sktime: A Unified Interface for Machine Learning with Time Series

Markus Löning
The Alan Turing Institute

Anthony Bagnall
University of East Anglia

Sajaysurya Ganesh
University College London

Viktor Kazakov
University College London

Jason Lines
University of East Anglia

Franz J. Király[*]
The Alan Turing Institute

Abstract

We present sktime – a new scikit-learn compatible Python library with a unified interface for machine learning with time series. Time series data gives rise to various distinct but closely related learning tasks, such as forecasting and time series classification, many of which can be solved by reducing them to related simpler tasks. We discuss the main rationale for creating a unified interface, including reduction, as well as the design of sktime's core API, supported by a clear overview of common time series tasks and reduction approaches.

1 Introduction

Data scientific tasks beyond the standard tabular setting are one of the major challenges of contemporary machine learning. sktime[2] is a new open-source Python library for machine learning with time series. Our goal is to extend existing machine learning capabilities, most notably scikit-learn [16], to the temporal data setting by providing a unified interface for several time series learning tasks.

Time series data is ubiquitous in many applications. Examples include sensor readings from industrial processes, spectroscopy wave length data from chemical samples, or bed-side monitor medical data from patients. There is a broad variety of distinct but closely related learning tasks that arise in such contexts, including time series classification, forecasting and annotation among others. In section 2, we give a more detailed overview of time series tasks. The ambition of the project is to design and implement an API (application programming interface) that unifies these tasks.

A plethora of time series toolboxes exists that provide rich interfaces to specific model classes (ARIMA/filters [50, 54], neural networks [2]), or framework interfaces to isolated time series modelling tasks (forecasting [55, 29], feature extraction [36, 53, 19], annotation [17], classification [17, 53]).[3] Nevertheless, open-source machine learning capabilities for time series are still limited and existing libraries are often incompatible with each other. To the best of our knowledge, we are the first to present a unified interface that can explicitly represent and link multiple distinct tasks.

The main rationale for creating a unified API, as opposed to separate task-specific interfaces, is as follows: First, many time series learners are highly composite and often involve reduction from complex learning task (e.g. time series segmentation) to related simpler tasks (e.g. supervised learning). We describe exemplar reduction approaches with time series data in greater depth in section 3. A unified and composable interface for different tasks enables us to encapsulate reduction approaches as meta-estimators, exposing their implicit modeling choices as tunable hyper-parameters, and thus

[*]Corresponding author: `fkiraly@turing.ac.uk`

[2]`https://github.com/alan-turing-institute/sktime`

[3]For a more extensive and regularly updated overview of Python time series related libraries, see `https://github.com/alan-turing-institute/sktime/wiki/Related-software`.

enabling us to easily evaluate and compare them against other strategies. Second, the current lack of a unified API leads to unnecessary code replications and often error prone and statistically inappropriate reductions to those tasks that existing off-the-shelf toolboxes can deal with (e.g. reductions to tabular data supported by scikit-learn). A single API reduces confusion and enables us to focus on providing advanced time series analysis capabilities for researchers and practitioners. In addition, many tasks require common functionality such as distance measures and preprocessing routines. Providing them in a consistent and modular interface allows us to re-utilise them across different settings.

As of now, sktime includes state-of-the-art algorithms for time series classification, additional modular functionality for reduction, pipelining, ensembling and data transformations, as well as forecasting methods and benchmarking tools.

We first present in section 2 an overview of the most common time series tasks, and then discuss, in section 3, reduction as an approach to solving these tasks. Section 4 outlines the key design features built into the core interface. Section 5 describes the currently available functionality. We conclude by previewing future work in section 6.

2 Taxonomy of time series learning tasks

sktime provides tooling for learning with time series, i.e. data observed at a finite number of known time points. More precisely, whenever we refer to time series, we consider them explicitly consisting of both (i) time points at which they are observed and (ii) observations at those time points. In ad-hoc short-hand notation, we write, for example, $x(t_1), x(t_2), ..., x(t_T)$ for observations at time points $t_1, t_2, ..., t_T$, and $\mathbf{x}$ for the time series object that contains exactly that information.[4] An intrinsic characteristic of time series is that observations within time series are statistically dependent in assumed generative processes (which we avoid to introduce here to keep notation simple). Due to this dependency, time series data does not naturally fit into the standard machine learning framework for tabular (or cross-sectional) data, which implicitly assumes observations to be independent and identically distributed (i.i.d.). Consequently, many toolboxes for learning on tabular data, such as scikit-learn, consider learning with time series out of scope [16].

When learning with time series, it is important to understand the different forms such data may take. The data can come in the form of a single (or univariate) time series; but in many applications, multiple time series are observed. It is crucial to distinguish the two fundamentally different ways in which this may happen:

- **Multivariate time series data**, where two or more variables are observed over time, with variables representing *different kinds of measurements* within a single experimental unit;
- **Panel data**, sometimes also called longitudinal data, where multiple *independent instances* of the *same kinds of measurements* are observed, e.g. time series from multiple industrial processes, chemical samples or patients.[5]

In multivariate data, it is implausible to assume the different univariate component time series are i.i.d. In panel data, the i.i.d. assumption applied to the different instances is plausible, while time series observations within a given instances may still depend on adjacent observations. In addition, panel data may be multivariate, which corresponds to i.i.d. instances of multivariate time series. In this case, the different instances are i.i.d., but the univariate component series within an instance are not. This richness of generative scenarios is mirrored in a richness of learning tasks applicable to such data. We later show that these tasks are closely related through reduction, but first highlight some of the most common ones here:

- **Time series regression/classification.** We observe N i.i.d. panel data training instances of feature-label pairs $(\mathbf{x}_i, y_i)$, $i = 1 \ldots N$. Each instance of features is a time series $\mathbf{x}_i = (x_i(t_1) \ldots x_i(t_T))$. The task is to use the training data to learn a predictor $\hat{f}$ that can accurately predict a new target value, such that $\hat{y} = \hat{f}(\mathbf{x}_*)$ for a new input time series $\mathbf{x}_*$. For

[4]Formally, our data model for time series is that of a list of time/value pairs, or equivalently, a time indexed list or dictionary, with a finite set of indices.

[5]One complication is that observed time points may vary across variables and/or instances. While time-heterogeneous settings are not covered by our notation, they are covered by the sktime interface.

regression, $y_i \in \mathbb{R}$. For classification, y_i takes a value from a finite set of class values. Additionally, time-invariant features may be present. Compared to the tabular supervised setting, the only difference is that some features are time series, instead of being only primitives (e.g. numbers, categories or strings). Important sub-cases are (i) equally spaced observation times and (ii) equal length time series. For an overview of time series classification, see [3, 25].

- **Classical forecasting.** Given past observations $\mathbf{y} = (y(t_1) \dots y(t_T))$ of a single time series, the task is to learn a forecaster $\hat{f}$ which can make accurate temporal forward predictions $\hat{y} = \hat{f}(h_j)$ of observations at given time points $h_1 \dots h_H$ of the forecasting horizon, where $\hat{\mathbf{y}} = (\hat{y}(h_1) \dots \hat{y}(h_H))$ denotes the forecasted series. No i.i.d. assumption is made. Variants may be distinguished by the following: (i) whether one observes additional related time series (multivariate data); (ii) for multivariate data, whether one forecasts a single series or multiple series jointly (exogeneity vs vector forecasting) [45]; (iii) whether the forecasting horizon lies in the observed time horizon (in-sample predictions), in the future of the observed time series (forecasting), or for multivariate data, only in the future of the target variable but not the exogenous variables (nowcasting); (iv) whether there is a single time point to forecast ($H = 1$) or not (single-step vs multi-step); (v) whether the forecasting horizon is already known during training or only during forecasting (functional vs discrete forecast). For an overview of classical forecasting, see [14, 15, 35, 21].

- **Supervised/panel forecasting.** We observe N i.i.d. panel data training instances $(\mathbf{y}_i)$, $i = 1 \dots N$. Each instance is a sequence of past observations $\mathbf{y}_i = (y_i(t_1) \dots y_i(t_T))$. The task is to use the training data to learn a supervised forecaster $\hat{f}$ that can make accurate temporal forward predictions $\hat{y}_i = \hat{f}(\mathbf{y}_*, h_j)$ for a new instance $\mathbf{y}_*$ at given time points $h_1 \dots h_H$ of the forecasting horizon, where $\hat{y}_i = (\hat{y}_i(h_1) \dots \hat{y}_i(h_H))$ is the forecasted series. Variants include panel data with additional time-constant features and the same variants as found in classical forecasting. For an overview, see [5, 58, 24].

- **Time series annotation.** For given observations $\mathbf{x} = (x(t_1) \dots x(t_T))$ of a single time series, the task is to learn an annotator that accurately predicts a series of annotations $\hat{\mathbf{y}} = (\hat{y}(a_1) \dots \hat{y}(a_A))$ for the observed series $\mathbf{x}$, where $a_1 \dots a_A$ denotes the time indices of the annotations. The task varies by value domain and interpretation of the annotations $\hat{\mathbf{y}}$ in relation to $\mathbf{x}$: (i) in change-point detection, $\hat{\mathbf{y}}$ contains change points and the type of change point [30]; (ii) in anomaly detection, the a_j are a sub-set of the t_j and indicate anomalies, possibly with the anomaly type [14]; (iii) in segmentation, the a_j are interpreted to subdivide the series $\mathbf{x}$ into segments, annotated by the type of segment [39]. Time series annotation is also found in supervised form, with partial annotations within a single time series, or multiple annotated i.i.d. panel data training instances [23, 28].

3 Reductions with time series

While these tasks define distinct learning settings, they are closely related, which enables us to solve them via reduction. Reduction essentially decomposes a given task into simpler tasks so that solutions to the simpler tasks can be composed to give a solution to the original task. A classical example of reduction in tabular supervised learning is one-vs-all classification, reducing k-way multi-category classification to k binary classification tasks [11, 10, 9]. For time series, a common example of reduction is to solve classical forecasting through time series regression via a sliding window approach and iteration over the forecasting horizon [12]. Many reduction approaches are possible with time series, we highlight some of the most important ones in figure 1.

Reduction offers several key advantages with regard to API design [8, 11]: First, reductions convert any algorithm for a particular task into a learning algorithm for the new task. Any progress on the base algorithm immediately transfers to the new task, saving both research and software development effort. Second, reductions are modular and composable. Applying some reduction approach to n base algorithms gives n new algorithms for the new task. Reductions can be composed to solve more complicated problems, e.g. first reducing forecasting to time series regression which in turn can be reduced to tabular regression. Finally, reductions also help us better understand the relationship between tasks and reduce confusion between them.

Figure 1: Stylised overview of time series reduction approaches

$$\begin{tikzcd}
\text{classical forecasting} \arrow[r, shift left, "b"] \arrow[d, "a"'] & \text{supervised forecasting} \arrow[l, shift left, "c"] \arrow[r, "d"] \arrow[d, "a"'] & \text{time series regression} \arrow[r, "g"] \arrow[d, "f"'] & \text{tabular regression} \arrow[d, "f"'] \\
\text{time series annotation} \arrow[r, shift left, "b"] & \text{supervised annotation} \arrow[l, shift left, "c"] \arrow[r, "e"] & \text{time series classification} \arrow[r, "g"] & \text{tabular classification} \\
\underbrace{\hphantom{xxxxxxxx}}_{\text{single time series}} & \underbrace{\hphantom{xxxxxxxxxxxxxxxxxxxxxxxx}}_{\text{panel data}} & & \underbrace{\hphantom{xxxxxxxx}}_{\text{tabular data}}
\end{tikzcd}$$

Notes: (a) annotate time series with future values, (b) rolling window method to convert single series into panel data with multiple output time periods [12], (c) ignore training set (e.g. fit forecaster on test set only) or use training set for model selection, (d) iterate over output periods, optionally time binning/aggregation of output periods [12], (e) rolling window method to convert single series into panel data with single output period [23], (f) discretise output into one or more bins, (g) feature extraction [26, 19] or time binning/aggregation of input time points.

4 API design

The main goal of sktime is to create a unified API for multiple time series tasks, extending the common scikit-learn interface to the temporal setting, while staying close to its syntax and logic whenever possible. Following scikit-learn's API allows us to re-utilise many of the algorithms available in scikit-learn, which is especially useful because of reduction to tabular tasks and because many specialised algorithms for time series are composites with tabular supervised learning algorithms as their components. The key design features of the sktime API are as follows:

4.1 Data representation

Any machine learning library relies fundamentally on some data representation and the lack of powerful data structures for time series data has arguably been one of the main reasons for the lack of a unified interface. In order to combine different tasks and data formats, sktime requires a data container capable of handling multivariate and panel data with additional time-constant features, including time-heterogeneous time series, where the observed lengths and time points varies across instances and/or variables. While pandas [48] handles time series data, it is intended to store time series only in the long format, with rows representing time points and columns representing variables, precluding time-heterogeneous data. Technically, however, it is possible to store arbitrary types in the cells of pandas containers. Inspired by xpandas [20], we chose to exploit this feature and represent time series data in a nested format, with rows representing i.i.d. instances and columns representing different variables as before, but with cells now no longer representing only primitives but also entire time series. The reason for this choice is twofold: First, we can still make use of pandas, one of the most comprehensive and efficient data container in Python, while at the same time having a consistent data representation across different tasks which is flexible enough to handle multivariate, panel and time-heterogeneous data. Second, as rows still represent i.i.d. instances, this representation allows us to reuse many of the existing functionality in scikit-learn. Alternatives we considered but ultimately set aside include three-dimensional NumPy arrays [56] as used in [53] and xarray [34, 33], an extension of pandas to panel data, both however only support time-homogeneous data.[6]

4.2 Task-specific estimators

We follow scikit-learn [57, 49] and Weka [32, 31] in adopting a uniform basic API for estimators, consisting of a fit method used for learning a model from training data and a predict method used for making predictions based on the fitted model, as well as a common interface for setting and retrieving hyper-parameters. Estimators in sktime are task specific, extending scikit-learn's regressors and classifiers to their time series counterparts as well as adding new estimators such as forecasters and supervised forecasters among others, with the same fit and predict methdods, but varying function signatures and internal behaviour.

[6]Other pandas-based data containers we considered include entity sets from Featuretools [36] and pysf [29].

4.3 Transformers

Similar to estimators, transformers have a uniform API consisting of a fit and a transform method used to transform input data, and a corresponding method for the inverse transformation if available, in addition to the common hyper-parameters interface. Since many data transformations are applicable for different tasks, we develop a unified transformer interface. To reconcile the different settings, we introduce the following kinds of transformations, distinguishing between fitting over i.i.d. instances and fitting over time points.

- **Tabular.** scikit-learn like transformers, which operate over i.i.d. instances and are fitted during training (e.g. principal component analysis).
- **Series-to-primitives.** Operates over time points, transforms time series for each instance into a primitive number (e.g. feature extraction). If the transformer is fittable, it is fitted separately for each instance during both training and prediction.
- **Series-to-series.** Like series-to-primitives, but output of transformation is itself a series instead of a primitive (e.g. Fourier transform or series of fitted auto-regressive coefficients).
- **Detrending.** Operates over time points and transforms an input time series, returning a detrended time series in the same domain as the input series (e.g. polynomial or seasonal detrending). Detrending tranformers keep track of the time index seen in fitting, so that trends are correctly computed over new time indices when transforming new data. In forecasting, it is fitted only during training. For panel data, it can be applied like a series-to-series transformation by iterating over instances. Detrenders are designed to take forecasters as input arguments, so that they internally first fit the forecaster to the input series and then return the residuals after subtracting the in-sample forecasts from the input series. For example, to detrend a time series with an exponential smoothing forecaster in sktime, we can write:

  ```
  t = Detrender(forecaster=ExponentialSmoothingForecaster())
  yt = t.fit_transform(y)
  ```

  where the given input series `y` is transformed into `yt`, consisting of the residuals of the exponentially smoothed in-sample forecasts.

4.4 Composition

Many learning strategies are expressible as meta-estimators, including pipelines, ensembles, reduction approaches and model selection routines like grid-search cross-validation. As in scikit-learn, meta-estimators wrapping estimators are estimators themselves with the same uniform API, as well as support for setting and getting hyper-parameters of the wrapped estimators.

sktime is the first toolbox, as far as we know, that provides reduction strategies as composable meta-estimators with an explicit hyper-parameter interface. Reduction typically introduces implicit modelling choices (e.g. the window width and step length of the sliding operation when reducing forecasting to time series regression). Defining reduction approaches as meta-estimators enables us to expose these modelling choices as hyper-parameters, which allows us not only to easily compose different reduction strategies with configurable components, but also to optimise these modelling choices via common model selection techniques. For example, reducing classical forecasting to time series regression in sktime looks as follows:

```
f = ReducedRegressionForecaster(regressor=RandomForestRegressor(),
                                method='recursive', window_length=2)
f.fit(y)
y_hat = f.predict(fh=fh)
```

where `y` is an input series, `fh` the forecasting horizon and `y_hat` the forecasted series. `ReducedRegressionForecaster` is a meta-estimators with explicit, tunable hyper-parameters for determining the window length and the method for iterating over multiple output periods of the forecasting horizon (here set to the recursive approach as described in [12]). Since `ReducedRegressionForecaster` is a forecaster itself, it could be passed into a temporal grid-search cross-validation routine [7] in order to optimise the hyper-parameters. Through these composition interfaces for reduction, sktime's API allows to solve a wide variety of learning tasks with a small amount of easy-to-read code.

In addition, we introduce new modular composition meta-estimators for multivariate time series. This includes column-wise ensembling, where a different estimator is applied to each time series variable inspired by scikit-learn's column transformer, and column concatenation, where multiple time series columns are concatenated into a single, long time series column.

5 API overview

sktime is in an early stage of development and currently includes the following functionality:

- **Time series classification.** State-of-the-art algorithms, including
  - **Interval based.** Time series forest [22] and random interval spectral ensemble [43].
  - **Distance based.** Distance measures form a fundamental primitive of many time series tasks. We have implemented eight distance measures in Cython [6] for enhanced performance and several classifiers that use them, including the Elastic Ensemble [42], Proximity Forest [44] and all the kernel methods described in a recent survey paper [1].
  - **Shapelet based.** The shapelet package includes an implementation of the shapelet transform [13] and prototypes for learning shapelets [27] and Shapelet Forest [37].
  - **Dictionary based.** The symbolic aggregate approximation (SAX) [40] and symbolic Fourier approximation (SFA) [52] transform and the associated bag of patterns (BOP) [41] and bag of SFA symbols (BOSS) [51] classifiers.
  - **Deep learning.** A recent review paper described nine deep learning algorithms for time series classification [25]. With the help of the authors, we have ported these algorithms into sktime in a deep learning extension package[7] based on keras [18].
- **Classical forecasting.** We have implemented a range of statistical forecasting techniques, interfacing statsmodels [50] whenever possible, and reduction strategies to utilise supervised learning algorithms.
- **Transformers.** Various transformers have been implemented for segmenting time series, series-to-primitives and series-to-series feature extraction and detrending. To reduce time series regression/classification to tabular supervised learning, a transformer for time binning input series is included.
- **Composition.** Pipelining for both feature and target variables following scikit-learn's API and multivariate composite strategies have been implemented.
- **Benchmarking.** Inspired by the mlaut package [38], sktime includes tools for automatic orchestration of prediction experiments evaluating one or more models on one or more data sets, with post-hoc statistical methods for comparing predictive performances.

The majority of the implemented algorithms has been tested for correctness against implementations in other languages and benchmarked on archive data [4]. We have reproduced the results presented in a comparative benchmarking study [3] and are in the process of recreating results from forecasting benchmarking studies [47, 46].

6 Conclusion and future directions

We have discussed the main rationale for a unified interface for machine learning with time series and outlined the key features of sktime's API, including new meta-estimators for reduction and multivariate ensembling. For future development, there are several directions the sktime project aims to focus on and we are actively looking for contributors to implement task-specific interfaces and reduction approaches. At present, most methods in sktime only support data with equal length series and no missing values. We aim to extend existing functionality to cover these situations as well. In addition, the composite structure of many of the implemented time series classifiers allows us to easily refactor them into their regressor counterparts. Having designed and implemented the key building blocks of the API for time series classification/regression and classical forecasting, the next major addition to sktime will be supervised forecasting, based on a modified pysf interface [29]. Finally, many implemented tools in sktime (e.g. distance measures) can be re-utilised for related unsupervised learning task, including time series clustering and motif discovery.

[7] `https://github.com/uea-machine-learning/sktime-dl`

Authors' contributions

ML made key contributions to architecture and design, including composition and reduction interfaces. ML is also one of sktime's core contributors and maintainers, having implemented, and contributing to, almost all parts of it, including the overall framework, the forecasting module, and specific algorithms. ML drafted and wrote most of this manuscript, partly based on sketches and presentation content by FK.

FK conceived the project and architectural outlines, including task taxonomy, composition and reduction. FK further made key contributions to architecture and design, and contributed to writing of this manuscript.

AB implemented time series forest and the random interval spectral ensemble, and contributed design ideas and to writing of this manuscript.

JL implemented and conceived modularised interfaces of several algorithms: distance based algorithms, including time series k-nearest-neighbours, Cython implementations of time series distance functions, and the shapelet transform.

SG contributed to the initial design and implementation of the time series classification setting, as well as implementation of the overall framework.

VK contributed to the design and implementation of the benchmarking module based on the mlaut package.

All authors reviewed the manuscript and participated in final copy-editing and proof-reading.

Acknowledgements

We would like to thank all participants of the 2019 joint sktime/MLJ development sprint who helped implement various time series classification algorithms, including Amaia Abanda Elustondo, Aaron Bostrom, Saurabh Dasgupta, David Guijo-Rubio, James Large, Matthew Middlehurst, George Oastler, Piotr Oleśkiewicz, Mathew Smith and Jeremy Sellier.

The first phase of development for sktime was done jointly between researchers at the University of East Anglia (UEA), University College London (UCL) and The Alan Turing Institute as part of a UK Research and Innovation (UKRI) project to develop tools for data science and artificial intelligence.

Markus Löning's contribution was supported by the Economic and Social Research Council (ESRC) [grant: ES/P000592/1], the Consumer Data Research Centre (CDRC) [ESRC grant: ES/L011840/1], and The Alan Turing Institute (EPSRC grant no. EP/N510129/1).

References

[1] A. Abanda, U. Mori, and J. Lozano. A review on distance based time series classification. *Data Mining and Knowledge Discovery*, 33(2):378–412, 2019.

[2] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, et al. Gluonts: Probabilistic time series models in python. *arXiv preprint arXiv:1906.05264*, 2019.

[3] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

[4] Anthony Bagnall, Markus Löning, Matthew Middlehurst, and George Oastler. A tale of two toolkits, report the first: benchmarking time series classification algorithms for correctness and efficiency. *arXiv preprint arXiv:1909.05738*, 2019.

[5] Badi H. Baltagi. *Econometric Analysis of Panel Data*. John Wiley & Sons, 4 edition, 2008.

[6] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2):31–39, 2011.

[7] Christoph Bergmeir, Rob J Hyndman, and Bonsoo Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83, 2018.

[8] Alina Beygelzimer, Varsha Dani, Tom Hayes, John Langford, and Bianca Zadrozny. Error limiting reductions between classification tasks. In *Proceedings of the 22nd international conference on Machine learning*, pages 49–56. ACM, 2005.

[9] Alina Beygelzimer, Hal Daumé, John Langford, and Paul Mineiro. Learning reductions that really work. *Proceedings of the IEEE*, 104(1):136–147, 2015.

[10] Alina Beygelzimer, John Langford, and Bianca Zadrozny. Weighted one-against-all. In *American Association for Artificial Intelligence (AAAI)*, pages 720–725, 2005.

[11] Alina Beygelzimer, John Langford, and Bianca Zadrozny. Machine learning techniques—reductions between prediction quality metrics. In *Performance Modeling and Engineering*, pages 3–28. Springer, 2008.

[12] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. Machine Learning Strategies for Time Series Forecasting. In *Business Intelligence*, pages 62–77. Springer, Berlin, Heidelberg, 2013.

[13] A. Bostrom and A. Bagnall. Binary shapelet transform for multiclass time series classification. *Transactions on Large-Scale Data and Knowledge Centered Systems*, 32:24–46, 2017.

[14] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[15] Peter J. Brockwell and Richard A. Davis. *Introduction to Time Series and Forecasting*. Springer Texts in Statistics. Springer International Publishing, 2016.

[16] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas C Müller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake Vanderplas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. *ArXiv e-prints*, 2013.

[17] David M Burns and Cari M Whyne. Seglearn: a python package for learning sequences and time series. *The Journal of Machine Learning Research*, 19(1):3238–3244, 2018.

[18] François Chollet et al. Keras. `https://keras.io`, 2015.

[19] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing*, 307:72–77, sep 2018.

[20] Vitaly Davydov and Franz J Király. xpandas: Python data containers for structured types and structured machine learning tasks. *openreview.net*, 2018.

[21] Jan G De Gooijer and Rob J Hyndman. 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473, 2006.

[22] Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 2013.

[23] Thomas G. Dietterich. Machine Learning for Sequential Data: A Review. In Caelli T., Amin A., Duin R.P.W., de Ridder D., and Kamel M., editors, *Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30. Springer, Berlin, Heidelberg, 2002.

[24] Peter Diggle, Patrick Heagerty, Kung-Yee Liang, and Scott Zeger. *Analysis of longitudinal data*. Oxford University Press, 2nd edition, 2013.

[25] H. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[26] Ben D Fulcher and Nick S Jones. hctsa: A Computational Framework for Automated Time-Series Phenotyping Using Massive Feature Extraction. *Cell systems*, 5(5):527–531.e3, nov 2017.

[27] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme. Learning time-series shapelets. In *Proc. 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[28] Alex Graves. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer, 2012.

[29] Ahmed Guecioueur. pysf: Supervised forecasting of sequential data in Python, 2018.

[30] Valery Guralnik and Jaideep Srivastava. Event detection from time series data. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 33–42. ACM, 1999.

[31] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software. *ACM SIGKDD Explorations Newsletter*, 11(1):10, nov 2009.

[32] G. Holmes, A. Donkin, and I.H. Witten. WEKA: a machine learning workbench. In *Proceedings of ANZIIS '94 - Australian New Zealnd Intelligent Information Systems Conference*, pages 357–361. IEEE, 1994.

[33] Stephan Hoyer, Clark Fitzgerald, Joe Hamman, et al. xarray: v0.8.0, August 2016.

[34] Stephan Hoyer and Joseph J. Hamman. xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, 5(1), apr 2017.

[35] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[36] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*, pages 1–10. IEEE, 2015.

[37] I. Karlsson, Papapetrou P, and H. Boström. Forests of randomized shapelet trees. In *International Symposium on Statistical Learning and Data Sciences*, pages 126–136. Springer, 2015.

[38] Viktor Kazakov and Franz J Király. Machine learning automation toolbox (mlaut). *arXiv preprint arXiv:1901.03678*, 2019.

[39] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. Segmenting time series: A survey and novel approach. In *Data mining in time series databases*, pages 1–21. World Scientific, 2004.

[40] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2), 2007.

[41] J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39(2):287–315, 2012.

[42] J. Lines and A. Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29:565–592, 2015.

[43] J. Lines, S. Taylor, and A. Bagnall. Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles. *ACM Trans. Knowledge Discovery from Data*, 12(5), 2018.

[44] B. Lucas, A. Shifaz, C. Pelletier, L. O'Neill, N. Zaidi, B. Goethals, F. Petitjean, and G. Webb. Proximity forest: an effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, 33(3):607–635, 2019.

[45] Helmut Lütkepohl. *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005.

[46] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. Statistical and machine learning forecasting methods: Concerns and ways forward. *PloS one*, 13(3):e0194889, 2018.

[47] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 2019.

[48] Wes McKinney. pandas: a Foundational Python Library for Data Analysis and Statistics. In *Python for High Performance and Scientific Computing*, 2011.

[49] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[50] Josef Perktold and Skipper Seabold. Statsmodels: Econometric and Statistical Modeling with Python Quantitative histology of aorta View project Statsmodels: Econometric and Statistical Modeling with Python. In *Proceedings of the 9th Python in Science Conference*, 2010.

[51] P. Schäfer. The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, 2015.

[52] P. Schäfer and M. Högqvist. SFA: a symbolic Fourier approximation and index for similarity search in high dimensional datasets. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 516–527, 2012.

[53] Romain Tavenard. tslearn: A machine learning toolkit dedicated to time-series data, 2017.

[54] R Taylor. Pyflux: An open source time series library for python, 2016.

[55] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[56] Stéfan van der Walt, S Chris Colbert, and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science & Engineering*, 13(2):22–30, mar 2011.

[57] Gaël Varoquaux, L. Buitinck, Gilles Louppe, Olivier Grisel, F. Pedregosa, and A. Mueller. Scikit-learn: Machine Learning Without Learning the Machinery. *GetMobile: Mobile Computing and Communications*, 19(1):29–33, jun 2015.

[58] Jeffrey M Wooldridge. *Econometric analysis of cross section and panel data*. MIT Press, 2010.