Executive Summary of the Thesis

Anomalearn: a modular and extensible library for the development of time series anomaly detection models

Laurea Magistrale in Computer Science and Engineering - Ingegneria Informatica

Author: Marco Petri

Advisor: Prof. Piero Fraternali

Co-advisor: Nicolò Oreste Pinciroli Vago

Academic year: 2021-2022

1. Introduction

Time series analysis is a relevant topic dating back to the early 20th century. Initially, it involved weather, medical, economic and astronomic data for several purposes. In this thesis, time series data are handled to build a library for aiding the creation of new models for anomaly detection on time series. That is the creation of models able to identify portions of data dissimilar from normal data for detection purposes. These abnormal data may represent dangerous or faulty behaviour in industrial equipment or signs of illnesses in medical data. This thesis does not focus on a specific application field (like ECG data); it looks at general approaches to anomaly detection on time series. To this end, it is possible to observe that there are several publicly available datasets for this task. Between these datasets, some contain simply discoverable anomalies, and some include complex anomalies requiring the adoption of complex and powerful techniques for detection. Inspired by the work of [7], this thesis addresses the problem of analysing the complexity in time series anomaly detection datasets by employing automatic techniques for the computation of a score representing the degree of simplicity of datasets. Then, given the lack of a Python library with the aim of enabling the data scientist to develop time series anomaly detection methods, this thesis proposes anomalearn. Anomalearn is a Python library aimed at minimizing the effort in the implementation of new methods of anomaly detection on time series. The library is compared to scikit-learn [4], Orion [2], and ML-Blocks [5] for evaluation of the other libraries devoted to the development of new machine learning models.

2. Evaluation of datasets

Datasets are essential for evaluating the efficacy of machine and deep learning methods, and it is possible to use both public and private datasets. However, the usage of publicly available datasets is always encouraged since it makes the experiment reproducible by other data scientists. Up to now, datasets have been used to assess the quality of methods, but there is no commonly established automatic procedure to evaluate the quality and attractiveness of such datasets. Consequently, some ordinarily used and publicly available datasets for anomaly detection contain several trivial anomalies [7] that are not assisting the scientific community in pur-

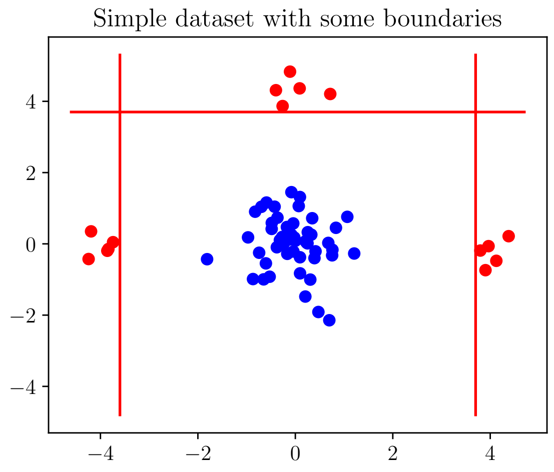

Figure 1: Simple dataset that is not linearly separable. Blue points are normalities, red are anomalies.

suing scientific progress. Thereupon, there is a trend in showing that deep learning and complex methods are not the silver bullets to solve these tasks; the notable work in [7] assesses the triviality of publicly available datasets by making considerations regarding whether simple approaches can solve them (that is, achieving maximum accuracy). In their work, they depict a dataset as simple if there is a model capable of solving it whose implementation can be encoded in one line, excluding libraries implementing machine or deep learning solutions. Therefore, they propose the following one-line approach:

$$x > c_1 \text{movmean}(x, w) + c_2 \text{movstd}(x, w) + b \tag{1}$$

Where x is the time series, and $w \geq 1$ is the length of the sliding window. If for any value of the constants c_1, c_2, b, this method solves the dataset; the dataset is labelled as simple. Even though this description suffices to state triviality, using it in parallel with the following trivial approach gives a thorough description:

$$x < c_1 \text{movmean}(x, w) - c_2 \text{movstd}(x, w) - b \tag{2}$$

This approach for evaluating simplicity inspired the creation and definition of simplicity for time series anomaly detection datasets. The idea consists in providing a method-agnostic definition of simple computation. To accomplish the task, although linear separability seems to fit the requirements, it might not flag as simple datasets as that in figure 1, in which anomalies and normalities are effortlessly separable. Therefore, this thesis defines a time series as simple if it is possible to obtain Accuracy $= 1$ by comparing at least one dimension with at most two constants called upper and lower bounds that classify a point as normal or anomalous. Figure 1 contains a simple dataset whose horizontal dimension has a lower and upper bound, and the vertical dimension has only an upper bound. This definition enables the introduction of a score evaluating the degree of simplicity. The degree of simplicity is $max(TPR)$ @ $TNR = 1$ by placing lower and upper bounds on the dimensions of the dataset, i.e., the percentage of anomalies separable from normalities. This problem is analogous to finding the minimum bounding box for normal points: everything outside the bounding box is considered anomalous, and the percentage of anomalous points outside the bounding box is the degree of simplicity. The degree of simplicity is refined to evaluate a score of simplicity of the moving average or moving standard deviation, i.e., verifying whether some subsequences have an abnormally high or low mean or standard deviation. These scores give a spatial method-agnostic definition of simplicity, and allow the computation of such scores on derived series, such as the differentiation of the time series. Together with the definition of these scores, the thesis proposes algorithms for their computation. The proposed algorithms have worst-case complexity $\Theta(NF\log(N)+ANF)$ and best-case time complexity $\Theta(NF\log(N))$, where N is the number of points in the series, F is the number of features, and A is the number of anomalies. Moreover, the algorithms for calculating the moving average and standard deviation scores introduce a heuristic to define which lengths of the sliding windows to use in the computation of simplicity; therefore, their time complexities are equal to the previous multiplied by the number of windows to try. The heuristic is necessary for efficiency since the exact score can be obtained only by trying each window from 1 to the length of the series. Figure 2 contains the scores of simplicity of some publicly available datasets (the moving average score), showing similar results to those obtained by [7] stating that Yahoo, SMD, and NASA are simple datasets.

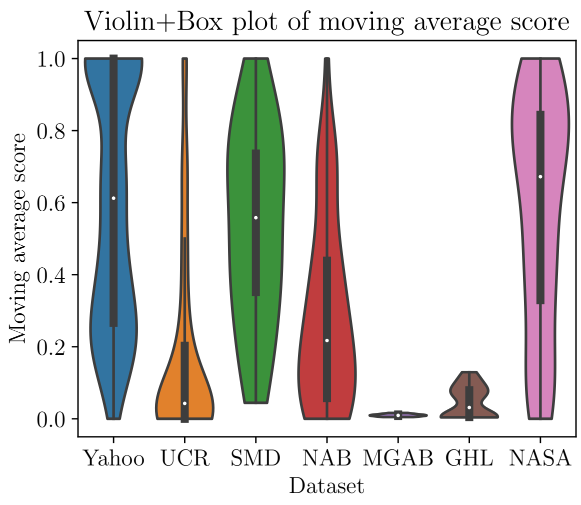

Figure 2: Simplicity scores of publicly available dataset searched on the series and its differentiations up to the third order. The higher, the simpler.

3. Anomalearn

Machine and deep learning libraries are beneficial tools for data science. They assist the data scientist in creating solutions in diverse ways: by providing the implementation of models and estimators in literature, furnishing transformation objects, and delivering objects for mixing the previous. Therefore, the data scientist can concentrate more on data analysis and modelling data rather than on implementing known approaches. However, among the machine learning libraries in Python, there is still a lack of two major components: data readers capable of scanning publicly available datasets and returning a standardized data structure, and a modular and extensible API for creating new models and related processors. Especially, anomaly detection has a surprisingly low number of published libraries: notable examples are Orion [2], PyOD [8], and Anomalib [1]. Of these three libraries, only Orion aims at easing the data scientist in inventing new models for time series anomaly detection. PyOD is a collection of anomaly detection algorithms mainly for tabular data, and Anomalib is a collection of deep learning algorithms for anomaly detection in images. Therefore, the availability of tools for developing new approaches is limited, and if the data scientist desires to follow an established technique for developing machine learning solutions for time series anomaly detection, the choice is forced. This preamble explains the reasons for creating and shipping a library aiding the data scientist in developing end-to-end approaches from data reading to label generation. Anomalearn is designed to be modular and achieve the lowest level of coupling. Figure 3 shows the UML package diagram of the top-level packages of anomalearn. There are two types of top-level packages:

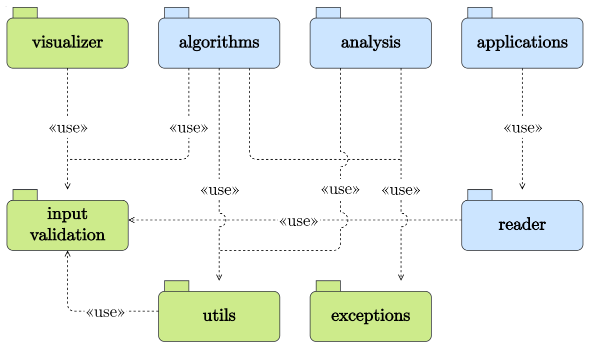

Figure 3: UML package diagram of the top-level packages of anomalearn. Blue elements are core packages, green elements are utility packages.

- Core packages: they offer the main functionalities of anomalearn ranging from preprocessing to hyper-parameter tuning.
- Utility packages: they offer utility functionalities and are mainly introduced for the needs of other packages or to ship all-in-one solutions.

For the development of such packages, the design choices were crucial. Since anomalearn aimed at supplying a helpful and simple approach, two strategies have been analysed: scikit-learn [4], and MLBlocks [5]. The former adopts a duck-typing structure: an object is described by the methods it exposes, not by its type. Therefore, it requires the developer to deeply know the conventions used for an object of a specific type, e.g., a predictor will always have the method `predict`, and there is no interface for that. The latter approach requires shipping both the code of the function or class implementing a machine learning procedure and a description of its entry points and outputs in JSON (i.e., an annotation file). The advantage of the second approach is that for developing a method, there is no convention regarding the naming of functions, but the data scientist needs to know the format of the annotation file to make the code work in the ecosystem. Anomalearn chooses an interface-based approach, trying to mix the advantages and reduce the disadvantages. Objects should implement one of the interfaces shipped with the library (e.g., classifiers implement `IClassifier`): they are portrayed by the implemented interfaces. However, interfaces subclass the behaviour of the subclass check of Python to consider any class implementing the same methods its subclass: it allows duck typing in some cases. Therefore, the developer needs neither to know any format for annotation files nor a massive amount of naming conventions: the interfaces expose methods along with their documentation with self-descriptive names (e.g., `IClassifier` exposes the method `classify`).

3.1. Dataset reading

Among the components, anomalearn ships diverse interfaces for readers of time series and time series anomaly detection datasets, together with some implementations. The output of a dataset reader is a pandas `DataFrame` [3, 6] composed of at least three and at most four types of columns. The three types of recurrent columns are:

- Index: it is the column containing the index of the time series.
- Data: it is the column or the set of columns containing the data of the time series (single column for univariate, multiple columns for multivariate).
- Class: it is the column containing the labels of points.

The optional column consists of the training column: it states the training points of the time series. This column is optional because some datasets do not ship a train/test split, and leave this duty to the data scientist. Conversely, when the dataset specifies it, it is reported in the optional column. Regarding the interface, a dataset reader is an iterator over the time series of the dataset: almost all publicly available time series anomaly detection datasets comprise multiple time series. The iterator returns the `DataFrame` of the series, and the reader also exposes methods to access specific time series, enabling the data scientist to define any different order of iteration. To enhance freedom of iteration, the reader exposes both the indexing operator [ ] and the method `read`.

3.2. Experiments

Over the data readers, anomalearn builds an object called `ExperimentLoader` for aiding the data scientist in composing experiments. This component exposes APIs to specify a sequence of data readers beside splits and a list of the series to utilize for each dataset. By default, it receives a list of data readers and exposes an iterator over the series of each reader: each series is returned once. However, since it is common practice to execute methods over a subset of the dataset and to define a different split for diverse datasets, it also allows the selection of the train/test split and the subset of series to return for each dataset.

3.3. Pipeline

The primary component of anomalearn is the pipeline. Among machine learning libraries, scikit-learn [4] and MLBlocks [5] implement their own version of a pipeline. Therefore, as with the interface-based approach, other approaches are examined to motivate the introduction of the pipeline of anomalearn. The former is a sequential pipeline: a sequence of transformers (objects transforming the input) followed by an estimator at the end (the model). Even if the approach is simple and intuitive, it has some drawbacks: a pipeline can contain only one estimator and cannot comprise transformers after the estimator. In addition, this limitation extends to the composition of pipelines: it is impossible to have chained pipelines containing estimators. This constraint hinders the creation of time series anomaly detection methods since they usually include postprocessing operations. The latter technique follows a similar structure to that of the MLBlocks [5] objects. A pipeline is a sequence of blocks: a set of annotation files. The input of the pipeline flows sequentially through the layers. To enable the data to flow between components, they employ a data structure called context dictionary: a dictionary in which keys are names of variables, and values are the contents. Each block has input and output variable names annotated in the JSON file, which makes the pipeline aware of which variables will be overwritten (by output variables) and which variables the current layer needs to execute. This intricate mechanism enables the creation of complex pipelines in which

data can undergo any transformation, and the addition of variables to the context dictionary such that the following layers will receive them as input. Again, the annotation files format must be known such that the overall pipeline works as expected. Moreover, neither the former nor the latter pipeline provide a safe saving method to serialize the learnt weights during the training. Utilizing pickle is mandatory.
Anomalearn provides a pipeline which does not require any annotation file, permits multiple models, and allows the utilization of postprocessing operations after models. These functionalities are offered by employing interfaces and abstract classes for the pipeline and layers. Therefore, any object insertable in a pipeline must inherit from one of the abstract classes for layers. Because of this abstraction, a pipeline must be independent of the number and concrete type of objects extending them. Interfaces and abstract classes suffice for implementing all the functionalities of a pipeline, and concrete objects received at run-time must not influence its operation. In addition, a pipeline must be a layer itself, enabling the composition of pipelines. Currently, the only implementation of the interface for pipelines is `Pipeline`. It is a sequential pipeline composed of any number and type of objects: it can include preprocessing, models, or postprocessing objects in any order and number (provided that they are compatible with each other). Besides its composition, once a pipeline has been trained, it can be serialized safely to disk thanks to the presence of two abstract classes for layers: serializable layers and layers that can only be instantiated. The realization of this feature agnostic to concrete types is offered through two utility functions, allowing to instantiate or load any estimator of the anomalearn library or any compliant object. These functions only need the name of the class and an optional list of class objects not included as part of anomalearn. Therefore, any object inheriting anomalearn interfaces not present in the anomalearn packages, can be added to the list to enable its loading and instantiation.

4. Conclusions

This thesis presented some methods to evaluate the simplicity of datasets and a library for developing time series anomaly detection models. The two contributions are distinct in the aim and approach: the library is more practical than theoretical. The methods for evaluating the simplicity of datasets furnish the data scientist with a set of tools for assessing the simplicity of datasets. These instruments not only allow the publication of a new dataset accompanied by a description of its complexity, but also permit to execute and evaluate methods on datasets at varying difficulty levels. Therefore, the data scientist is aided in the execution of a deeper analysis of the performance of methods by relating it to the three types of complexity of datasets.
The second contribution of this thesis, anomalearn, supplies the tools for creating time series anomaly detection models (or related objects) by minimizing the amount of code to write. Pipelines of anomalearn allow the data scientist to use all the implemented preprocessing, postprocessing, transformations and models for creating either new models or ensembles. Furthermore, the library ships data reader objects to ease the process of reading publicly available datasets and objects for creating experiments on a sequence of datasets with specified train/test split in any order. These components can be used individually and mixed for building experiments without requiring any code beyond that used to create the object and to iterate on the supplied iterators.

5. Acknowledgements

I would like to express my gratitude to Politecnico di Milano and the teaching staff for the excellent education I received as a Computer Engineer, to my supervisor Professor Piero Fraternali, and to my co-supervisor Nicolò Oreste Pinciroli Vago.
Moving on, I would like to acknowledge everyone who stood by me outside of the university. First and foremost, I am deeply grateful to my family.

References

[1] Samet Akcay, Dick Ameln, Ashwin Vaidya, Barath Lakshmanan, Nilesh Ahuja, and Utku Genc. Anomalib: A Deep Learning Library for Anomaly Detection, 2 2022.

[2] Sarah Alnegheimish, Dongyu Liu, Carles Sala, Laure Berti-Equille, and Kalyan Veera-

machaneni. Sintel: A machine learning framework to extract insights from signals. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 1855–1865. Association for Computing Machinery, 2022.

[3] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] Micah J. Smith, Carles Sala, James Max Kanter, and Kalyan Veeramachaneni. The machine learning bazaar: Harnessing the ml ecosystem for effective system development. *arXiv e-prints*, page arXiv:1905.08942, 2019.

[6] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[7] R. Wu and E. J. Keogh. 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.

[8] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.