SoftwareX 22 (2023) 101363

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

tegdet: An extensible Python library for anomaly detection using time evolving graphs

Simona Bernardi [a,*], Raúl Javierre [b], José Merseguer [a]

[a] Dept. de Informática e Ing. de Sistemas, Universidad de Zaragoza, Spain
[b] Hiberus, Spain

ARTICLE INFO

Article history:
Received 28 October 2022
Received in revised form 23 January 2023
Accepted 8 March 2023

Dataset link: https://github.com/DiasporeUnizar/TEG

Keywords:
Unsupervised anomaly detection
Univariate time-series
Time evolving graphs
Dissimilarity metrics

ABSTRACT

This paper presents **tegdet**, a new **Python** library for anomaly detection in unsupervised approaches. The input of the library is a univariate time series, representing observations of a given phenomenon. Then, **tegdet** identifies *anomalous epochs*, i.e., time intervals where the observations differ in a given percentile of a baseline distribution. *Epochs* are represented by *time evolving graphs* and the baseline distribution is given by the dissimilarities between a reference graph and the graphs of the epochs. Currently, the library implements 28 dissimilarity metrics, i.e., 28 different anomaly detection techniques, and its extensible design allows to easily introduce new ones. **tegdet** exposes a complete functionality to carry out the anomaly detection, through a straightforward designed API. Summarizing, to the best of our knowledge, **tegdet** is the first publicly available library, based on time evolving graphs, for anomaly detection in time series. Our experimentation shows promising results. For example, Clark and Divergence techniques can achieve an accuracy of 100%, while the time to build the model and predict lasts for few hundreds milliseconds.

Code metadata

Current code version	v1.0.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-22-00349
Permanent link to Reproducible Capsule	
Legal Code License	GNU GPL-2.0
Code versioning system used	pypi, git
Software code languages, tools, and services used	python
Compilation requirements, operating environments & dependencies	python >= 3.6.1, pandas, scipy
If available Link to developer documentation/manual	https://github.com/DiasporeUnizar/TEG
Support email for questions	simonab@unizar.es

1. Motivation and significance

Time series analysis and forecasting is a noticeable branch of data science that focuses on developing models derived from a sequence of data points, observed at different time instants, to gain an understanding of a given phenomenon and to make predictions on its future [1]. Anomaly detection [2] in time series is a step forward and, nowadays, one of the main topics of interest for researchers and practitioners in uncountable application domains. For example, network intrusion detection, malware detection, fraud detection, data center monitoring, industrial damage detection, medical images, military surveillance or social media research.

Numerous anomaly detection methods, based on time series, have been proposed in the literature, and three categories have been identified [3]: statistical approaches, machine learning approaches and deep learning approaches. Detection methods can deal with univariate and/or multivariate time series. A univariate time series is an ordered set of real-valued observations, where each observation is recorded at a specific time. A multivariate time series is an ordered set of k-dimensional vectors, where each vector is recorded at a specific time and consists of k real-valued observations [4].

The **tegdet** library implements a statistical approach, in particular a class of dissimilarity-based anomaly detection methods

* Corresponding author.
E-mail address: simonab@unizar.es (Simona Bernardi).

https://doi.org/10.1016/j.softx.2023.101363

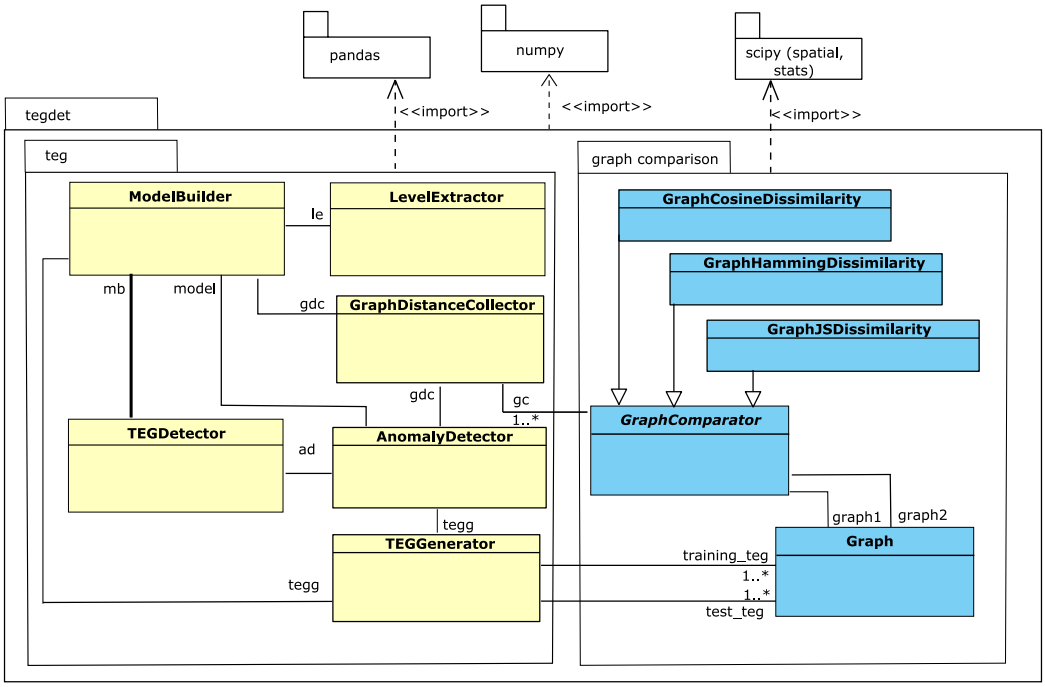


Fig. 1. Overview of the **tegdet** library architecture.

for univariate time series. These methods rely on *time evolving graphs* [5] (TEGs), which offer a graph-based representation of fixed-length subsequences of the original time series. According to Akoglu et al. [5] many reasons make graph-based approaches vital and necessary, among them a powerful representation of data inter-dependencies and its robust machinery. Each detection method in **tegdet** leverages a dissimilarity metric, which defines how to compare graph-based subsequences with a reference graph, and how to build a baseline distribution model. Then, *anomalous epochs* are identified as data subsequences that differ, from the reference graph, in a given percentile of the baseline distribution.

Statistical libraries for the analysis and forecasting of time series are available in Python [6] and R [7], e.g., `statmodels`[1] in Python or `stats`[2] in R. However, to the best of our knowledge, none of these libraries provide support to the anomaly detection of time series based on time evolving graphs. Moreover, most dissimilarity-based implementations from the literature [4] are based on only one fixed metric, typically the Euclidean distance. The **tegdet** library overcomes these limitations by currently supporting 28 dissimilarity metrics using TEGs, and it has been designed to be easily extended with new ones. This is important since, as pointed out in [3], each detection method is specific to the type data, hence the dissimilarity metric greatly influences the accuracy of the detection results. For example, techniques based on Clark and Divergence metrics [8] are well suited to detect energy frauds [9]. In the end, offering **tegdet** many different statistical techniques greatly improves the chances of getting better results in different fields.

2. Software description

tegdet is a novel library for anomaly detection, based on time evolving graphs (TEGs) and implemented in **Python** language [6] (compatible version >= 3.6.1). The input of the library must be a univariate time series representing observations of a given phenomenon. The output identifies *anomalous epochs*, as previously defined.

2.1. Software architecture

Fig. 1 depicts a modules view of the library's software architecture. It is made of two packages and it uses three other different **Python** libraries (*pandas*, *numpy* and *scipy*):

- *teg*: It is the main package. It defines the API for the users of the library.
- *graph comparison*: It is responsible for creating graphs and computing dissimilarities between these graphs, according to a given metric.

Each package of the library is made up of a set of **Python** classes,[3] which are also depicted in Fig. 1.

As previously introduced, **tegdet** implements 28 dissimilarity metrics, hence, 28 different anomaly detection techniques. Moreover, it has been designed to be easily extended to introduce new anomaly detection techniques. The **tegdet** design ensures the extensibility by decoupling the computation of the dissimilarity metric and the rest of the training and testing process. In particular, the abstract class ***GraphComparator***, Fig. 1, will be specialized in as many classes as techniques want to be implemented. Fig. 1 depicts three specialized classes as examples, e.g., ***GraphJSDissimilarity***, which implements the Jensen–Shannon metric [8]. Each specialized class only needs to implement the abstract method ***compare_graphs***, having the purpose of implementing the technique defined by the dissimilarity metric.

The decoupling introduced by **tegdet** not only ensures the extensibility, it also favors the good performance results of the library, as proved in [9]. Being the temporal complexity to create the TEGs linear, with respect to the length of the input, in

1 https://www.statsmodels.org/

2 https://stat.ethz.ch/R-manual/R-devel/library/stats/html/00Index.html

3 A detailed description of the attributes and methods of these classes can be found in: https://github.com/DiasporeUnizar/TEG.

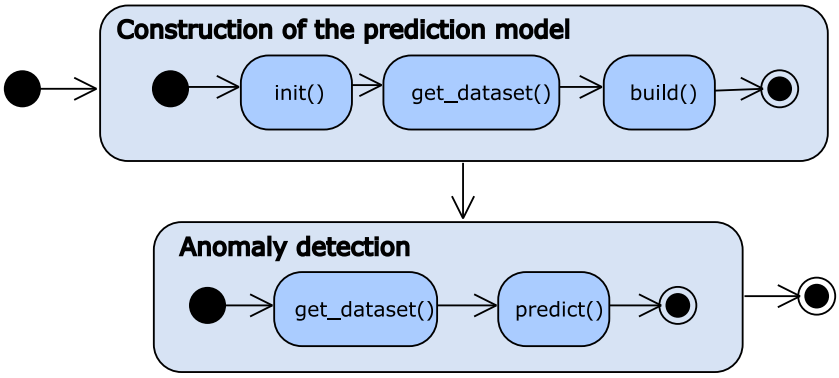


Fig. 2. Workflow of the usage of the library.

the worst case, then, the performance impact of adding a new detection technique is attributable to the implementation of the technique exclusively.

2.2. Software functionalities

The methods of the ***TEGDetector*** class, conform the API of the **tegdet** library. These methods are for the users of the library to: (a) discretize a dataset, (b) create TEGs, (c) construct a prediction model, and (d) detect outliers. The following list describes each of these methods in detail.

init(): This method is the constructor of the ***TEGDetector*** class. Hence, it is the entry point for the user of the library to initialize the parameters for constructing the model and carrying out the detection. These parameters conform the attributes of the class, they are described at the end of this section.

get_dataset() This method is used to load a dataset, which is indicated as a parameter. The dataset file must be in CSV (comma-separated values) format. The method returns a *pandas Dataframe* ready to be used for building a model.

build_model() This method is used to build a prediction model. The method receives as a parameter a training dataset, created by the previous method. The model is returned as a `ModelBuilder` object, also the time to build the model is returned.

predict() This method is used to make a prediction. It receives a testing dataset and a model. Then, the method returns: the outliers, as a *numpy array* of {0,1} values, the total number of observations, and the time spent to make the prediction.

compute_confusion_matrix() This method is used to compute a confusion matrix. It receives two *numpy* arrays, one with ground true values and the other with the predicted values obtained by the previous method. The method returns the confusion matrix.

print_metrics() This method is used to print on the standard output. In particular, it prints: the name of the metric, the setting of the input parameters, the name of the testing dataset, the performance metrics (i.e., time to build the model and time to make predictions) and finally the confusion matrix.

metrics_to_csv() This method is used to save information, in CVS format, in the file indicated as a parameter. In particular, it saves: the current configuration, the name of the testing dataset, the performance metrics and finally the confusion matrix.

The attributes of the ***TEGDetector*** class, which are the parameters of the constructor method, represent the information that the user needs to provide for carrying out the anomaly detection. The following list describes each of these attributes in detail.

***metric*:** The name of the dissimilarity metric used to compare graphs. The complete list of metrics currently implemented in the library appears in Listing 1, lines 11–16.

***n_bins*:** Being the time series observations mapped into an ordered set of levels, then *n_bins* represents the cardinality of this set, i.e., the number of levels (its default value is 30). The size of the generated graphs (i.e., number of nodes and edges) is proportional to this value.

***n_obs_per_period*:** Being the time series partitioned into a set of consecutive periods, i.e., epochs, then this attribute represents the number of observations in each period, see Fig. 3. The length of the graph sequence, that corresponds to the number of period partitions, is inversely proportional to this value. Its default value is 336.

***alpha*:** Represents a threshold for the anomaly detection (its default value is 5). Hence, an epoch will be labeled as *anomalous* when its observations differ from the reference model in a quantity above the *100-alpha* percentile of the baseline distribution.

2.3. Usage of the library

The methods of the API previously described are for the users of the library to construct a prediction model and carry out the anomaly detection. Fig. 2 summarizes such steps, which are detailed in the following.

Construction of the prediction model. First, the *init* method is called to set the parameters needed to create the training TEGs and the prediction model. Next, the *get_dataset* method loads the training dataset. Then, the *build_model* method generates the prediction model, as follows: (a) it produces a discretized dataset; (b) it generates a sequence of training graphs; (c) it obtains the global graph; and (d) computes the dissimilarities between each graph of the sequence and the global graph.

Anomaly detection. Initially, the *get_dataset* method is called to load the testing dataset. Then, the *predict* method is called, which triggers: (a) the computation of the dissimilarities between each testing graph of the sequence and the global graph; and (b) the computation of outliers.

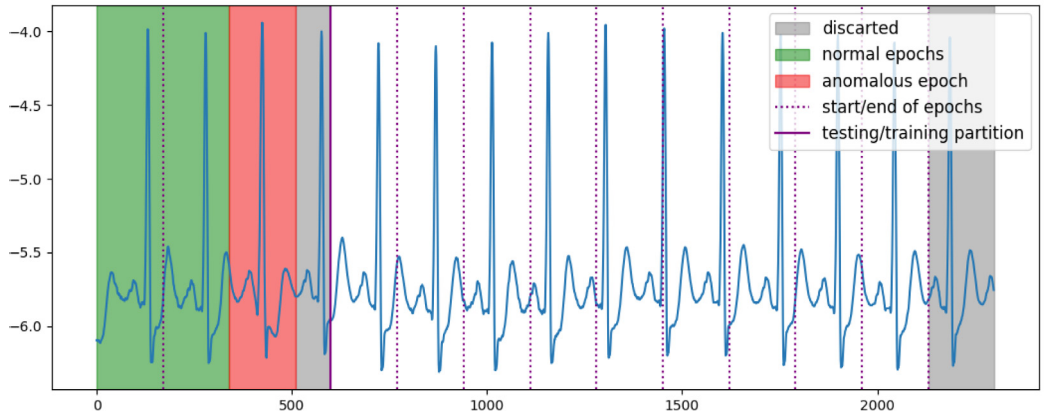


Fig. 3. Time series partitioning (*n_obs_per_period* = 170). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Excerpt from the results file, saved in CVS format.

```
detector,n_bins,n_obs_per_period,alpha,testing_set,time2build,time2predict,tp,tn,fp,fn
Hamming,30,170,5,testing,0.029548168182373047,0.012964010238647461,0,2,0,1
Cosine,30,170,5,testing,0.03642392158508301,0.01453399658203125,1,2,0,0
...
Clark,30,170,5,testing,0.030735254287719727,0.013725996017456055,1,1,1,0
Additivesymmetric,30,170,5,testing,0.028728961944580078,0.012762784957885742,1,2,0,0
```

3. Illustrative examples

The following examples demonstrate how to use **tegdet** to: build prediction models using training datasets, make predictions using testing datasets and save results in CSV format. The script in Listing 1, that uses the methods of the API described in previous section, clearly identifies each step for using **tegdet**. The datasets and scripts used in these examples are available at the **tegdet** repository.[4]

3.1. Heartbeat anomaly detection

This example considers the QTDB 0606 ECG dataset [10], in textual format, which has been downloaded from the GrammarViz repository [11]. The time series, shown in Fig. 3 (blue curve), consists of 2299 observations, and the third heartbeat, emphasized by a red region, is known to be anomalous.

Concerning the input parameters: the number of observations per period (*n_obs_per_period*) has been set to 170, this value corresponds to the window size used in the experiment by GrammarViz [11], the number of levels and the significance level (*n_bins* and *alpha*) are set to their default values. In Fig. 3, the vertical dotted lines mark the start/end of each epoch according to this setting. The green and red regions in the plot, emphasize the normal and anomalous epochs, respectively, of the testing set.

Table 1 shows an excerpt of the results produced by running the script in Listing 1, which corresponds to this example. The first line contains the headers of the columns for the other lines. Then, each of the following lines indicate: the name of the metric, the setting of the parameters, the performance results, i.e., time to build the model and time to make predictions, and, finally, the entries of the confusion matrix, i.e, true positive, true negative, false positive and false negative, respectively.

Table 2
Formulae using the confusion matrix entries.

true positive rate (*tpr*)	$tp/(tp+fn)$
true negative rate (*tnr*)	$tn/(fp+tn)$
balanced accuracy	$(tpr+tnr)/2$

These results have been post-processed to analyze the quality of the metrics, in terms of performance and detection capability. Fig. 4 shows the balanced accuracy of each metric, that is a function of the confusion matrix entries, as indicated by the formula in Table 2. Observe that most of the metrics make a correct prediction, i.e., two normal heartbeats and one anomalous (balanced accuracy of 100%), three metrics correctly identify the anomalous heartbeat but are not able to recognize the normal ones (balanced accuracy of 75%), whereas the remaining metrics have poor quality (balanced accuracy of 50%).

3.2. Energy fraud detection

Although resilient and reliable to supply electricity, smart grids are vulnerable to attacks and frauds [12]. In [9], **tegdet** is used to identify energy frauds caused by *swap* attacks, i.e., attacks aim at defrauding the energy utility by paying less than consumed. Moreover, [9] presents a complete assessment of the quality of the **tegdet** library. Experiments in [9] assume a time-of-use contract, and the cost of the energy depends on peak and off-peak periods [13].

The dataset used comes from the Ireland's Commission for Energy Regulation [14], and it collects the energy consumption (in kWh) of a particular smart meter, every half an hour during 75 weeks. The time series is assumed not to be affected by attacks and it has been partitioned in training and testing sets,

[4] https://github.com/DiasporeUnizar/TEG

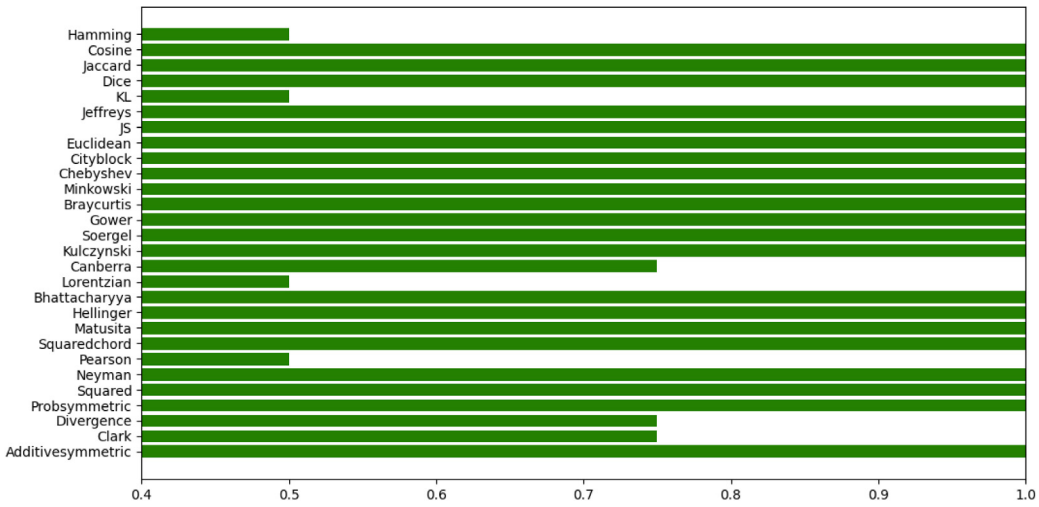


Fig. 4. Metrics balanced accuracy ($n\_bins = 30$, $n\_obs\_per\_period = 170$, $alpha = 5$).

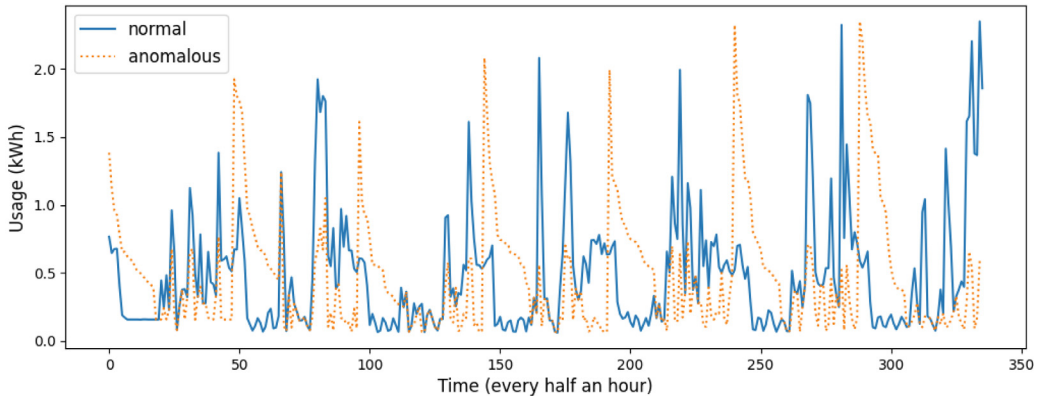
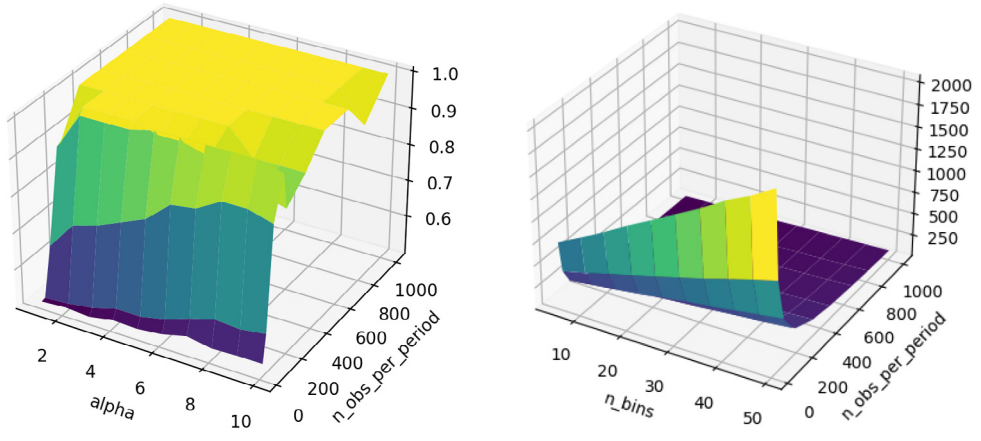


Fig. 5. Time series of the two testing sets, one week observations.



(a) n_bins=30 (b) $alpha$=5

Fig. 6. Results of Clark method: (left) accuracy (right) mean time to build the model (ms.).

respectively of 60 and 15 weeks length. Additionally, a synthetic time-series, representing a *swap* attack, is used as a second training set: it is generated from the first training set, by swapping the observations between peak and peak-off periods (from 9:00am to midnight and from midnight to 9:00am) [15]. Fig. 5 shows the first week observations from the two testing sets.

Listing 1: Script `ecg_tegdet_variants.py`

```
import os
import pandas as pd
import numpy as np
from tegdet.teg import TEGDetector

#Input dataset/output results paths
TS_PATH = "/dataset/ecg/ecg0606.csv"
RESULTS_PATH = "/script_results/ecg/tegdet_variants_results.csv"

#List of metrics (detector variants)
list_of_metrics = ("Hamming","Cosine","Jaccard","Dice","KL","Jeffreys","JS",
        "Euclidean", "Cityblock", "Chebyshev", "Minkowski", "Braycurtis",
        "Gower", "Soergel", "Kulczynski", "Canberra", "Lorentzian",
        "Bhattacharyya", "Hellinger", "Matusita", "Squaredchord",
        "Pearson", "Neyman", "Squared", "Probsymmetric", "Divergence",
        "Clark", "Additivesymmetric" )

def build_and_predict(metric,n_bins,n_obs_per_period,alpha):

    #Create a new tegdet
    tegd = TEGDetector(metric,n_bins,n_obs_per_period,alpha)

    #Load time series
    cwd = os.getcwd()
    ts_path = cwd + TS_PATH
    ts = tegd.get_dataset(ts_path)

    #Partition the time series in training and testing sets
    test = ts[:600]
    train = ts[600:]

    #Build model with the training set
    model, time2build = tegd.build_model(train)

    #Make prediction with the testing set
    outliers, n_periods, time2predict = tegd.predict(test, model)

    #Set ground true values
    ground_true = np.zeros(n_periods)
    ground_true[n_periods-1] = 1

    #Compute confusion matrix
    cm = tegd.compute_confusion_matrix(ground_true, outliers)

    #Collect detector configuration
    detector = {'metric': metric, 'n_bins': n_bins,
                'n_obs_per_period':n_obs_per_period, 'alpha': alpha}

    #Collect performance metrics in a dictionary
    perf = {'tmc': time2build, 'tmp': time2predict}

    #Store basic metrics
    results_path = cwd + RESULTS_PATH
    tegd.metrics_to_csv(detector, "testing", perf, cm, results_path)

if __name__ == '__main__':

    for metric in list_of_metrics:
        build_and_predict(metric,30,170,5)
```

The experiments in [9] produced promising results regarding the accuracy of the Clark and Divergence dissimilarity metrics.[5] For example, Fig. 6(a) shows that the Clark based detection method is slightly sensitive to the significance level (*alpha*) and it predicts correctly all the testing epochs, both normal and anomalous ones, when the length of the epochs are set at least half-week (168 observations). Concerning the execution time of the implemented algorithms, it is sensitive to the values set to the *n_obs_per_period* and *n_bins* parameters, which define the characteristics of the graph sequence. Fig. 6(b) shows the mean time to build the model: the lower is the value of *n_obs_per_period* the higher is the execution time, since the longer will be the sequence of graphs to be generated. Besides, the higher is the value of *n_bins* the higher is the execution time, since the size of the graphs to be generated (number of nodes and edges), as well as the computation of the graph dissimilarity, are in direct proportion to the number of levels.

4. Impact

Anomaly detection is a broad field of research that can be applied in uncountable application domains. The methods supporting the anomaly detection span multiples fields, from classical statistical methods to more actual machine learning techniques. The work in [3] compared the performance of 20 univariate anomaly detection methods by using 368 univariate time series

[5] Accuracy metric was considered in the experiments, since they were based on balanced normal and anomalous testing sets.

datasets. The authors concluded that the statistical approaches perform best on univariate time-series by detecting *anomalous epochs*, also detecting point anomalies. The work affirms that statistical techniques require less computation time compared to machine learning and deep learning approaches. Although these facts highlight the significance of the statistical libraries, such as **tegdet**, we consider necessary to have software packages for each anomaly detection field. Moreover, endowing these packages with capabilities for being easily extended, as **tegdet** does, the chances for addressing a large number of domains grow, hence the potential impact of the **tegdet** library.

In this context, **tegdet** is the first publicly available library for anomaly detection in time series, based on time evolving graphs. These graphs offer a better representation of inter-dependencies, as well as more robust mathematical machinery, than other formalisms [5]. Aside the experiments presented in this paper and in [9], **tegdet** was used in a project related to smart grid security [16] and it is currently being used in the project here referred in the Acknowledgments section. Since the release of version 1.0.0 in June 2022 in the PyPI public repository, **tegdet** has had around 960 downloads, according to Google BigQuery [17].

5. Conclusions

The **tegdet** library offers a simple API that intuitively enables the user to load datasets, build a prediction model and detect outliers. The user can fine-tune the detection process by customizing a few input parameters. Concretely, the metric or detection method, the number of observations per period, the number of levels and the significance level. The work in [9] explains how to easily improve the anomaly detection process by setting these parameters. For example, regarding the accuracy, it was achieved a 100%: first, by analyzing the periodicity of the time series thus setting the length of the periods, and later carrying out a sensitivity analysis over the complete set of the metrics in the library.

The software design of the library decouples, for the sake of the extensibility, the inherent processes of the anomaly detection. In fact, the simplicity of the design makes that users with very basic skills in **Python**, and completely oblivious to the time evolving graphs formalism, can introduce new methods of detection.

Another important aspect of the library is the good performance results achieved. Indeed, the experiments in [9] show that when the number of graphs is in the order of hundreds and their size ranges from 10 to 50 nodes then the time to build the model is around few hundreds milliseconds. For longer graph sequences, in the order of thousands, the time to build the model is few seconds, where the most time consuming step concerns the computation of graph dissimilarities (i.e., 70% of the overall time). As future work, we aim to investigate whether advanced representations for TEGs could improve the time to build the model. For example, graph implementations based on sparse matrices could improve the memory and time efficiency of the graphs comparison.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data/code is available at: https://github.com/DiasporeUnizar/TEG.

Acknowledgments

S. Bernardi and J. Merseguer were supported by the Spanish Ministry of Science and Innovation [ref. PID2020-113969RB-I00]

Appendix A. Library repositories

The **tegdet** library is available at the official *PyPi* repository[6] as well as at the *GitHub* repository.[7] In particular, the latter also includes the following resources:

- API documentation.
- Documentation about the installation and implementation.
- Datasets used in this paper.
- **Python** scripts. Listing 1 shows the script used in Section 3.1.

References

[1] Shumway Robert H, Stoffer David S. Time series analysis and its applications. Springer texts in statistics, fourth ed.. Springer Science+Business Media; 2017.
[2] Grubbs Frank E. Procedures for detecting outlying observations in samples. Technometrics 1969;11(1):1–21.
[3] Braei Mohammad, Wagner Sebastian. Anomaly detection in univariate time-series: A survey on the state-of-the-art. 2020, CoRR, abs/2004.00433.
[4] Blázquez-García Ane, Conde Angel, Mori Usue, Lozano Jose A. A review on outlier/anomaly detection in time series data. ACM Comput Surv 2021;54(3).
[5] Akoglu Leman, Tong Hanghang, Koutra Danai. Graph based anomaly detection and description: A survey. Data Min Knowl Discov 2015;29(3):626–88.
[6] van Rossum Guido. Python programming language. In: Chase Jeff, Seshan Srinivasan, editors. Proceedings of the 2007 USENIX annual technical conference, Santa Clara, CA, USA, June (2007) 17-22. USENIX; 2007.
[7] R Core Team. R: A language and environment for statistical computing. Vienna, Austria: R Foundation for Statistical Computing; 2018.
[8] Cha Sung-Hyuk. Comprehensive survey on distance/similarity measures between probability density functions. Int J Math Models Methods Appl Sci 2007;1(4):300–7.
[9] Bernardi Simona, Merseguer José, Javierre Raúl. Tegdet: An extensible python library for anomaly detection using time-evolving graphs. 2022, https://arxiv.org/abs/2210.08847, [Accessed: 2022-10-20].
[10] Laguna P, Mark RG, Goldberg A, Moody GB. A database for evaluation of algorithms for measurement of qt and other waveform intervals in the ecg. In: Computers in cardiology. 1997, p. 673–6.
[11] Pavel Senin. GrammarViz GitHub repository. 2022, https://github.com/GrammarViz2/grammarviz2_src, [Accessed: 2022-10-13].
[12] He Haibo, Yan Jun. Cyber-physical attacks and defences in the smart grid: a survey. IET Cyber-Phys Syst: Theory Appl 2016;1(1):13–27.
[13] Krishna VB, Lee K, Weaver GA, Iyer RK, Sanders WH. F-DETA: A framework for detecting electricity theft attacks in smart grids. In: 2016 46th Annual IEEE/IFIP international conference on dependable systems and networks (DSN), 2016, p. 407–18.
[14] Irish social science data archive, commission for energy regulation. CER Smart Metering Project; 2012, URL https://www.ucd.ie/issda/data/commissionforenergyregulationcer/.
[15] Bernardi Simona, Javierre Raúl, Merseguer José, Requeno José Ignacio. Detectors of smart grid integrity attacks: an experimental assessment. In: 17th European dependable computing conference, EDCC 2021, Munich Germany, September (2021) 13-16. IEEE Computer Society; 2021.
[16] Bernardi Simona, Merseguer José. Model & data-driven resilience engineering for complex dynamic systems (medrese). Ministerio de economía, industria y competitividad; 2019-2021, RTI2018-098543-B-I00.
[17] Naidu Siddartha, Tigani Jordan. Google bigquery analytics. John Wiley & Sons; 2014.

6 PyPi url: https://pypi.org/project/tegdet/.
7 GitHub url: https://github.com/DiasporeUnizar/TEG.