide variety of time series data mining tasks, including motif discovery, classification, clustering, and anomaly detection.

Computing the matrix profile efficiently, however, was a major hurdle to its scalability until Zhu *et al.* [30] introduced the *scalable time series ordered-search matrix profile* (STOMP) algorithm, which is capable of successfully scaling to hundreds of millions of sub-sequences. Since then, additional modifications have made the computation of the matrix profile even more efficient.

Once computed, the matrix profile can be employed for time series anomaly detection. This typically involves finding the time series *discords*—sub-sequences of a time series that are maximally far from their nearest neighbour. A handful of algorithms have been proposed for this purpose, including MERLIN [18], MERLIN++ [19], and DAMP [14,15]. The latter algorithm, in particular, was shown to be highly effective. To the best of our knowledge, the DAMP algorithm with pre-processing is state of the art, and consequently this is the yardstick with which we have compared our proposed approach. Additionally, the local outlier factor algorithm shares its distance-based methodology, which makes their juxtaposition even more interesting.

3 Harnessing the algorithm for time series anomaly detection

Due to its ability to find *local* anomalies (and not just global outliers), the local outlier factor algorithm is well-suited to the domain of time series. Since time series data (typically) exhibit temporal dependencies (*i.e.* autocorrelation), it is vital that these data points are considered within the context of surrounding timesteps, instead of in isolation. The local outlier factor algorithm is demonstrably capable of detecting contextual anomalies in this sense.

It is crucial, however, that the local outlier factor algorithm is applied to *sub-sequences*, as opposed to individual points, since otherwise it would not be able to capture any temporal dependencies. As a result, a key pre-processing step involves applying a sliding window to the time series. Once the local outlier factor values have been calculated for these sub-sequences, the results may be aggregated to obtain anomaly scores for the individual points.

The size of the aforementioned sliding windows is a key parameter which has to be decided upon, as it may affect performance significantly. In a recent comparative study [4] in the context of a univariate time series anomaly detection benchmark data set, it was found that the local outlier factor algorithm is at least competitive with the best algorithms in the literature. Noting its stellar results without any hyperparameter tuning, the sub-sequence local outlier factor algorithm is a natural candidate for possible further improvements. In other words, it would be sensible to assume that the algorithm's accuracy can be

improved if appropriate values are selected for its hyperparameters—namely the window size and number of neighbours considered.

4 Sensitivity analysis in the context of time series

The logical next step would be to verify the above assumption, and also to ascertain *how significant* potential accuracy gains might be. To this end, we performed a simple sensitivity analysis on the sub-sequence local outlier factor algorithm.

4.1 Experimental setup

We used the UCR Time Series Anomaly Detection benchmark data set [10] exclusively, as a result of the issues raised by Wu *et al.* [25] concerning existing data sets, including mislabeled anomalies, overly simple anomalies, unrealistic anomaly densities, and a run-to-failure bias. By design, each of the 250 time series in the data set only contains a single anomaly, which means that the thresholding step of anomaly detection is not required and one can evaluate the effectiveness of the anomaly scoring step in isolation. As a result, the evaluation methodology is straightforward. The model makes a prediction, which is compared with the true anomaly, and then these results are aggregated in order to obtain an accuracy score as a percentage. We have made use of the same "wiggle room" of 100 timesteps as suggested by the data set creator [10]. This means that if the prediction is within 100 timesteps of the true anomaly, it is considered correct. This evaluation metric has been selected in lieu of the commonly-used precision and recall metrics, which have the tendency to inflate an algorithm's performance [8,9,11,19].

All experiments were performed on a personal computer equipped with an Intel Core i9-12900k CPU and NVidia RTX 3080 Ti GPU. The evaluation framework was written in Python, and we made use of scikit-learn's [21] implementation of the local outlier factor algorithm for our experiments in this section.

A set of sensible parameter combinations (for the window size and the number of nearest neighbours) were selected and their efficacy compared. For the window size w, we considered the values 10, 25, 50, 100, 250, and 500, and for the number of nearest neighbours k, we implemented the values 5, 10, 20, 50, and 100.

4.2 Results and discussion

The performance of each of the thirty hyperparameter combinations is illustrated graphically in Figure 1. It is evident, based on these results, that the window size and number of nearest neighbours hyperparameters have a significant impact on the accuracy of the algorithm. For instance, the best configuration solved 62.4% of the problem instances correctly, while the worst only managed 17.6%. This reinforces our claim for a need to select these hyperparameter values very carefully.

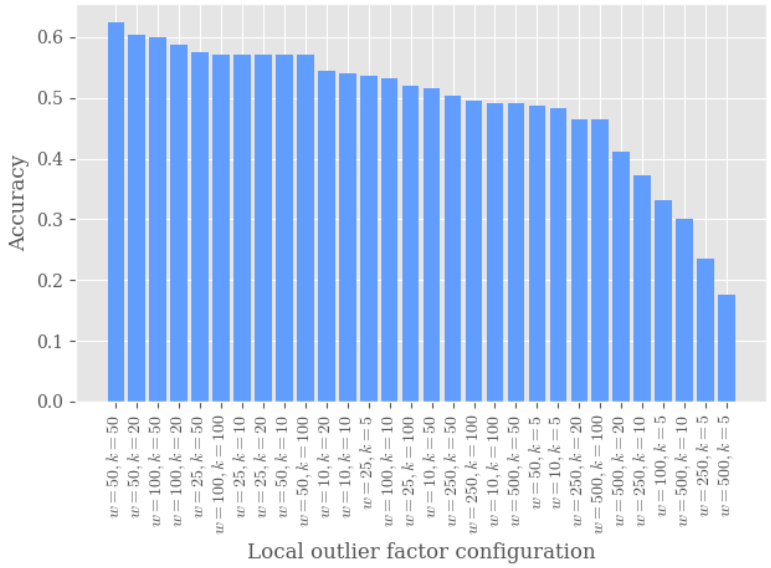

Fig. 1. A comparison of the accuracy of thirty hyperparameter combinations.

One may notice that configurations with a large window size tended to perform poorly. This is most likely due to the fact that large sliding windows tend to have a "smoothing" effect, making it difficult to distinguish true anomalies from normal points. It is also worth noting that small values of k tended to perform poorly. This may be explained by configurations with a small value of k being particularly susceptible to noise in the data set.

A natural follow-up question is whether the different predictive behaviours of various algorithm configurations have distinct strengths and weaknesses—in other words, whether the poorly performing configurations are capable of outperforming the best performing configurations in certain situations. The performance of each configuration on each problem instance is illustrated in Figure 2.

As may clearly be seen in Figure 2, certain algorithm configurations thrive in certain problem contexts, while others struggle—and *vice versa*. If one were able to select the most appropriate hyperparameter configuration for each problem instance, then a remarkable accuracy of at most 86% would be achievable—22% higher than the accuracy of the best configuration (although this upper bound is realistically unattainable). Moreover, the diverse predictive behaviour of the various configurations suggests that an ensemble might be suitable for enhancing prediction accuracy.

It is also worth noting which 35 problem instances *none* of the configurations were able to solve correctly. These were problems 1, 36, 40, 42, 45, 53, 57, 74, 78, 79, 80, 81, 82, 93, 104, 105, 107, 108, 109, 152, 153, 161, 182, 187, 188, 189, 190, 210, 211, 222, 230, 240, 241, 244, and 247. A deeper examination of

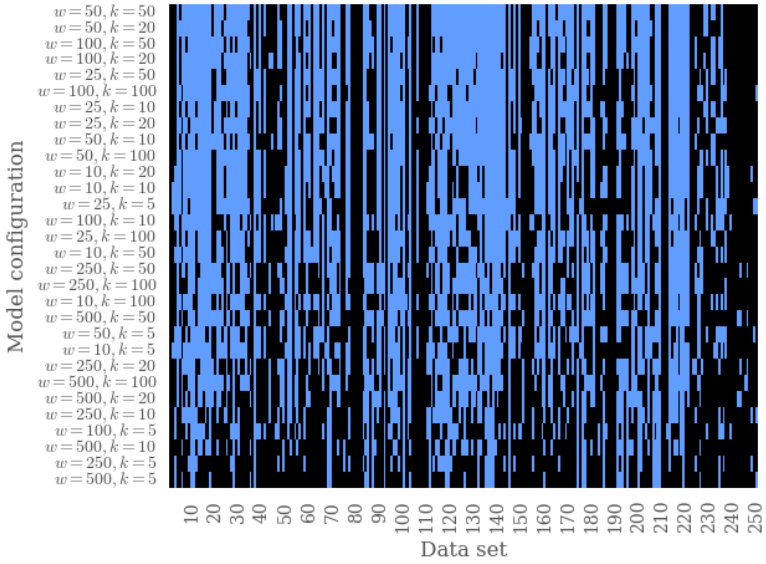

Fig. 2. A comparison of the performance of each configuration in respect of each benchmark problem instance, with the most accurate models at the top. Blue indicates that a problem instance was correctly solved, while black represents the opposite.

these problems might reveal potential shortcomings of the local outlier factor algorithm, and ways in which it could be improved.

5 Accelerating the algorithm

While the results above bode well in terms of potential accuracy improvements, techniques involving ensembling and/or hyperparameter tuning are most likely not computationally feasible if the goal is to detect anomalies in real time. As a result, it is evidently necessary to speed up the local outlier factor algorithm in order to reduce its runtime.

The utility of GPUs in significantly accelerating the training of neural networks is widely known, while their usefulness in speeding up many traditional learning algorithms is arguably somewhat neglected and underutilised. Although some research has been conducted on harnessing GPUs in conjunction with the local outlier factor algorithm, these papers unfortunately typically do not include implementations with source code. In our experimentation with one of the rare exceptions, PyTOD [28], the speed-up was not as significant as we had hoped, and the implementation ran into memory problems when applied to larger data sets. As a result, we sought to develop our own solution.

By far the computationally most expensive step of the local outlier factor algorithm is the calculation of the k-nearest neighbours for all of the data points

(or sub-sequences, in our case) in the data set. More specifically, in our experimentation on large data sets, up to 99.5% of the local outlier factor algorithm runtime was consumed by the k-nearest neighbours subroutine, as illustrated in Figure 3. As a result, it stands to reason that if the k-nearest neighbours subroutine were to be optimised, then the entire local outlier factor algorithm would be accelerated significantly.

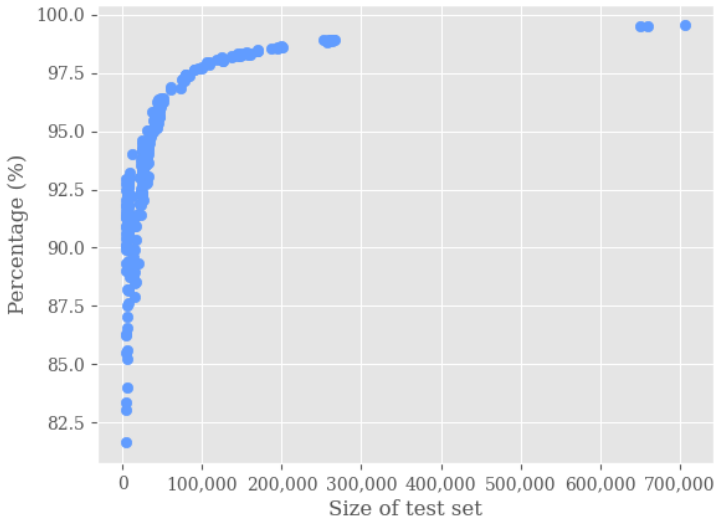

Fig. 3. The percentage of the local outlier factor algorithm's runtime consumed by the k-nearest neighbours subroutine.

Fortunately, in the case of the k-nearest neighbours algorithm, performant GPU implementations with source code *do* exist. We made use of the RAPIDS cuML library [23], which was found to have a highly efficient k-nearest neighbours implementation while still retaining a similar API to scikit-learn [21]. As a result, our solution simply involved integrating this k-nearest neighbours implementation into an otherwise fairly standard version of the sub-sequence local outlier factor algorithm. The pseudocode for our approach is shown in Algorithm 1.

Despite its simplicity, this approach had the desired effect. As may be seen in Figure 4, the GPU version of the local outlier factor algorithm was up to eleven times faster than the CPU version. Note that these results are obviously extremely dependent on the hardware—we should reiterate our usage of an Intel Core i9-12900k and NVidia RTX 3080 Ti for this comparison. Both the CPU

Algorithm 1 A GPU-optimised sub-sequence local outlier factor algorithm

Input:

`T`: A time series (1D array) consisting of n real numbers.
`k`: The number of neighbours for the KNN algorithm.
`w`: The sub-sequence window size.

Output:

`pointScores`: The anomaly scores for each point t_i determined using the local outlier factor.

Assume:

- kNN(`array`, `k`): The GPU-enabled k-nearest neighbours implementation. Returns the `indices` and `distances` of the k-nearest neighbours.
- applyWindowing(`array`, `w`): Converts a 1D time series to a 2D matrix containing $n - w - 1$ sub-sequences of length w.
- reverseWindowing(`subsequences`, `w`): Converts the array of size $n - w - 1$ containing anomaly scores for the sub-sequences into an array of size n containing the anomaly scores for the points in T.
- getReachDist(`indices`, `distances`, `k`): Calculates the reachability distance for all sub-sequences as defined in (1) in §2.2.
- getLRD(`indices`, `distances`, `k`): Calculates the local reachability densities for all sub-sequences as defined in (2) §2.2.
- getLOF(`localReachabilityDensities`, `indices`): Calculates the local outlier factor for all sub-sequences as defined in (3) §2.2.

```
subsequences ← applyWindowing(T, w)
indices, distances ← kNN(subsequences, k)
reachabilityDistances ← getReachDist(indices, distances, k)
localReachabilityDensities ← getLRD(reachabilityDistances)
subsequenceScores ← getLOF(localReachabilityDensities, indices)
pointScores ← reverseWindowing(subsequenceScores, w)
return pointScores
```

and GPU are widely-available consumer-grade hardware, and at a comparable price.

6 Enhancing the accuracy of the algorithm

The dramatic speed increase demonstrated in Figure 4 unlocks techniques for improving real-time anomaly detection accuracy which might not have been feasible otherwise. This includes ensembling, which is a technique that combines multiple weaker models into a single, stronger model.

6.1 Our ensembling approach

Based on the diverse predictive behaviour exhibited by the various hyperparameter configurations considered in Section 4, an ensemble model suggests itself. As a result, we modified Algorithm 1 for this purpose, as shown in Algorithm 2.

Algorithm 2 Ensembling local outlier factor models

Input:

`T`: A time series (1D array) consisting of n real numbers.
`neighbours`: The set of neighbour parameter values in the ensemble.
`windows`: The set of window sizes in the ensemble.

Output:

`combinedScores`: The anomaly scores for each point t_i determined using an ensemble of local outlier factor models.

Assume:

- All the function definitions in Algorithm 1 apply here too.
- getSubsets(`indices`, `distances`, `k`): Returns the subset of indices and neighbours for only the nearest k neighbours.
- addNewScores(`combinedScores`, `pointScores`): Adds the new scores to `combinedScores` array in such a way that scores are "spread" to neighbouring points (in this case, within 100 timesteps) to make the ensemble less sensitive to each ensemble member's noise.

```
combinedScores ← {}
for w in windows do
    subsequences ← applyWindowing(T, w)
    indices, distances ← kNN(subsequences, k)
    for k in neighbours do
        kIndices, kDistances ← getSubsets(indices, distances, k)
        reachabilityDistances ← getReachDist(kIndices, kDistances, k)
        localReachabilityDensities ← getLRD(reachabilityDistances)
        subsequenceScores ← getLOF(localReachabilityDensities, kIndices)
        pointScores ← reverseWindowing(subsequenceScores, w)
        combinedScores ← addNewScores(combinedScores, pointScores)
return combinedScores
```

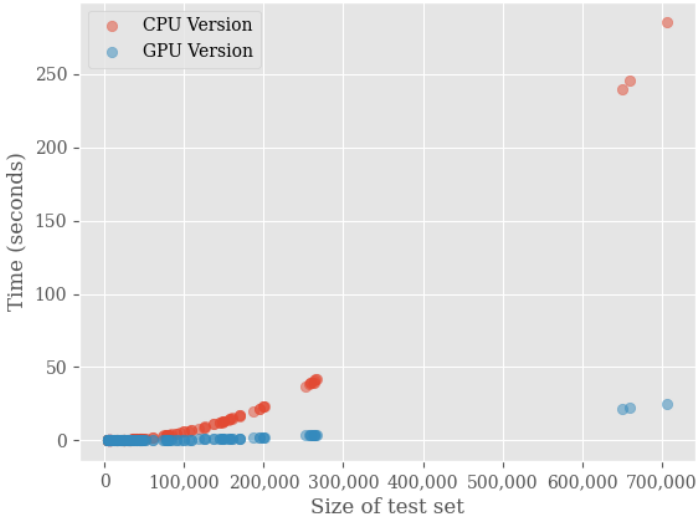

Fig. 4. A comparison of the computational times of the CPU and GPU versions of the local outlier factor algorithm.

Instead of ingesting just one value each for w and k, the ensembling algorithm makes use of two lists of parameter values, namely `neighbours` and `windows`. The ensemble comprises all combinations of k and w.

We employed a number of small tricks to optimise the ensemble runtime. Most importantly, we only call the k-nearest neighbours subroutine once per value of w (for the largest value of k) instead of for every value of k. This saves a significant amount of time, since the output of the k-nearest neighbours algorithm for smaller values of k is a subset of the output for the largest value of k.

We attempted to combine the predictions of the ensemble members in a few different ways. Based on the results of our experimentation, we opted for a hard voting approach where the anomaly scores were first "spread" to neighbouring points (within 100 timesteps). "Spreading" the scores was necessary since the naive approach of simply adding up the anomaly scores of each point and then returning the timestep with the highest combined score performed relatively poorly (this was highly susceptible to noise). For example, if many ensemble members predict slightly different timesteps within the correct region, they might be outvoted by a small handful of members who predicted exactly the same timestep in the wrong region.

Although our suggested approach works well, it is possible that other ensembling methods might work better. For instance, a soft voting approach could

work well, where each ensemble member's vote is weighted by its performance in respect of a validation set (if available) or based on some heuristic.

6.2 Other ideas to improve accuracy

While our ensembling approach was a success, we also experimented with a few other ideas aimed at improving accuracy even further. These ideas were not particularly successful, but we offer a brief description of some of the things we tried since better implementations than ours might lead to more favourable outcomes.

As illustrated in Section 4, the performance of the sub-sequence local outlier factor algorithm is highly sensitive to two parameters—the window size and the number of neighbours. Based on Figure 2, selecting the ideal hyperparameter configuration for each data set would yield an accuracy improvement of 22%. A manual approach may involve examining the time series and setting the window size based on its periodicity, although this is not necessarily feasible, realistic, or accurate for many large data sets. As a result, we experimented with the method employed by Lu *et al.* [15], who set the window size to the peak of the autocorrelation in the range 10 to 1000.

Deciding on the number of nearest neighbours to consider is also far from trivial. Breunig *et al.* [6] provided some guidelines, however, which include a minimum of 10 in order to remove undesirable statistical fluctuations. This is borne out by our results in Figure 1, where configurations with $k = 5$ performed particularly poorly. Moreover, they suggested setting a range of possible k-values based on cluster sizes, although this introduces further hurdles. Not only would clustering the sub-sequences increase the computational time significantly, but these clustering algorithms have their own hyperparameter values which would have to be tuned. In short, our attempts to tune the hyperparameters without a validation set were not particularly successful, and we did not manage to outperform the best configuration in Figure 1 (namely $w = 50,\ k = 50$).

We also attempted to pre-process the data in a better way using the sharpening method, as in DAMP [14,15]. Unfortunately, this also did not seem to improve our results, and actually made them worse. While we do suspect that better pre-processing or hyperparameter tuning should be able to improve upon the results in Figure 1, our experiments did not bear fruit.

7 Results and discussion

In Table 1, we summarise our results, and contrast them with the best approaches in the literature. The comparison is carried out in terms of runtime as well as accuracy, in order to give some indication of how feasible each approach might be when applied to large data sets or in an online setting. We implemented configurations typeset in **boldface** ourselves, while the rest of the results were borrowed from the papers referenced. The computation times are not directly comparable due to the different hardware setups, and are provided merely for

informative purposes. For instance, our models were executed on an NVidia RTX 3080 Ti GPU, while DAMP and NormA were run on an Intel Core i7-9700 CPU.

Model name	**Accuracy**	**Total time**
LOF ensemble	66.8%	2h 5m
DAMP (sharpened data) [15]	63.2%	4h 16m
LOF with $w = 50$, $k = 50$	62.4%	1m 43s
DAMP (out-of-the-box) [15]	51.2%	4h 16m
NormA [5]	47.4%	17m 48s
MERLIN++ [19]	42.4%	14m 30s
SCRIMP [29]	41.6%	24m 30s
LOF with $w = 1$, $k = 50$	14.4%	33s

Table 1. The accuracy and computational time results for a selection of algorithms tested on the univariate time series benchmark dataset of Keogh [10].

The first point worth noting is that our proposed ensembling approach comfortably outperforms all other models, including DAMP with sharpened data, meaning that it is, to the best of our knowledge, the new state-of-the-art algorithm for the UCR Time Series Anomaly Detection benchmark data set [10]. It achieves this feat without being prohibitively time-consuming. These promising results are particularly impressive considering how little fine-tuning was required. Not only does the local outlier factor algorithm have no need for a training set, but its only two hyperparameters (the lists of values for k and w) are relatively simple to set based on a given computational budget.

A second major takeaway is that the ensemble model is 4.4% more accurate than the best local outlier factor configuration (of those we tested), which indicates that hyperparameter tuning in isolation is very unlikely to outperform ensembling. We also included the results returned by the local outlier factor algorithm without windowing (*i.e.* a window size of 1) in our comparison to illustrate two closely-related points. First, we wanted to demonstrate the necessity of applying sliding windows on time series data—without this, performance is dreadful. Secondly, we aimed to show that the problem instances in the benchmark data [10] are far from trivial, and that most are unsolvable without accounting for the temporal relationship between timesteps.

Something worth pointing out is the distinct lack of deep learning approaches in Table 1. Numerous authors [4,15,25] have noted the poor performance of deep learning approaches on the benchmark data set, and have postulated that deep learning is overkill and unsuitable for univariate time series anomaly detection in general. It remains to be seen, however, whether the remarkable performance of the local outlier factor ensemble translates to similar performance in complex, multivariate time series.

8 Conclusion and future work

In this paper, we proferred a new approach towards solving the topical problem of anomaly detection in time series by putting a fresh twist on the well-established local outlier factor algorithm. This involved two relatively simple, but highly effective enhancements.

The first of these involved accelerating the local outlier factor algorithm significantly. We achieved this by parallelising the most expensive step of the local outlier factor algorithm (namely finding the k-nearest neighbours, which took up to 99% of the total computation time for large data sets) on a GPU. This resulted in a significant speed-up of up to 1100% for large data sets.

The second enhancement was aimed at improving the prediction accuracy of the local outlier factor algorithm by leveraging ensembles. After noticing the diverse predictive behaviour of different hyperparameter configurations, we developed an ensemble model that combined the predictions of these configurations by means of hard voting. This approach was simple but effective in establishing what is, to the best of our knowledge, a new state of the art of 66.8% on the benchmark data set. Moreover, thanks to the GPU acceleration, this was achieved within a very reasonable time.

In spite of the very promising results reported in this paper, we have no doubt that there is plenty of room for further algorithmic improvement. It is likely that even higher accuracy scores might be achieved by integrating some hyperparameter tuning and more advanced data pre-processing, or by developing a smarter way of selecting ensemble members. Although we limited ourselves to the local outlier factor algorithm in this paper, we anticipate that adding other anomaly detection algorithms (such as isolation forest or one-class support vector machines) to the ensemble would improve accuracy further. Another key avenue for future research is to extend and evaluate our approach in respect of streaming and multivariate time series data. We hope that our findings pave the way for further refinements in anomaly detection algorithms and continued progress in the quest for improved anomaly detection accuracy and efficiency.

References

1. Alghushairy, O., Alsini, R., Soule, T., Ma, X.: A review of local outlier factor algorithms for outlier detection in big data streams. Big Data and Cognitive Computing **5**(1) (2021). https://doi.org/10.3390/bdcc5010001
2. Alshawabkeh, M., Jang, B., Kaeli, D.: Accelerating the local outlier factor algorithm on a gpu for intrusion detection systems. In: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units. p. 104–110 (2010). https://doi.org/10.1145/1735688.1735707
3. Auskalnis, J., Paulauskas, N., Baskys, A.: Application of local outlier factor algorithm to detect anomalies in computer network. Elektronika ir Elektrotechnika **24**(3), 96–99 (2018). https://doi.org/10.5755/j01.eie.24.3.20972
4. Barrish, D., van Vuuren, J.: A taxonomy of univariate anomaly detection algorithms for predictive maintenance. South African Journal of Industrial Engineering **34**, 28–42 (2023). https://doi.org/10.7166/34-3-2943

5. Boniol, P., Linardi, M., Roncallo, F., Palpanas, T., Meftah, M., Remy, E.: Unsupervised and scalable subsequence anomaly detection in large data series. The VLDB Journal **30**(6), 909–931 (2021). https://doi.org/10.1007/s00778-021-00655-8
6. Breunig, M.M., Kriegel, H., Ng, R.T., Sander, J.: Lof: Identifying density-based local outliers. In: Proceedings of the 26th ACM SIGMOD International Conference on Management of Data. p. 93–104 (2000). https://doi.org/10.1145/335191.335388
7. Cheng, Z., Zou, C., Dong, J.: Outlier detection using isolation forest and local outlier factor. In: Proceedings of the 9th Conference on Research in Adaptive and Convergent Systems. p. 161–168 (2019). https://doi.org/10.1145/3338840.3355641
8. Huet, A., Navarro, J.M., Rossi, D.: Local evaluation of time series anomaly detection algorithms. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. p. 635–645. KDD '22, Association for Computing Machinery (2022). https://doi.org/10.1145/3534678.3539339
9. Hwang, W.S., Yun, J.H., Kim, J., Min, B.G.: Do you know existing accuracy metrics overrate time-series anomaly detections? In: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing. p. 403–412. SAC '22, Association for Computing Machinery (2022). https://doi.org/10.1145/3477314.3507024
10. Keogh, E.: Time-series anomaly detection datasets, sigkdd 2021, [Online], [Cited December 18th, 2023], Available from `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/UCR_TimeSeriesAnomalyDatasets2021.zip`
11. Kim, S., Choi, K., Choi, H.S., Lee, B., Yoon, S.: Towards a rigorous evaluation of time-series anomaly detection. In: Proceedings of the 36th AAAI Conference on Artificial Intelligence. pp. 7194–7201 (2022). https://doi.org/10.1609/aaai.v36i7.20680
12. Kriegel, H.P., Kröger, P., Schubert, E., Zimek, A.: Loop: Local outlier probabilities. In: Proceedings of the 18th ACM Conference on Information and Knowledge Management. p. 1649–1652 (2009). https://doi.org/10.1145/1645953.1646195
13. Li, A., Xu, W., Liu, Z., Shi, Y.: Improved incremental local outlier detection for data streams based on the landmark window model. Knowledge and Information Systems **63**(8), 2129–2155 (2021). https://doi.org/10.1007/s10115-021-01585-1
14. Lu, Y., Wu, R., Mueen, A., Zuluaga, M.A., Keogh, E.: Matrix profile xxiv: Scaling time series anomaly detection to trillions of datapoints and ultra-fast arriving data streams. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. p. 1173–1182 (2022). https://doi.org/10.1145/3534678.3539271
15. Lu, Y., Wu, R., Mueen, A., Zuluaga, M.A., Keogh, E.: Damp: Accurate time series anomaly detection on trillions of datapoints and ultra-fast arriving data streams. Data Mining and Knowledge Discovery **37**(2), 627–669 (2023). https://doi.org/10.1007/s10618-022-00911-7
16. Ma, M., Ngan, H.Y., Liu, W.: Density-based outlier detection by local outlier factor on large-scale traffic data. In: Proceedings of the 28th International Symposium on Electronic Imaging (2016). https://doi.org/10.2352/ISSN.2470-1173.2016.14.IPMVA-385
17. Mishra, S., Chawla, M.: A comparative study of local outlier factor algorithms for outliers detection in data streams. In: Proceedings of the 1st International Conference on Emerging Technologies in Data Mining & Information Security. pp. 347–356 (2019). https://doi.org/10.1007/978-981-13-1498-8_31
18. Nakamura, T., Imamura, M., Mercer, R., Keogh, E.: Merlin: Parameter-free discovery of arbitrary length anomalies in massive time series archives. In: Proceedings of the 20th IEEE International Conference on Data Mining (ICDM). pp. 1190–1195 (2020). https://doi.org/10.1109/ICDM50108.2020.00147

19. Nakamura, T., Mercer, R., Imamura, M., Keogh, E.: Merlin++: Parameter-free discovery of time series anomalies. Data Mining and Knowledge Discovery **37**(2), 670–709 (2023). https://doi.org/10.1007/s10618-022-00876-7
20. Oehmcke, S., Zielinski, O., Kramer, O.: Event detection in marine time series data. In: Proceedings of the 38th Annual German Conference on AI. pp. 279–286 (2015). https://doi.org/10.1007/978-3-319-24489-1_24
21. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011). https://doi.org/10.5555/1953048.2078195
22. Pokrajac, D., Lazarevic, A., Latecki, L.J.: Incremental local outlier detection for data streams. In: 2007 IEEE Symposium on Computational Intelligence and Data Mining. pp. 504–515 (2007). https://doi.org/10.1109/CIDM.2007.368917
23. Raschka, S., Patterson, J., Nolet, C.: Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence (2020), `https://arxiv.org/abs/2002.04803`
24. Salehi, M., Leckie, C., Bezdek, J.C., Vaithianathan, T., Zhang, X.: Fast memory efficient local outlier detection in data streams. IEEE Transactions on Knowledge and Data Engineering **28**(12), 3246–3260 (2016). https://doi.org/10.1109/ICDE.2017.32
25. Wu, R., Keogh, E.J.: Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. IEEE Transactions on Knowledge and Data Engineering **35**(3), 2421–2429 (2023). https://doi.org/10.1109/ICDE53745.2022.00116
26. Xu, Z., Kakde, D., Chaudhuri, A.: Automatic hyperparameter tuning method for local outlier factor, with applications to anomaly detection. In: 2019 IEEE International Conference on Big Data. pp. 4201–4207 (2019). https://doi.org/10.1109/BigData47090.2019.9006151
27. Yeh, C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Silva, D.F., Mueen, A., Keogh, E.: Matrix profile i: All pairs similarity joins for time series—a unifying view that includes motifs, discords and shapelets. In: Proceedings of the 16th IEEE International Conference on Data Mining. pp. 1317–1322 (2016). https://doi.org/10.1109/ICDM.2016.0179
28. Zhao, Y., Chen, G.H., Jia, Z.: Tod: Gpu-accelerated outlier detection via tensor operations. Proceedings of the VLDB Endowment **16**(3), 546–560 (2022). https://doi.org/10.14778/3570690.3570703
29. Zhu, Y., Yeh, C.C.M., Zimmerman, Z., Kamgar, K., Keogh, E.: Matrix profile xi: Scrimp++: Time series motif discovery at interactive speeds. In: 2018 IEEE International Conference on Data Mining (ICDM). pp. 837–846 (2018). https://doi.org/10.1109/ICDM.2018.00099
30. Zhu, Y., Zimmerman, Z., Senobari, N.S., Yeh, C.C.M., Funning, G., Mueen, A., Brisk, P., Keogh, E.: Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In: 2016 IEEE 16th International Conference on Data Mining (ICDM). pp. 739–748 (2016). https://doi.org/10.1109/ICDM.2016.0085