Neurocomputing 307 (2018) 72–77

Contents lists available at ScienceDirect

Neurocomputing

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

Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package)

Maximilian Christ[a], Nils Braun[b], Julius Neuffer[a], Andreas W. Kempa-Liehr[c,d,*]

[a] Blue Yonder GmbH, Karlsruhe, Germany
[b] Institute of Experimental Particle Physics, Karlsruhe Institute of Technology, Germany
[c] Department of Engineering Science, University of Auckland, New Zealand
[d] Freiburg Materials Research Center, University of Freiburg, Germany

ARTICLE INFO

Article history:
Received 31 May 2017
Revised 22 March 2018
Accepted 23 March 2018
Available online 4 May 2018

Communicated by Dr. Francesco Dinuzzo

Keywords:
Feature engineering
Time series
Feature extraction
Feature selection
Machine learning

ABSTRACT

Time series feature engineering is a time-consuming process because scientists and engineers have to consider the multifarious algorithms of signal processing and time series analysis for identifying and extracting meaningful features from time series. The Python package `tsfresh` (Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests) accelerates this process by combining 63 time series characterization methods, which by default compute a total of 794 time series features, with feature selection on basis automatically configured hypothesis tests. By identifying statistically significant time series characteristics in an early stage of the data science process, `tsfresh` closes feedback loops with domain experts and fosters the development of domain specific features early on. The package implements standard APIs of time series and machine learning libraries (e.g. `pandas` and `scikit-learn`) and is designed for both exploratory analyses as well as straightforward integration into operational data science applications.

1. Introduction

Trends such as the Internet of Things (IoT) [1], Industry 4.0 [2], and precision medicine [3] are driven by the availability of cheap sensors and advancing connectivity, which among others increases the availability of temporally annotated data. The resulting time series are the basis for machine learning applications like the classification of hard drives into risk classes concerning a specific defect [4], the analysis of the human heart beat [5], the optimization of production lines [6], the log analysis of server farms for detecting intruders [7], or the identification of patients with high infection risk [8]. Examples for regression tasks are the prediction of the remaining useful life of machinery [9] or the estimation of conditional event occurrence in complex event processing applications [10]. Other frequently occurring temporal data are event series from processes, which could be transformed to uniformly sampled time series via process evolution functions [11]. Time series feature extraction plays a major role during the early phases of data science projects in order to rapidly extract and explore different time series features and evaluate their statistical significance for predicting the target. The Python package `tsfresh` supports this process by providing automated time series feature extraction and selection on basis of the FRESH algorithm [12].

2. Problems and background

A time series is a sequence of observations taken sequentially in time [13]. In order to use a set of time series $\mathcal{D} = \{\chi_i\}_{i=1}^N$ as input for supervised or unsupervised machine learning algorithms, each time series χ_i needs to be mapped into a well-defined feature space with problem specific dimensionality M and feature vector $\vec{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,M})$. In principle, one might decide to map the time series of set $\mathcal{D}$ into a design matrix of N rows and M columns by choosing M data points from each time series χ_i as elements of feature vector $\vec{x}_i$. However, from the perspective of pattern recognition [14], it is much more efficient and effective to characterize the time series with respect to the distribution of data points, correlation properties, stationarity, entropy, and nonlinear time series analysis [15]. Therefore the feature vector $\vec{x}_i$ can be constructed by applying time series characterization methods f_j:

* Corresponding author at: Department of Engineering Science, University of Auckland, New Zealand.

E-mail addresses: max.christ@me.com (M. Christ), nils.braun@kit.edu (N. Braun), julius.neuffer@blue-yonder.com (J. Neuffer), a.kempa-liehr@auckland.ac.nz (A.W. Kempa-Liehr).

https://doi.org/10.1016/j.neucom.2018.03.067

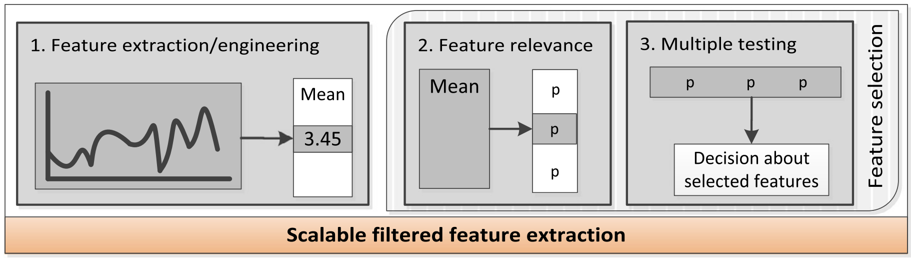


Fig. 1. The three steps of the `tsfresh` algorithm are feature extraction (1.), calculation of *p*-values (2.) and a multiple testing procedure (3.) [12]: Both steps 1. and 2. are highly parallelized in `tsfresh`, further 3. has a negligible runtime For 1, the public function extract_features is provided. 2. and 3. can be executed by the public function select_features. The function extract_relevant_features combines all three steps. By default the hypothesis tests of step 2 are configured automatically depending on the type of supervised machine learning problem (classification/regression) and the feature type (categorical/continuous) [12].

$\chi_i \rightarrow x_{i,j}$ to the respective time series χ_i, which results into feature vector $\vec{x}_i = (f_1(\chi_i), f_2(\chi_i), \ldots, f_M(\chi_i))$. This feature vector can be extended by additional univariate attributes $\{a_{i,1}, a_{i,2}, \ldots, a_{i,U}\}_{i=1}^{N}$ and feature vectors from other types of time series. If a machine learning problem had a total of K different time series and U univariate variables per sample i, the resulting design matrix would have i rows and $(K \cdot M + U)$ columns. Here, M denotes the number of features from time series characterization methods. The processed time series do not need to have the same number of data points.

For classification and regression tasks the significance of extracted features is of high importance, because too many irrelevant features will impair the ability of the algorithm to generalize beyond the train set and result in overfitting [12]. Therefore, `tsfresh` provides a highly parallel feature selection algorithm on basis of statistical hypothesis tests, which by default are configured automatically depending on the type of supervised machine learning problem (classification/regression) and the feature type (categorical/continuous) [12].

3. Software framework

3.1. Software architecture

By widely deploying pandas.DataFrames, e.g. as input and output objects, and providing `scikit-learn` compatible transformer classes, `tsfresh` implements the application programming interfaces of the most popular Python machine learning and data analysis frameworks such as `scikit-learn` [16], `numpy` [17], `pandas` [18], `scipy` [19], `keras` [20] or `tensorflow` [21]. This enables users to seamlessly integrate `tsfresh` into complex machine learning systems that rely on state-of-the-art Python data analysis packages.

The feature_extraction submodule (Fig. 1) contains both the collection of feature calculators and the logic to apply them efficiently to the time series data. The main public function of this submodule is extract_features. The number and parameters of all extracted features are controlled by a settings dictionary. It can be filled manually, instantiated using one of the predefined objects, or reconstructed from the column names of an existing feature matrix. The feature_selection submodule provides the function select_features, which implements the highly parallel feature selection algorithm [12]. Additionally, `tsfresh` contains several minor submodules: utilities provides helper functions used all over the package. convenience contains the extract_relevant_features function, which combines the extraction and selection with an additional imputing step in between. transformers enables the usage of `tsfresh` as part of `scikit-learn` [16] pipelines. The test suite covers 97% of the code base.

The feature selection and the calculation of features in `tsfresh` are parallelized and unnecessary calculations are prevented by calculating groups of similar features and sharing auxiliary results. For example, if multiple features return the coefficients of a fitted autoregressive model (AR), the AR model is only fitted once and shared. Local parallelization is realized on basis of the Python module `multiprocessing`, which is used both for feature selection and feature extraction. Distributed computing on a cluster is supported on basis of Dask [22].

The parallelization in the extraction and selection process of the features enables significant runtime improvement (Fig. 2). However, the memory temporarily used by the feature calculators quickly adds up in a parallel regime. Hence, the memory consumption of the parallelized calculations can be high, which can make the usage of a high number of processes on machines with low memory infeasible.

3.2. Software functionalities

`tsfresh` provides 63 time series characterization methods, which compute a total of 794 time series features. A design matrix of univariate attributes can be extended by time series features from one or more associated time series. Alternatively, a design matrix can be generated from a set of time series, which might have different number of data points and could comprise different types of time series. The resulting design matrix can be either used for unsupervised machine learning, or supervised machine learning, in which case statistically significant features can be selected with respect to the classification or regression problem at hand. For this purpose hypothesis tests are automatically configured depending on the type of supervised machine learning problem (classification/regression) and the feature type (categorical/continuous) [12].

Rows of the design matrix correspond to samples identified by their `id`, while its columns correspond to the extracted features. Column names uniquely identify the corresponding features with respect to the following three aspects. (1) the `kind` of time series the feature is based on, (2) the name of the feature calculator, which has been used to extract the feature, and (3) key-value pairs of parameters configuring the respective feature calculator:

[kind]_[calculator]_[parameterA]_[valueA]_[parameterB]_[valueB]

For supervised machine learning tasks, for which an instance of `sklearn` compatible transformer FeatureSelector had been fitted, the feature importance can be inspected, which is reported as (1 $-p$-value) with respect to the result of the automatically configured hypothesis tests.

4. Illustrative example and empirical results

For this example, we will inspect 15 force and torque sensors on a robot that allow the detection of failures in the robots

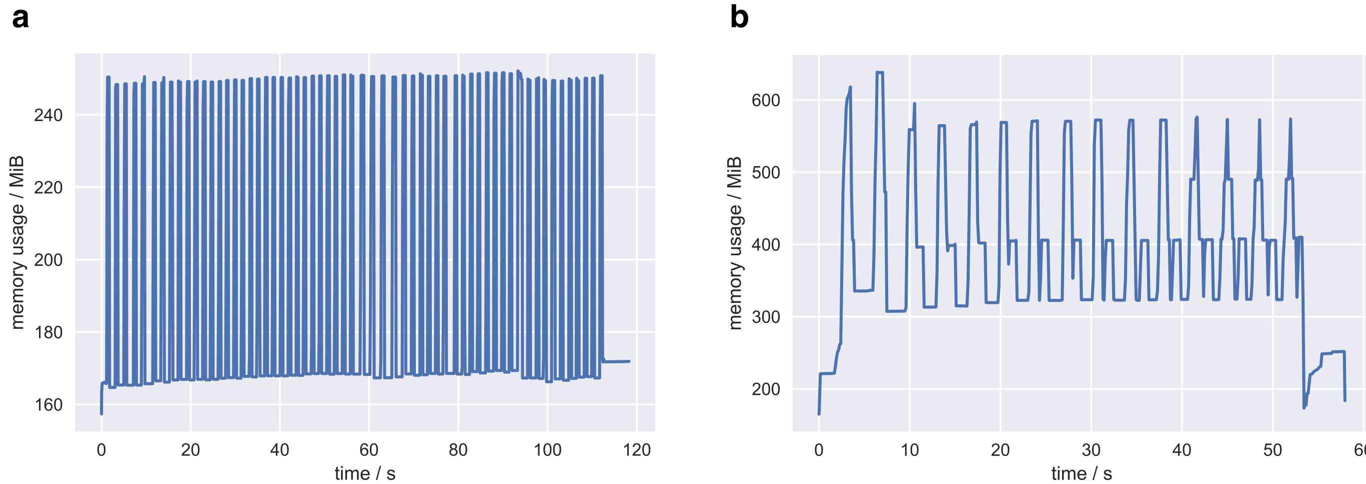


Fig. 2. Memory consumption of extraction and selecting time series features from 30 time series on MacBook Pro, 2.7 GHz Intel Core i5 and `tsfresh` v0.11.0 (Table 1). Each time series has a length of 1000 data points. (a) one core, (b) four cores (b).

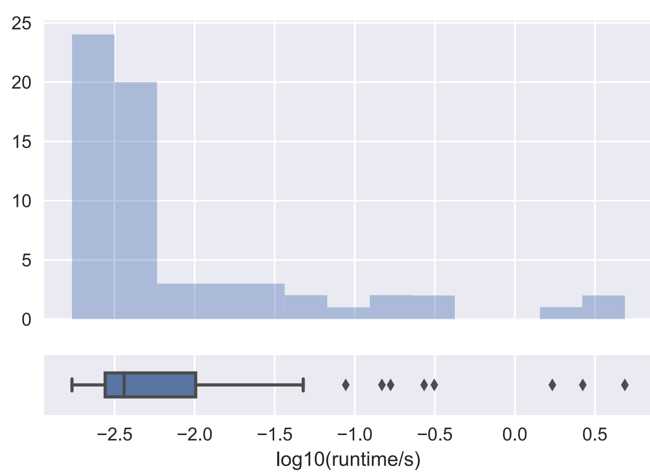


Fig. 3. Average runtime of feature calculation algorithms without parallelization for time series with 1000 data points on MacBook Pro, 2.7 GHz Intel Core i5 and `tsfresh`v0.11.0 (Table 1). The runtime varies between 1.7 ms and 4.8 s with a median of 3.6 ms.

execution [23,24]:

```
1  from tsfresh import select_features, extract_features
2  from tsfresh.examples.robot_execution_failures import
       load_robot_execution_failures
3  from tsfresh.utilities.dataframe_functions import impute
4
5  df, y = load_robot_execution_failures()
6  X = extract_features(df, column_id="id", column_sort="time")
7  impute(X)
8  X_filtered = select_features(X, y)
```

In line 5 we load the dataset. Then, in line 6, the features are extracted from the pandas.DataFrame df containing the time series. The resulting pandas.DataFrame X, the design matrix, comprises 3270 time series features. We have to replace non-finite values (e.g. NaN or inf) by imputing in line 7. Finally, the feature selection of `tsfresh` is used to filter out irrelevant features. The final design matrix X_filtered contains 623 time series features, which can now be used for training a classifier (e.g. a RandomForest from the `scikit-learn` package) to predict a robot execution failure based on the sensor time series. This example is extended in the Jupyter notebook `robot_failure_example.ipynb`.[1]

The runtime analysis given in the appendix (Fig. A.1) is summarized in Fig. 3. It shows the histogram and boxplot of the logarithm of the average runtimes for the feature calculators of `tsfresh` for a time series of 1000 data points. The average has been obtained from a sample of 30 different time series for which all features had been computed three times. The time series were simulated beforehand from the following sequence $x_{t+1} = x_t + \left[0.0045(y - 1/0.3)x_t - 325x_t^3\right] + 6.75 \cdot 10^{-5}\eta_t$ with $\eta_t \sim \mathcal{N}(0,1)$[25, p. 164] and y being the target. The figure shows that all but 9 of the time series feature extraction methods have an average runtime below 25 ms.

On a selection of 31 datasets from the UCR time series classification archive as well as an industrial dataset from the production of steel billets by a German steel manufacturer the FRESH algorithm was able to automatically extract relevant features for time series classification tasks [12]. The fresh algorithm is able to scale linearly with the number of used features, the number of devices/samples and the number of different time series [12]. Its scaling with respect to the length of the time series or number of features depends on the individual feature calculators. Some features such as the maximal value scale linearly with the length of the time series while other, more complex features have higher costs such as the calculation of the sample entropy (Fig. A.1). So, adjusting the calculated features can change the total runtime of `tsfresh` drastically.

[1] https://github.com/blue-yonder/tsfresh/blob/v0.11.0/notebooks/robot_failure_example.ipynb.

5. Conclusions

The Python based machine learning library `tsfresh` is a fast and standardized machine learning library for automatic time series feature extraction and selection. It is the only Python based machine learning library for this purpose. The only alternative is the Matlab based package `hctsa` [26], which extracts more than 7700 time series features. Because `tsfresh` implements the application programming interface of `scikit-learn`, it can be easily integrated into complex machine learning pipelines.

The widespread adoption of the `tsfresh` package shows that there is a pressing need to automatically extract features, originating from e.g. financial, biological or industrial applications. We expect that, due to the increasing availability of annotated temporally data, the interest in such tools will continue to grow.

Current code version

Acknowledgments

The authors would like to thank M. Feindt, who inspired the development of the FRESH algorithm, U. Kerzel and F. Kienle for their valuable feedback, Blue Yonder GmbH and its CTO J. Karstens, who approved the open sourcing of the `tsfresh` package, and the contributors to `tsfresh`: M. Frey, earthgecko, N. Haas, St. Müller, M. Gelb, B. Sang, V. Tang, D. Spathis, Ch. Holdgraf, H. Swaffield, D. Gomez, A. Loosley, F. Malkowski, Ch. Chow, E. Kruglick, T. Klerx, G. Koehler, M. Tomlein, F. Aspart, S. Shepelev, J. White, jkleint. The development of `tsfresh` was funded in part by the German Federal Ministry of Education and Research under Grant number 01IS14004 (project iPRODICT).

Appendix A. Detailed runtime of time series feature extraction

The average runtime has been obtained from a sample of 30 different time series for which all features had been computed three times. The time series were simulated beforehand from the following sequence:

$$x_{t+1} = x_t + \left[0.0045(y - 1/0.3)x_t - 325x_t^3\right] + 6.75 \cdot 10^{-5}\eta_t \quad \text{(A.1)}$$

with $\eta_t \sim \mathcal{N}(0, 1)$ [25, p. 164] and y being the target.

Table 1
Code metadata of `tsfresh`.

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v0.11.0
C2	Permanent link to code/repository used of this code version	https://github.com/blue-yonder/tsfresh/releases/tag/v0.11.0
C3	Legal Code License	MIT
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python
C6	Compilation requirements, operating environments & dependencies	OS-X, Unix-Like (e.g. Linux), Microsoft Windows A Python interpreter (2.7 or 3.6.3) and the following Python packages: requests (2.9.1), numpy (1.10.4), pandas (0.20.3), scipy (0.17.0), statsmodels (0.6.1), patsy (0.4.1), scikit-learn (0.17.1), future (0.16.0), six (1.10.0), tqdm (4.10.0), ipadress (1.0), dask (0.15.2), distributed (1.18.3)
C7	If available Link to developer documentation/manual	http://tsfresh.readthedocs.io
C8	Support email for questions	max.christ@me.com

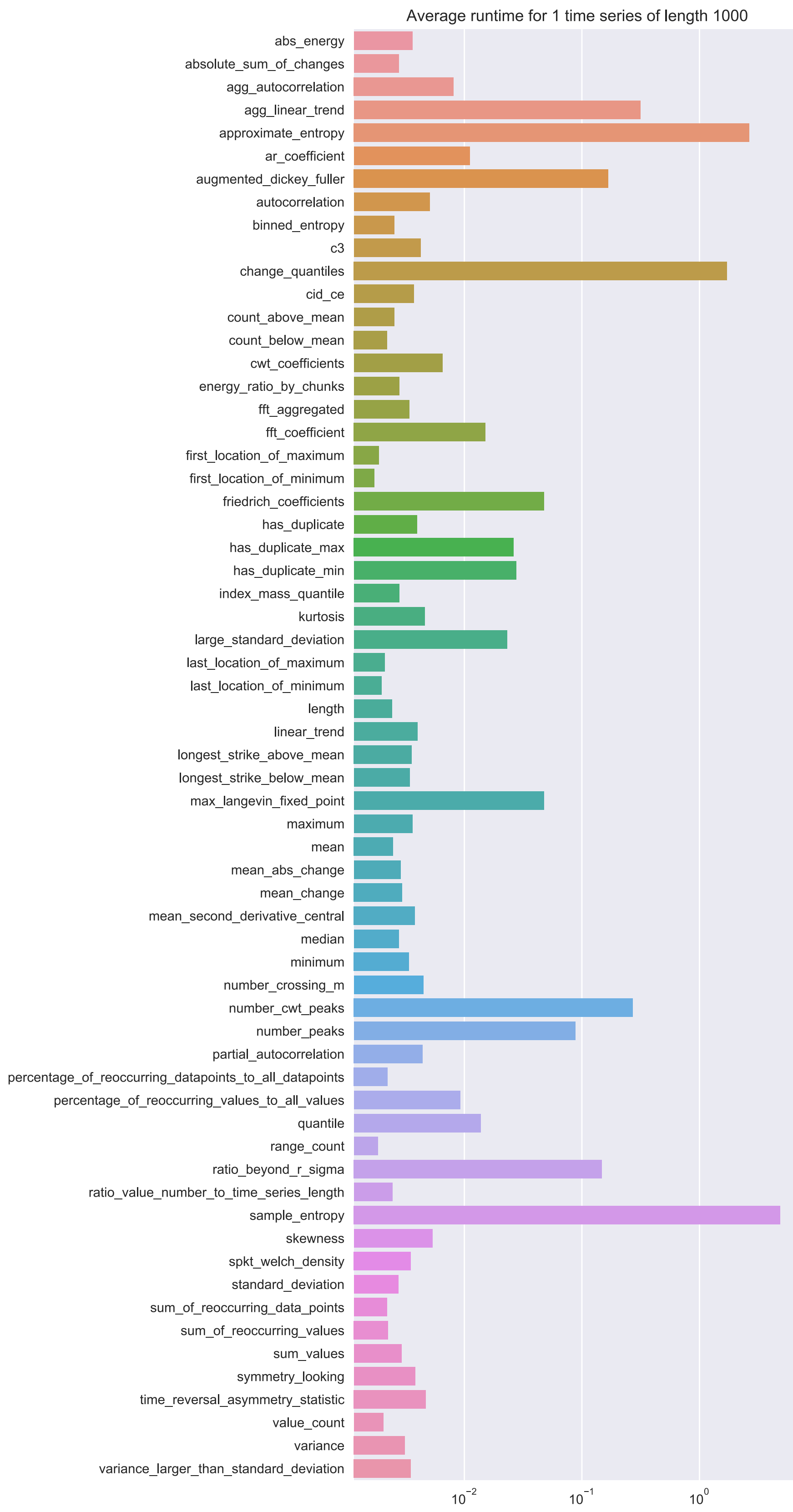


Fig. A.1. Average runtime of time series feature extraction methods documented in `http://tsfresh.readthedocs.io/en/latest/text/list_of_features.html`.

References

[1] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of Things (IoT): a vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660.

[2] M. Hermann, T. Pentek, B. Otto, Design principles for Industrie 4.0 scenarios, in: Proceedings of 2016 49th Hawaii International Conference on System Sciences (HICSS), 2016, p. 3928.

[3] F.S. Collins, H. Varmus, A new initiative on precision medicine, N. Engl. J. Med. 372 (9) (2015) 793–795, doi:10.1056/NEJMp1500523.

[4] R.K. Mobley, An introduction to predictive maintenance, second, Elsevier Inc., Woburn, MA, 2002.

[5] B.D. Fulcher, M.A. Little, N.S. Jones, Highly comparative time-series analysis: the empirical structure of time series and their methods, J. R. Soc. Interface 10 (83) (2013) 20130048.

[6] M. Christ, F. Kienle, A.W. Kempa-Liehr, Time series analysis in industrial applications, in: Proceedings of Workshop on Extreme Value and Time Series Analysis, KIT Karlsruhe, 2016, doi:10.13140/RG.2.1.3130.7922.

[7] A.L. Buczak, E. Guven, A survey of data mining and machine learning methods for cyber security intrusion detection, IEEE Commun. Surv. Tutor. 18 (2) (2016) 1153–1176.

[8] J. Wiens, E. Horvitz, J.V. Guttag, Patient risk stratification for hospital-associated c. diff as a time-series classification task, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 467–475.

[9] J. Yan, Machinery Prognostics and Prognosis Oriented Maintenance Management, John Wiley & Sons, Singapore, 2015.

[10] M. Christ, J. Krumeich, A.W. Kempa-Liehr, Integrating predictive analytics into complex event processing by using conditional density estimations, in: Proceedings of IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW), IEEE Computer Society, Los Alamitos, CA, USA, 2016, pp. 1–8, doi:10.1109/EDOCW.2016.7584363.

[11] A. Kempa-Liehr, Performance analysis of concurrent workflows, J. Big Data 2 (10) (2015) 1–14, doi:10.1186/s40537-015-0017-0.

[12] M. Christ, A.W. Kempa-Liehr, M. Feindt, Distributed and parallel time series feature extraction for industrial big data applications. Asian Machine Learning Conference (ACML) 2016, Workshop on Learning on Big Data (WLBD), Hamilton (New Zealand), ArXiv preprint arXiv: 1610.07717v1.

[13] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung, Time Series Analysis: Forecasting and Control, fifth ed., John Wiley & Sons, Hoboken, New Jersey, 2016.

[14] C.M. Bishop, Pattern Recognition and Machine Learning, Information Science and Statistics, Springer, New York, 2006.

[15] B.D. Fulcher, Feature-Based Time-Series Analysis, Cornell University Library, 2017. arXiv: 1709.08055v2.

[16] 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, E. Duchesnay, Scikit-learn: machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[17] S.V.D. Walt, S.C. Colbert, G. Varoquaux, The numpy array: a structure for efficient numerical computation, Comput. Sci. Eng. 13 (2) (2011) 22–30.

[18] W. McKinney, Data structures for statistical computing in Python, in: S. van der Walt, J. Millman (Eds.), Proceedings of the 9th Python in Science Conference, 2010, pp. 51–56.

[19] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: open source scientific tools for Python, 2001. http://www.scipy.org/.

[20] F. Chollet, Keras, 2015. https://github.com/fchollet/keras.

[21] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, in: Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org

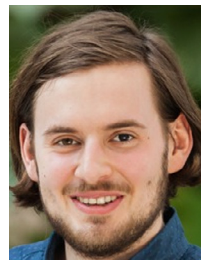
[22] M. Rocklin, Dask: Parallel computation with blocked algorithms and task scheduling, in: K. Huff, J. Bergstra (Eds.), Proceedings of the 14th Python in Science Conference, 2015, pp. 130–136.

[23] L.S. Lopes, Robot learning at the task level: a study in the assembly domain, Ph.D. thesis, Universidade Nova de Lisboa, Portugal, 1997.

[24] M. Lichman, UCI machine learning repository, 2013. http://archive.ics.uci.edu/ml.

[25] A.W. Liehr, Dissipative solitons in reaction diffusion systems, in: Mechanisms, Dynamics, Interaction, 70, Springer Series in Synergetics, Berlin, 2013. 10.1007/978-3-642-31251-9.

[26] B.D. Fulcher, N.S. Jones, hctsa: a computational framework for automated time-series phenotyping using massive feature extraction, Cell Syst. 5 (5) (2017) 527–531. E3. doi: 10.1016/j.cels.2017.10.001.



Maximilian Christ received his M.S. degree in Mathematics and Statistics from Heinrich-Heine-Universität Düsseldorf, Germany, in 2014. In his daily work as a Data Science Consultant at Blue Yonder GmbH he optimizes business processes through data driven decisions. Beside his business work he is persuing a Ph.D. in collaboration with University of Kaiserslautern, Germany. His research interest relate on how to deliver business value through Machine Learning based optimizations.

Nils Braun is a Ph.D. student in High Energy Particle Physics at Karlsruher Institut of Technology (KIT), Germany. His research is focussed on developing and optimizing scientific software for analysing and processing large amounts of recorded data efficiently. He received his M.S. degree in Physics at KIT in 2016. He has worked at Blue Yonder GmbH as a Data Science Engineer, where he developed platforms and utilities for various data science tasks.

Julius Neuffer is a Software Developer at Blue Yonder GmbH. He has a background in computer science and philosophy. Aside from software development, he takes interest in applying machine learning to real-worldproblems.

Andreas W. Kempa-Liehr is a Senior Lecturer at the Department of Engineering Science of the University of Auckland, New Zealand, and an Associate Member of the Freiburg Materials Research Center (FMF) at the University of Freiburg, Germany. Andreas received his doctorate from the University of Münster in 2004 and continued his research as head of service group Scientific Information Processing at FMF. From 2009 to 2016 he was working in different data science roles at EnBW Energie Baden-Württemberg AG and Blue Yonder GmbH.