Tslearn, A Machine Learning Toolkit for Time Series Data

Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Russwurm, Kushal Kolar, et al.

To cite this version:

HAL Id: hal-02883390
https://inria.hal.science/hal-02883390
Submitted on 29 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Tslearn, A Machine Learning Toolkit for Time Series Data

Romain Tavenard
Université de Rennes, CNRS, LETG-Rennes, IRISA-Obelix, Rennes, France

Johann Faouzi
Aramis Lab, INRIA Paris, Paris Brain Institute, Paris, France

Gilles Vandewiele
IDLab, Ghent University – imec, Ghent, Belgium

Felix Divo
Technische Universität Darmstadt, Darmstadt, Germany

Guillaume Androz
Icentia Inc., Québec, Canada

Chester Holtz
Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA, USA

Marie Payne
McGill University, Montreal, Québec, Canada

Roman Yurchak
Symerio, Paris, France

Marc Rußwurm
Technical University of Munich, Chair of Remote Sensing Technology, Munich, Germany

Kushal Kolar
Sars International Centre for Marine Molecular Biology, University of Bergen, Norway

Eli Woods
Eaze Technologies, Inc., San Francisco, CA, USA

Editor: Alexandre Gramfort

Abstract

tsklearn is a general-purpose Python machine learning library for time series that offers tools for pre-processing and feature extraction as well as dedicated models for clustering, classification and regression. It follows scikit-learn’s Application Programming Interface for transformers and estimators, allowing the use of standard pipelines and model selection tools on top of tsklearn objects. It is distributed under the BSD-2-Clause license, and its source code is available at https://github.com/tslearn-team/tslearn.

Keywords: time series, clustering, classification, pre-processing, data mining

1. Introduction

Temporal data are ubiquitous in many application domains, such as medicine, robotics, or videos. Dealing with such data requires dedicated methods that take into account the
high correlation between consecutive samples in a time series. Moreover, in many cases, one would like a time series approach to encode invariance to small time shifts, which once again implies using specific methodologies. tslearn is a Python package that provides tools to fit relevant models to time series data. In the following, time series data is understood as series of features collected over time. It includes pre-processing routines, feature extractors, and machine learning models for classification (Bagnall et al., 2017; Fawaz et al., 2019), regression and clustering (Aghabozorgi et al., 2015). Both univariate and multivariate time series can be handled in tslearn. Further, data sets can contain time series of variable-length, as discussed below. tslearn follows scikit-learn’s API for transformers and estimators, allowing the use of scikit-learn’s pipelines and model selection tools on tslearn objects. We use continuous integration tools to test each contribution to the library and target a coverage of at least 90% to detect bugs as early as possible. Online documentation is available at https://tslearn.readthedocs.io/ and semantic versioning is used (this document describes version 0.3.1). tslearn is distributed under the BSD-2-Clause license.

In the following, we present basic usage of the library, an overview of the implemented algorithms and a comparison to other related software.

2. Implementation Description

tslearn v0.3.1 is a cross-platform software package for Python 3.5+. It depends on numpy (Van Der Walt et al., 2011) & scipy (Virtanen et al., 2020) packages for basic array manipulations and standard linear algebra routines and on scikit-learn (Pedregosa et al., 2011) for its API and utilities. It also utilizes Cython (Behnel et al., 2011), numba (Lam et al., 2015) and joblib (Varoquaux et al., 2010) for efficient computation. Finally, keras (Chollet et al., 2015) with tensorflow (Abadi et al., 2016) backend is an optional dependency that is necessary to use the shapelets module in tslearn that provides an efficient implementation of the shapelet model by Grabocka et al. (2014).

In tslearn, a time series data set can be represented through a three-dimensional numpy array of shape \((n, T, d)\) where \(n\) is the number of time series in the set, \(T\) their length, and \(d\) their dimensionality. If time series from the set are not equal-sized, NaN values are appended to the shorter ones and \(T\) is hence the maximum of sizes for time series in the set:

```python
from tslearn.utils import to_time_series_dataset
my_first_time_series = [1, 3, 4, 2]
my_second_time_series = [1, 2, 4, 2, 1]
formatted_dataset = to_time_series_dataset([my_first_time_series,
                                          my_second_time_series])
# formatted_dataset.shape is then (2, 5, 1)
```

Then, data can be fed to transformers that are available in the preprocessing and piecewise modules and/or estimators that follow the scikit-learn API\(^1\). Note that scikit-learn’s pipelines and model-selection tools can be used in conjunction with tslearn transformers and estimators, as shown in the code snippet below:

---

\(^1\) One noticeable difference when compared with scikit-learn estimators and transformers, though, is that tslearn ones expect 3-dimensional arrays which can contain NaN values, as this is the basic format for tslearn data sets.
from sklearn.model_selection import KFold, GridSearchCV
from tslearn.neighbors import KNeighborsTimeSeriesClassifier

knn = KNeighborsTimeSeriesClassifier(metric="dtw")
p_grid = {"n_neighbors": [1, 5]}
cv = KFold(n_splits=2, shuffle=True, random_state=0)
clf = GridSearchCV(estimator=knn, param_grid=p_grid, cv=cv)
clf.fit(X, y)

3. Contents

Our package first offers basic utilities to format data sets for use with tslearn transformers and estimators, and to cast time series data sets from and to other Python time series toolkit formats. It also provides standard pre-processing techniques and feature extraction methods, implemented as scikit-learn-compatible transformers.

Many of our learning algorithms rely on the use of time series specific metrics that are themselves made available in a dedicated metrics module.

In terms of machine learning methods, we provide implementations of clustering algorithms (some of which rely on barycenter computation routines that are also part of our public API) specifically tailored for temporal data. Supervised learning methods (for regression and classification) are also provided. All these are implemented as scikit-learn-compatible estimators.

The importance of providing time-series specific methods for machine learning is illustrated in the example below and the corresponding Figure 1, where standard Euclidean $k$-means fails while DTW-based ones (Sakoe and Chiba, 1978; Petitjean et al., 2011; Cuturi and Blondel, 2017) can distinguish between different time series profiles:

from tslearn.clustering import TimeSeriesKMeans
from tslearn.datasets import CachedDatasets

# Load the 'Trace' data set
X_train = CachedDatasets().load_dataset('Trace')[0]

# Define parameters for each metric
euclidean_params = {"metric": "euclidean"}
dba_params = {"metric": "dtw"}
sdtw_params = {"metric": "softdtw", "metric_params": {"gamma": 0.01}}

# Perform clustering for each metric
y_preds = []
for params in (euclidean_params, dba_params, sdtw_params):
    km = TimeSeriesKMeans(n_clusters=3, random_state=0, **params)
    y_preds.append(km.fit_predict(X_train))

4. Comparison to Related Software

tslearn is not the only Python package to focus on machine learning for time series. Some packages, like cesium-ml and seglearn (Burns and Whyne, 2018) focus on pre-

2. The cesium-ml package can be found here: http://cesium-ml.org
Figure 1: $k$-means clustering ($k = 3$) using different base metrics. Each graph represents a cluster (i.e. a different $y_{\text{preds}}$ value), with its centroid plotted in bold red.

processing time series data to feed scikit-learn models. Similarly, tsfresh (Christ et al., 2018) specializes in feature extraction from time series. pyts (Faouzi and Janati, 2020) and sktime (Löning et al., 2019), on the other hand, focus on supervised learning. Other packages focus on some families of methods, such as statsmodel (Seabold and Perktold, 2010) that provides standard statistical models for time series analysis, hmmlearn\(^3\) that focuses on Hidden Markov Models, stumpy (Law, 2019) that relies on the matrix profile data structure (Yeh et al., 2016), or pyflux (Taylor, 2016), which offers a large panel of probabilistic models aimed at forecasting.

Compared to these packages, tslearn is a general-purpose machine learning library for time series that offers pre-processing and feature extraction tools as well as dedicated models for clustering, classification and regression. Note however that, since the packages listed above are more focused than tslearn, they tend to incorporate a wider range of algorithms for their task of interest than our library. This is why we have included utilities to cast data sets between tslearn format and the ones used by these libraries, in order to help facilitate interoperability.

5. Conclusion

tslearn is a general-purpose Python machine learning library for time series. It implements several standard estimators for time series for problems such as clustering, classification and regression. It is under active development with aim at the integration of additional methods.

\(^3\) The hmmlearn package can be found here: https://github.com/hmmlearn/hmmlearn.
Specific attention will be paid to keep tslearn’s genericity rather than focus, for example, on supervised settings for which other tools exist.

Acknowledgments

Romain Tavenard would like to acknowledge support for this project from the Agence Nationale de la Recherche (ANR grant ANR-18-CE23-0006).

References


