SoftwareX 18 (2022) 101049

Contents lists available at ScienceDirect

SoftwareX

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

Original software publication

TSSEARCH: Time Series Subsequence Search Library

Duarte Folgado [a,b,*,1], Marília Barandas [a,b,1], Margarida Antunes [a], Maria Lua Nunes [a], Hui Liu [c], Yale Hartmann [c], Tanja Schultz [c], Hugo Gamboa [a,b]

[a] Associação Fraunhofer Portugal Research, Rua Alfredo Allen 455/461, Porto, Portugal
[b] LIBPhys (Laboratory for Instrumentation, Biomedical Engineering and Radiation Physics), NOVA School of Science and Technology (Campus de Caparica), 2829-516 Caparica, Portugal
[c] Cognitive Systems Lab, University of Bremen, Bremen, Germany

ARTICLE INFO

Article history:
Received 30 October 2021
Received in revised form 11 March 2022
Accepted 15 March 2022

Keywords:
Time series
Subsequence search
Distances
Similarity measurements
Query-based search
Segmentation
Python package

ABSTRACT

Subsequence search and distance measures are crucial tools in time series data mining. This paper presents our Python package entitled TSSEARCH, which provides a comprehensive set of methods for subsequence search and similarity measurement in time series. These methods are user-customizable for more flexibility and efficient integration into real deployment scenarios. TSSEARCH enables fast exploratory time series data analysis and was validated in the context of human activity recognition and indoor localization.

Code metadata

Current code version	v0.1.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00210
Code Ocean compute capsule	n/a
Legal Code License	BSD 3-Clause Licence
Code versioning system used	Git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	NumPy, Scipy, Matplotlib and Numba
If available Link to developer documentation/manual	https://tssearch.readthedocs.io
Support email for questions	info@fraunhofer.pt

Software metadata

1. Motivation and significance

Time series data mining algorithms use subsequence search as a subroutine. Subsequence search is a useful exploratory tool that has been used in many areas such as music information retrieval [1], gait analysis [2], and indoor localization [3] to find common patterns and behaviors in data.

Let us define two time series $Q := (q_1, q_2, \ldots, q_N)$ and $Y := (y_1, y_2, \ldots, y_M)$ of length N and M respectively. The subsequence search problem on time series can be formally defined as given a query Q and a longer target time series Y, find the set of best subsequences of Y, beginning at Y_i, that minimize a similarity measurement to Q, where i is the initial instant of each subsequence [4].

We present in this paper a Python package entitled TSSEARCH, which provides techniques for query-based subsequence search on time series, along with implementations of commonly used distance measures. TSSEARCH supports both univariate and multivariate time series and was written to be easily extendable by the community with additional time series distances or subsequence search methods.

In literature, there are related packages available with base distance measures for time series, such as TSdist [5]

* Corresponding author at: Associação Fraunhofer Portugal Research, Rua Alfredo Allen 455/461, Porto, Portugal.
E-mail address: duarte.folgado@fraunhofer.pt (Duarte Folgado).
[1] These authors contributed equally

https://doi.org/10.1016/j.softx.2022.101049

and `dtaidistance` [6]. Other complementary packages, such as `pyts` [7] and `sktime` [8], focus on supervised learning problems distinct from the search problem. Another example is `tslearn` [9], a general-purpose machine learning library with supporting tools for the complete development pipeline. However, none of the packages listed above focus on addressing the problem of subsequence search. This problem was addressed in `UCR Suite` [10], which enables optimized subsequence search on extensive datasets. However, its software implementation is not easily extendable. Compared to these packages, `TSSEARCH` focuses primarily on the subsequence search problem and performs highly customizable query-based searches with additional functionalities.

`TSSEARCH` is currently being used to segment repetitive motion in the context of human activity recognition [11] and find similar location patterns in indoor localization [3].

In the rest of this paper, we present an overview of the implemented architecture and functionalities, illustrative examples of elementary usage, and the impact and contribution to spark future research on the topic.

2. Software description

2.1. Software architecture

`TSSEARCH` is a cross-platform package for Python 3. It depends on `Numpy` [12] and `SciPy` [13], which provide efficient numerical routines for multi-dimensional data. It also relies on `Numba` [14] for efficient computation and `Matplotlib` [15] for visualization.

In order to perform a similarity search between time series, a distance measurement is required. Distance measures that compare the ith point of one time series to the ith point of another are denoted as *lock-step* measures (e.g., Euclidean distance and the other Lp norms). Distances that allow the comparison of one-to-many points (e.g., DTW and Longest Common Subsequence (LCSS)) are denoted as *elastic* measures [16]. Many forms of time series queries have been proposed over the years. For non-repetitive data, query search finds the occurrences of a query on an longer signal. For repetitive data, we shall refer to a slight variation as segmentation, whose goal is to identify the boundary time instants between consecutive repetitions.

In Fig. 1, we summarize the `TSSEARCH` processing pipeline. The data consists of time series represented through Numpy arrays of shape (T, D), where T is their length and D is their dimensionality. For multivariate time series, the dimensionality of the query and sequence must be the same. Usually, the length of the query is much smaller than the sequence. Users can choose the desired search methods or distances, either by individual selection or filtering based on the distance type. This search configuration is specified using a JSON formatted file. An example with a portion of the configuration file is illustrated in Listing 1. The syntax includes the distance name, a short description, the method name, associated parameters, and whether it supports multivariate input. The `use` key defines which distances are used for the search. If the user selects multiple distances, the routine will run separately for each distance. The output consists of a detailed report of the search. It describes the occurrences found by presenting the time instants where they are located, the optimal alignment path (for the case of *elastic* measures), and the distance to the query.

The package is divided into three main submodules: *tssearch.distances* contains the implementation of the similarity measures; *tssearch.search* contains the implementation of high-level routines for query search and segmentation; and *tssearch.utils*, which contains some utilities along with visualization methods to support the output report.

```
{
  "elastic": {
    "Dynamic Time Warping": {
      "multivariate": "yes",
      "description": "",
      "function": "dtw",
      "parameters": {
        "dtw_type": "dtw",
        "alpha": 1
      },
      "use": "yes"
    },
    "Longest Common Subsequence": {
      "multivariate": "yes",
      "description": "",
      "function": "lcss",
      "parameters": {
        "eps": 1,
        "report": "distance"
      },
      "use": "yes"
    }
  }
}
```

Listing 1: Example of the JSON configuration file with two implemented *elastic* distances.

Table 1
Overview of the implemented distances as of version v0.1.1.

Lock-step
Lp Distances
Cross Correlation Distance
Pearson Correlation Distance
Short Time Series Distance (STS) [17]
Elastic
Dynamic Time Warping (DTW)
Longest Common Subsequence (LCSS)
Time Warp Edit Distance (TWED) [18]
Time
Time Alignment Measurement (TAM) [19]

2.2. Software functionalities

2.2.1. Time series distances

A summary of the implemented time series distances included in v0.1.1 of `TSSEARCH` is presented in Table 1. These distances are used for the search routines. Nevertheless, if the user is only interested in measuring the distance between time series using any of these distances, it can be achieved using the *time_series_distance* method, which receives two time series and the configuration file declaring which distances will be computed.

2.2.2. Search methods

`TSSEARCH` addresses the time series search problem with two approaches. The *tssearch.time_series_search* method locates the k-best occurrences of a given query on a longer sequence based on a distance measurement. By default, k is set to retrieve the maximum number of matches. The user can also explicitly define the value of k to retrieve the k-best occurrences. The *tssearch.time_series_segmentation* locates the time instants between consecutive query repetitions on a longer and repetitive sequence.

2.2.3. Query sample weights

For some circumstances, it would be helpful to assign weights to each point of the query that measure their relative contribution to the overall distance. An example of use cases for this functionality would be finding a query where we are more uncertain on the shape of some intervals, thus assigning lower weights.

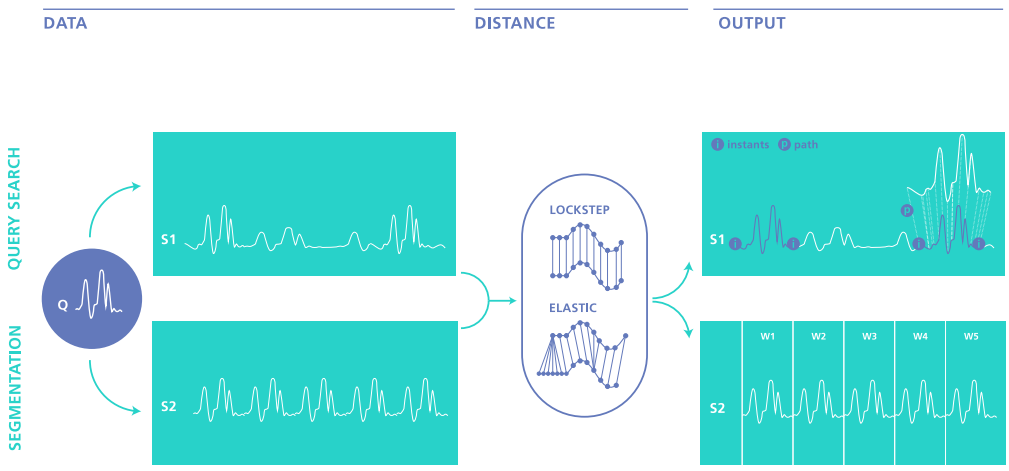

Fig. 1. The software searches for the occurrences of query Q in the $S1$ and $S2$ sequences. The user can customize the search by defining which elastic or lock-step distances will be used. The output consists of presenting the time instants where the query is found, the optimal alignment path (for the case of *elastic* measures), and the distance to the query. In the case of segmentation, it is assumed that the time series where the search is conducted are repetitive and, consequently, the objective is to identify the transition instants between subsequences.

2.2.4. Visualization

TSSEARCH implements visualization methods to present the results of the segmentation and query search. The visualization highlights the time instants where the matches are located and a color-coding representing their distance value to the query.

2.2.5. User customization

TSSEARCH provides flexibility for users to add their customized distances. Users can write the customized distance method and provide documentation that includes declaring the distance domain, either *lock-step* or *elastic*.

3. Illustrative examples

We now provide two illustrative examples of the software interaction and functionalities. The examples are available on a Google Colab, accessed from the package's GitHub repository.

3.1. Segmenting electrocardiography data

In this example, we used a ten-second segment from an ECG record. Several heartbeats are present in the recording. In order to simplify the example, we defined the query as one of the heartbeats present during the 10 s recording. Listing 2 presents an example code to perform the segmentation. In line 4, the user defines DTW as the distance for the segmentation. In line 6, the segmentation is calculated and the output assigned to a variable. The method receives as inputs the configuration file, the query, and the sequence. In this case, the user also specified a weights vector assigning less contribution to the second local maxima (T wave).

```
1  import tssearch
2
3  data = tssearch.load_ecg_example()
4  cfg = tssearch.get_distance_dict(["Dynamic Time
       Warping"])
5
6  out = tssearch.time_series_segmentation(cfg,
       data["query"], data["sequence"],
       weight=data["weight"])
```

Listing 2: Illustrative code example for query-based ECG segmentation.

The output visualization is represented in Fig. 2 and was generated using visualization methods available in the library.

For scenarios where the user might be interested in further characterizing each subsequence, this could be accomplished using the distances values calculated for each segment and/or using TSFEL [20] or MVTS [21] to extract temporal, statistical, and spectral features as data representations for classification algorithms.

3.2. Stride segmentation on walking data

We will now describe an additional example from a wearable sensor-based human activity dataset to further illustrate TSSEARCH's applicability on the content of kinesiology with an example of multidimensional data. The CSL-SHARE dataset covers 22 activities of daily living and sports from 20 subjects in a total time of 691 min. It contains synchronized multichannel biosignals recorded from various types of sensors with a relevant context for multidimensional subsequence searching [11].

In this example, stride segmentation is accomplished on a trial where a subject was walking. Two time series were considered for this multivariate query search example — acceleration data from an accelerometer and angular data from a goniometer attached to the subject's knee. Listing 3 presents an example code to perform the segmentation. In lines 7 and 8, the user defines the DTW with an additional parameter α that weights the contribution between the cost in the amplitude and its first derivative. In line 10, the query search is calculated, and the output is assigned to a variable. The method receives as inputs the configuration file, the query, and the sequence. Since the user did not explicitly define the number of matches, it retrieves the maximum number of matches.

```
1  import tssearch
2  import numpy as np
3
4  query = np.loadtxt("query.txt")
5  sequence = np.loadtxt("sequence.txt")
6
7  cfg = tssearch.get_distance_dict(["Dynamic Time
       Warping"])
8  cfg["elastic"]["Dynamic Time
       Warping"]["parameters"]["alpha"] = 0.5
9
10 out = tssearch.time_series_search(cfg, query,
       sequence)
```

Listing 3: Illustrative code example for query search applied to stride segmentation.

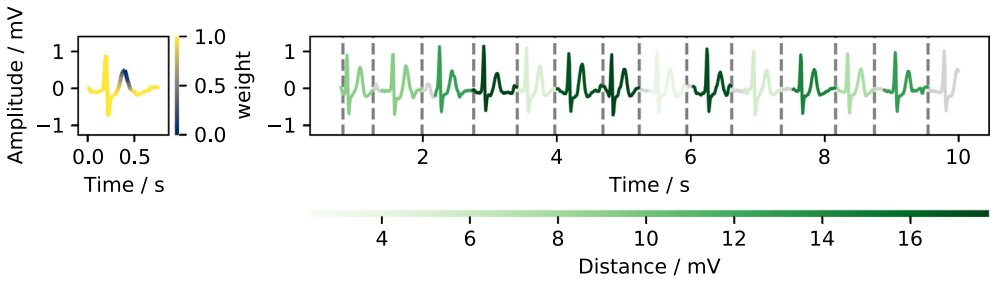

Fig. 2. Results of the query-based ECG segmentation example. The query is depicted on the left. The query was customized by the user, who assigned a region with relatively low importance (in blue) for the segmentation. The sequence is depicted on the right, and the vertical dashed lines represent the calculated segmentation instants. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

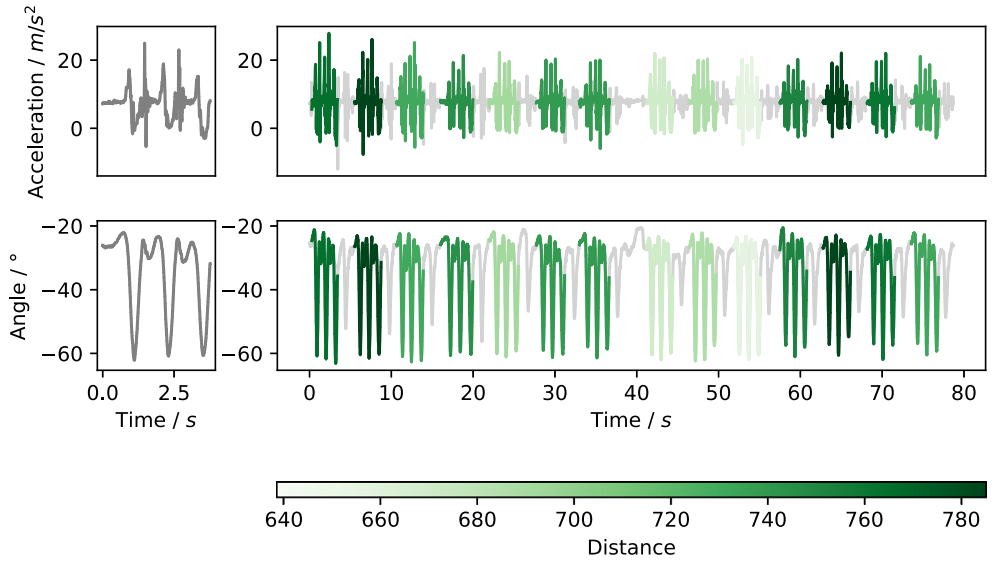

Fig. 3. Results of the multivariate query search applied to stride segmentation. The query and sequence are composed of two time series. The query is depicted on the left. The sequence is depicted on the right. The distance measures the similarity between the query and each match. The distance used was DTW.

The output visualization is represented in Fig. 3 and was generated using visualization methods available in the library.

4. Impact

The subsequence search is one of the most important subroutines for time series pattern mining. Subsequence search is used across different stages of the machine learning stack. In the initial stages, subsequence search can segment windows of interest, further characterized downstream using feature extraction methods. Furthermore, measuring the distance between a query and the segmented intervals provides quantitative data to perform downstream data mining tasks, such as clustering or supervised classification.

Specific applications include: financial, marketing, or stock price time series, where typical queries would be:

- Which companies have stock prices during a given time interval similar to our company?
- Did we have in the past similar cases that resemble the last month's sales pattern of our product?

Typical queries for scientific datasets composed of time series would be:

- Which past days showed earthquake activity similarly to today's pattern?
- Given a heartbeat irregularity defined by the query, how many similar heartbeat irregularities did the patient have during the exam?
- During a walking exercise, how much stride variability does exist in healthy subjects and patients with a neurological condition?

`TSSEARCH` helps time series data scientists by providing a set of methods for subsequence search and similarity measurement. Those methods were previously validated in the context of human activity recognition [11] and indoor localization [3]. `TSSEARCH` is also easily extendable so that advanced users can create additional search or distance measurement methods customized to their needs. The parameters for subsequence search experiments are also recorded in a configuration file, promoting the openness and reproducibility of the scientific method among researchers. For scenarios where the user after the subsequence search is interested in further characterizing each utterance, `TSSEARCH` was written to work along with `TSFEL` [20], a Python package dedicated to time series feature extraction. For this use case, the output is a standardized file format ready to be digested by classification libraries like `Orange` [22], `Weka` [23], or `scikit-learn` [24].

5. Conclusions

We have developed a Python package entitled `TSSEARCH`, which provides a comprehensive set of methods for subsequence search and similarity measurement in time series. `TSSEARCH` includes unit tests and proper code documentation. Besides these strongly oriented coding guidelines, this project has three main

contributions: (1) create a dedicated Python package to address the problem of subsequence search in univariate and multivariate time series; (2) incorporation of weights to customize the importance of given query intervals and (3) an easily extendable API to allow the customization of distance measurements and search methods.

TSSEARCH is open for contributions and invites users to extend the library with new subsequence search methods and similarity measurements across different scientific disciplines.

TSSEARCH is open-source, available online at GitHub, well-documented, and constantly maintained. The presented illustrative examples have shown that it might be a powerful tool dedicated to the problem of subsequence search in time series.

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.

Acknowledgments

This article is a result of the project ConnectedHealth (n.° 46858), supported by Competitiveness and Internationalisation Operational Programme, Portugal (POCI) and Lisbon Regional Operational Programme (LISBOA 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF).

References

[1] Müller M. Fundamentals of music processing. Cham: Springer International Publishing; 2015, http://dx.doi.org/10.1007/978-3-319-21945-5.

[2] Barth J, Oberndorfer C, Pasluosta C, Schülein S, Gassner H, Reinfelder S, et al. Stride segmentation during free walk movements using multi-dimensional subsequence dynamic time warping on inertial sensor data. Sensors 2015;15(3):6419–40. http://dx.doi.org/10.3390/s150306419.

[3] Santos R, Leonardo R, Barandas M, Moreira D, Rocha T, Alves P, et al. Crowdsourcing-based fingerprinting for indoor location in multi-storey buildings. IEEE Access 2021;9:31143–60. http://dx.doi.org/10.1109/ACCESS.2021.3060123.

[4] Keogh E, Chakrabarti K, Pazzani M, Mehrotra S. Dimensionality reduction for fast similarity search in large time series databases. Knowl Inform Syst 2001;3(3):263–86. http://dx.doi.org/10.1007/PL00011669.

[5] Mori U, Mendiburu A, Lozano J. Distance measures for time series in R: The tsdist package. R J 2016;8(2):451. http://dx.doi.org/10.32614/RJ-2016-058.

[6] Meert W, Hendrickx K, Craenendonck TV. Wannesm/dtaidistance v2.0.0. 2020, http://dx.doi.org/10.5281/ZENODO.3981067.

[7] Faouzi J, Janati H. Pyts: A python package for time series classification. J Mach Learn Res 2020;21:1–46, URL https://www.jmlr.org/papers/v21/19-763.html.

[8] Löning M, Bagnall A, Ganesh S, Kazakov V, Lines J, Király FJ. Sktime: A unified interface for machine learning with time series. 2019, arXiv:1909.07872 [Cs, Stat].

[9] Tavenard R, Faouzi J, Vandewiele G, Divo F, Androz G, Holtz C, et al. Tslearn, a machine learning toolkit for time series data. J Mach Learn Res 2020;21(118):1–6, URL https://jmlr.org/papers/v21/20-091.html.

[10] Rakthanmanon T, Campana B, Mueen A, Batista G, Westover B, Zhu Q, et al. Searching and mining trillions of time series subsequences under dynamic time warping. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining. Beijing, China: ACM Press; 2012, p. 262. http://dx.doi.org/10.1145/2339530.2339576.

[11] Liu H, Hartmann Y, Schultz T. CSL-SHARE: A multimodal wearable sensor-based human activity dataset. Front Comput Sci 2021;3:90. http://dx.doi.org/10.3389/fcomp.2021.759136.

[12] Harris CR, Millman KJ, Van Der Walt SJ, Gommers R, Virtanen P, Cournapeau D, et al. Array programming with NumPy. Nature 2020;585(7825):357–62. http://dx.doi.org/10.1038/s41586-020-2649-2.

[13] Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, et al. SciPy 1.0: Fundamental algorithms for scientific computing in python. Nat Methods 2020;17(3):261–72. http://dx.doi.org/10.1038/s41592-019-0686-2.

[14] Lam SK, Pitrou A, Seibert S. Numba: A LLVM-based python JIT compiler. In: Proceedings of the second workshop on the LLVM compiler infrastructure in HPC. 2015, p. 1–6. http://dx.doi.org/10.1145/2833157.2833162.

[15] Hunter JD. Matplotlib: A 2D graphics environment. Comput Sci Eng 2007;9(03):90–5. http://dx.doi.org/10.1109/MCSE.2007.55.

[16] Wang X, Mueen A, Ding H, Trajcevski G, Scheuermann P, Keogh E. Experimental comparison of representation methods and distance measures for time series data. Data Min Knowl Discov 2013;26(2):275–309. http://dx.doi.org/10.1007/s10618-012-0250-5.

[17] Möller-Levet CS, Klawonn F, Cho K-H, Wolkenhauer O. Fuzzy clustering of short time-series and unevenly distributed sampling points. In: International symposium on intelligent data analysis. Springer; 2003, p. 330–40. http://dx.doi.org/10.1007/978-3-540-45231-7_31.

[18] Marteau P-F. Time warp edit distance with stiffness adjustment for time series matching. IEEE Trans Pattern Anal Mach Intell 2008;31(2):306–18. http://dx.doi.org/10.1109/TPAMI.2008.76.

[19] Folgado D, Barandas M, Matias R, Martins R, Carvalho M, Gamboa H. Time alignment measurement for time series. Pattern Recog 2018;81:268–79. http://dx.doi.org/10.1016/j.patcog.2018.04.003.

[20] Barandas M, Folgado D, Fernandes L, Santos S, Abreu M, Bota P, et al. TSFEL: Time series feature extraction library. SoftwareX 2020;11:100456. http://dx.doi.org/10.1016/j.softx.2020.100456.

[21] Ahmadzadeh A, Sinha K, Aydin B, Angryk RA. MVTS-data toolkit: A python package for preprocessing multivariate time series data. SoftwareX 2020;12:100518. http://dx.doi.org/10.1016/j.softx.2020.100518.

[22] Demšar J, Curk T, Erjavec A, Črt Gorup, Hočevar T, Milutinovič M, Možina M, et al. Orange: Data mining toolbox in python. J Mach Learn Res 2013;14:2349–53, URL http://jmlr.org/papers/v14/demsar13a.html.

[23] Frank E, Hall M, Holmes G, Kirkby R, Pfahringer B, Witten IH, et al. Weka-A machine learning workbench for data mining. In: Data mining and knowledge discovery handbook. Springer; 2009, p. 1269–77. http://dx.doi.org/10.1007/978-0-387-09823-4_66.

[24] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in Python. J Mach Learn Res 2011;12:2825–30, URL https://www.jmlr.org/papers/v12/pedregosa11a.html.