Time Series Classification in Python

Johann Faouzi

Postdoctoral researcher

Aramis project-team, Paris Brain Institute, Inria, Sorbonne Université, CNRS, Inserm

Signal, Statistics and Learning Seminar

March 17, 2022

Outline

1. Time series classification
   - Metric-based approaches
   - Feature-based approaches
2. Managing your project as a software
3. `pyts`: A Python Package for Time Series Classification

Outline

Machine learning for time series

- Time series data is **unstructured** $\longrightarrow$ not suited as raw input to standard machine learning classifiers (e.g., logistic regression).
- Two main approaches: **feature-based** and **metric-based** approaches.
- Feature-based methods:
  - **Independent process**: Running the feature extraction process before fitting the classifier on the extracted features.
  - **Incorporated process**: Including the feature extraction process in the classifier (e.g., neural networks with several layers).
- Metric-based methods: **Adapting** existing machine learning classifiers to time series data (e.g., with specific metrics for nearest-neighbor methods and specific kernels for kernel methods).

Literature overview

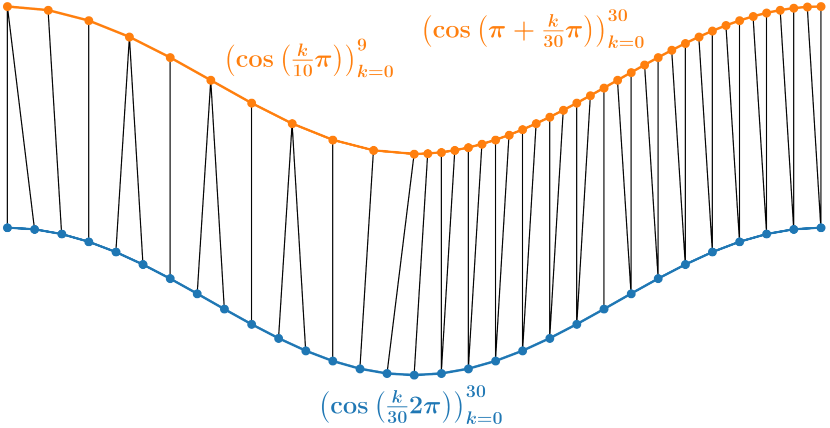
- Not an exhaustive literature review.
- Highlight the **main algorithms** and the **variety of methods**.
- Time series are assumed to be **univariate** (a real number at each timestamp) and **not multivariate** (a real-valued vector at each timestamps, e.g. (latitude, longitude) pairs for GPS coordinates).

Outline

Limitations of the Euclidean distance

- Simple example from speech recognition:
  - Two audio recordings of the **same person** pronouncing the **same sentence** but at **different speech rates**.
  - Expectations: a relevant metric should return a **low value** (i.e., both time series are similar).
- Two time series $X = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and $Y = (y_1, \ldots, y_m) \in \mathbb{R}^m$
- Limitations of the Euclidean distance for time series: $\left(\sum_i (x_i - y_i)^2 \right)^{1/2}$
  - Independent comparison (squared difference) in each dimension
  - Not defined for two vectors of different sizes

Global alignment



Dynamic time warping

- **Local divergence**: function that measures closeness between two values, e.g.:

$$\forall x, y \in \mathbb{R}, f(x, y) = (x - y)^2$$

- **Cost matrix**: evaluation of the local divergence for every pair (x_i, y_j)

$$\forall i, j \in \{1, \dots, n\} \times \{1, \dots, m\}, C_{ij} = f(x_i, y_j)$$

- **Warping path**: sequence $p = (p_1, \dots, p_L)$ such that:
  - value condition: $\forall l \in \{1, \dots, L\}, p_l = (i_l, j_l) \in \{1, \dots, n\} \times \{1, \dots, m\}$
  - boundary condition: $p_1 = (1, 1)$ and $p_L = (n, m)$
  - step condition: $\forall l \in \{1, \dots, L-1\}, p_{l+1} - p_l \in \{(0,1), (1,0), (1,1)\}$

Dynamic time warping

- **Cost associated with a warping path**:

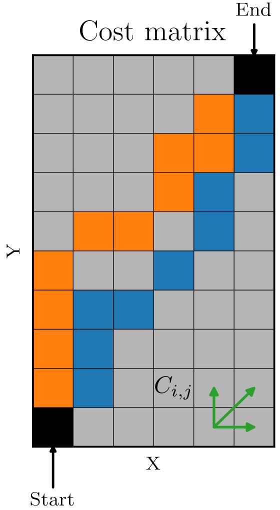
$$C_p(X, Y) = \sum_{l=1}^{L} C_{i_l, j_l}$$

- **Dynamic time warping** [SC78]: minimum cost among all the possible warping paths:

$$\text{DTW}(X, Y) = \min_{p \in \mathcal{P}} C_p(X, Y)$$

- Computed using **dynamic programming**:

$$\text{DTW}(X_{:i}, Y_{:j}) = C_{i,j} + \min\{\text{DTW}(X_{:i-1}, Y_{:j-1}), \text{DTW}(X_{:i-1}, Y_{:j}), \text{DTW}(X_{:i}, Y_{:j-1})\}$$



Limitations of dynamic time warping

- **High complexity**: $\mathcal{O}(nm)$ for two time series of sizes n and m.
- (Possibly too) large time warps.
- **Not a distance**: separation property and **triangle inequality** not satisfied) $\longrightarrow$ no efficient nearest-neighbor search algorithm.

Constraint regions

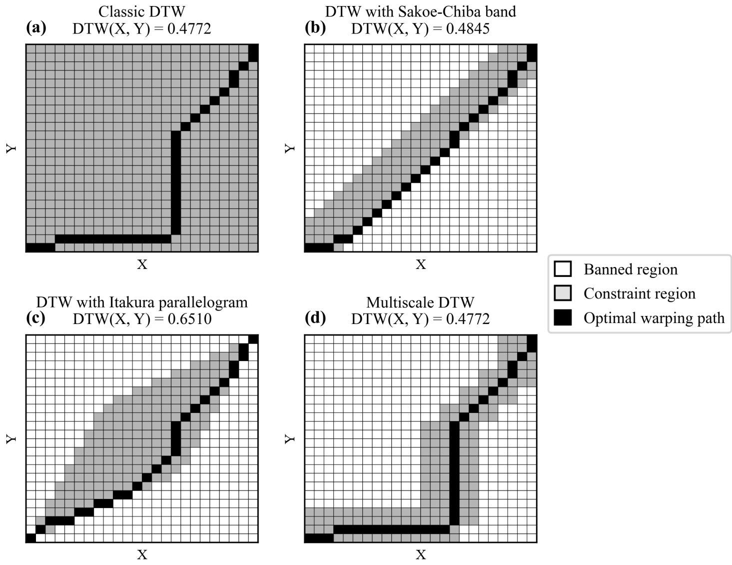
- **Idea**: Limit the possible values in a warping path.

-

Pros	Cons
Decrease maximum time warp	May not retrieve the optimal path
Decrease computational complexity	Hyperparameter

- A constraint region may depend on the values of both time series.
  - Series-independent constraint regions: Sakoe-Chiba band [SC78], Itakura parallelogram [Ita75].
  - Series-dependent constraint regions: Multiscale-DTW [MMK06], FastDTW [SC07].

Dynamic time warping (with constraint regions)



Global alignment kernel

- Dynamic time warping cannot be used to define a positive definite kernel since it does not satisfy the triangle inequality.

- **Global alignment kernel** [Cut11]:

$$k_{\mathsf{GA}}^{\gamma}(x, y) = \sum_{p \in \mathcal{P}} \exp\left(-C_p(x, y)/\gamma\right)$$

- k_{GA}^{γ} is a positive definite kernel under mild conditions.

- Soft dynamic time warping [CB17] (differentiable loss function):

$$\text{soft-dtw}_{\gamma} = -\gamma \log k_{\mathsf{GA}}^{\gamma}$$

Outline

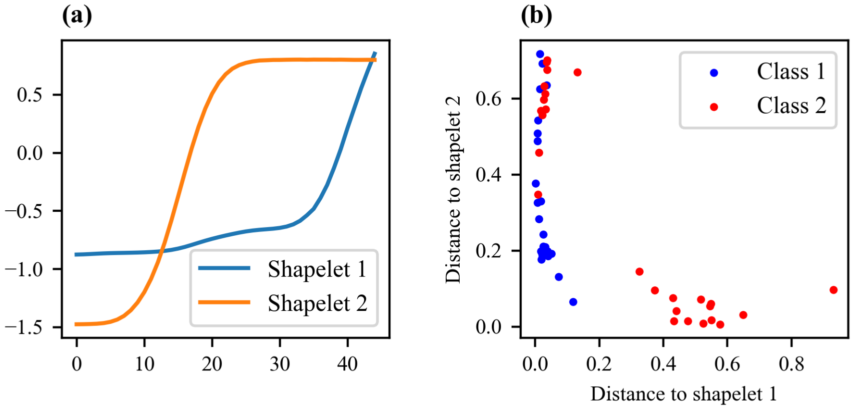
Shapelet-based algorithms

- **Idea**: Some small sequences of consecutive values may be specific to certain classes.
- **Shapelet**: real-valued vector of size $l \leq n$ (n being the size of the time series).
- **"Distance"** between a time series $X = (x_1, \ldots, x_n)$ and a shapelet $S = (s_1, \ldots, s_l)$:

$$d(X, S) = \min_{j \in \{0, \ldots, n-l\}} \sum_{i=1}^{l} (x_{i+j} - s_i)^2$$

- **Algorithms**: Shapelet transform [Lin+12], Learning shapelets [Gra+14].

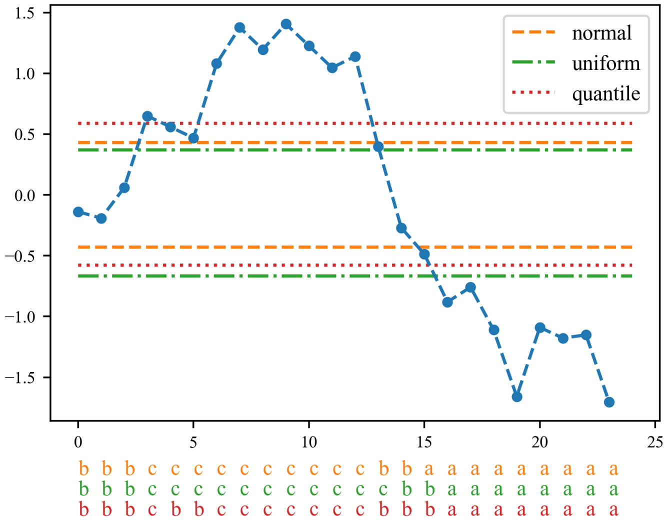
Learning shapelets



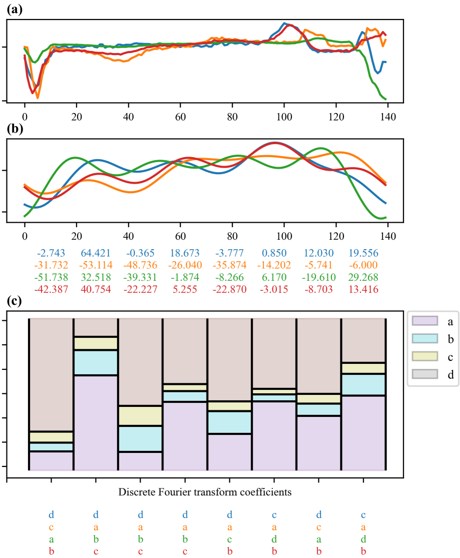
Dictionary-based approaches

- **Idea**: transform a time series into a bag of words.
- **General algorithm**:
  1. Extract subsequences using a sliding window.
  2. Transform each subsequence into a word.
  3. Perform classification based on the word frequencies.
- **Algorithms**: Bag-of-Patterns [LKL12], SAXVSM [SM13], BOSS [Sch15], BOSSVS [Sch16], WEASEL [SL17]...
- **Two main methods** to transform a subsequence into a word:
  - discretization of (standardized) values: SAX [Lin+07]
  - discretization of Fourier coefficient: SFA [SH12]

Symbolic Aggregate approXimation (SAX)



Symbolic Fourier Approximation (SFA)



Imaging time series

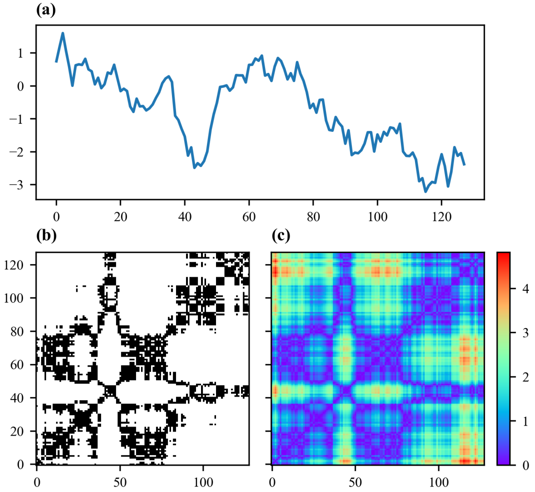
- Old concept (for visualizing dynamic systems).
- Motivated by **breakthroughs in computer vision** (convolutional neural networks).
- **Algorithms**: Recurrence plot [EKR87], Gramian angular field [WO15], Markov transition field [WO15].

Imaging time series: recurrence plots

$$\vec{x}_i = (x_i, x_{i+\tau}, \ldots, x_{i+(m-1)\tau})$$

$$R_{ij} = \mathbb{1}\left(\|\vec{x}_i - \vec{x}_j\|_2 < \varepsilon\right)$$

$$R_{ij} = \|\vec{x}_i - \vec{x}_j\|_2$$



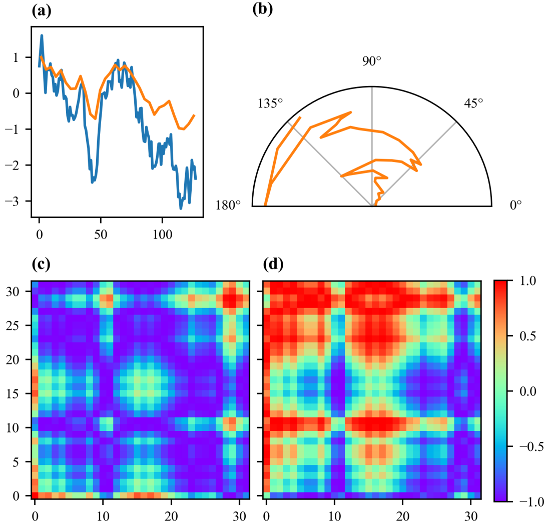
Imaging time series: Gramian angular fields

$$\tilde{x}_i = -1 + 2\frac{x_i - \min(x)}{\max(x) - \min(x)}$$

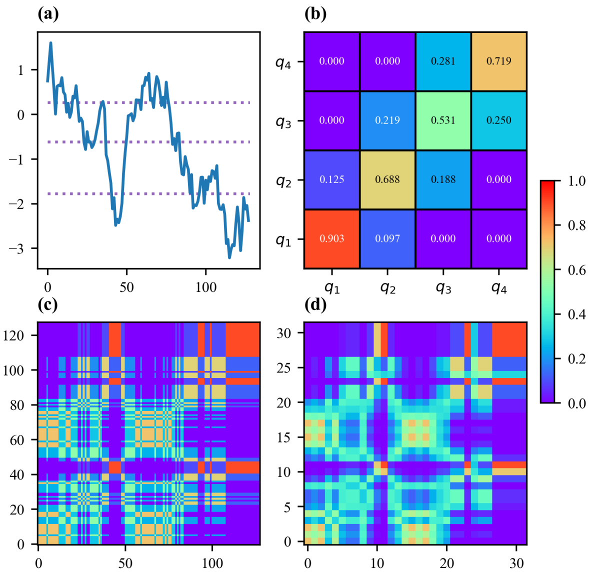
$$\phi_i = \arccos(\tilde{x}_i)$$

$$\text{GASF}_{i,j} = \cos(\phi_i + \phi_j)$$

$$\text{GADF}_{i,j} = \sin(\phi_i - \phi_j)$$



Imaging time series: Markov transition fields



Tree-based algorithms

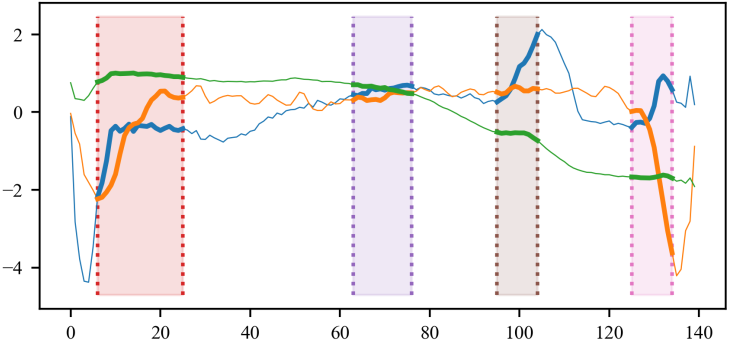
- Motivated by the **success** of the **random forest** and **extremely randomized trees** algorithms.

- Two main approaches:
    - **Extract features** that are then used to **fit a standard tree-based algorithm**.
    - **Modify the tree building process** to make use of the different **metrics for time series** published in the literature.

- **Algorithms**: Time series forest [Den+13], time series bag-of-features [BRT13], Proximity forest [Luc+19].

Tree-based algorithms: Time series forest

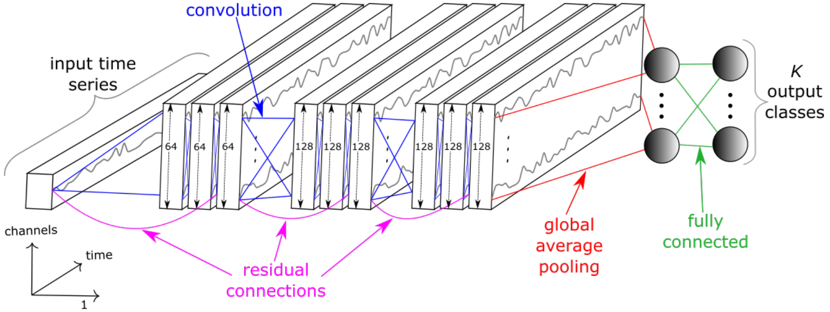
(a)



(b)

	Interval 1			Interval 2			Interval 3			Interval 4		
	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope
Time series 1	-0.61	0.504	0.052	0.58	0.086	0.02	1.004	0.572	0.197	0.189	0.504	0.156
Time series 2	-0.467	0.977	0.158	0.403	0.084	0.017	0.568	0.061	0.011	-1.204	1.305	-0.431
Time series 3	0.944	0.061	0.002	0.604	0.075	-0.018	-0.57	0.065	-0.017	-1.671	0.024	0.003

Neural networks: InceptionTime [Ism+20]



Random convolutional kernels

- Generating **random** convolutional kernels instead of learning them.
- **Different aggregated features** computed from each feature map from usual global average/max pooling:
  - proportion of positive values
  - longest period of consecutive positive values
- Ridge classifier fitted on these extracted features.
- **Algorithms**: ROCKET [DPW20], MiniROCKET [DSW21], MultiROCKET [Tan+21].

Ensemble models

- Ensemble of **several models** (different algorithms, same algorithms with different hyperparameters).
- **State-of-the-art** in terms of **predictive performance** only, but **very high algorithmic complexity**.
- **Algorithms**: COTE [Bag+15], HIVE-COTE [LTB18; Bag+20; Mid+21], TS-CHIEF [Shi+20].

Time Series Classification Archive

- **Website**: `http://timeseriesclassification.com`
- Over 100 univariate (and 30 multivariate) time series classification datasets.
- **Benchmark results** for many algorithms.

Conclusion

- **Many papers** describing **new algorithms** dedicated to time series classification have been published in the literature, with a **wide variety** of approaches being investigated.

- **Concrete application**:
  - One wants to tackle a real-world use case which is formulated as a time series classification task.
  - What are their possibilities?

Outline

Barriers to work on a real-world application

- Investigate **several algorithms** to see what works best.
- Possible issues with source code:
  - **Not available**.
  - Written in **different programming languages** (Java, MATLAB, Python, R, etc.).
  - Provided commands only aiming at **reproducing the results on some given datasets**.
  - **Barely commented** and **not easily extendable**.
  - **Barely documented**.

Replication crisis

- Little incentive to publish the source code associated to a paper (until recently).
- Source code rarely peer reviewed (until recently).
- **Yet, all the experiments, thus the results and conclusions, rely on the source code.**

Source code - different levels of usability

- **Code availability**: Easily accessing the source code of a project.
- **Reproducibility**: Reproducing (almost) the same experiments and obtaining (almost) the same results (hardware, float precision, etc.).
- **Replicability**: Slightly modifying the experiments (different dataset, different use case) and obtaining "good" results.
- **Reusability**: Easily integrating the tools made available in one project in another project.

Objective

- **Present the notions and tools that make producing reusable code easier.**
- **Advocate for managing your project as a software.**

Version control

- **Problem**: Updating the source code of a software may quickly become a mess because of multiple versions of the same software at any given time:
  - Remote version
  - Local version for each developer
- **Version control**: Tracking and providing control over changes to source code.
- **Distributed version control**: The complete codebase, including its full history, is mirrored on every developer's computer, enabling automatic management branching and merging.
- **Tools**:
  - git
  - Mercurial

Hosting your source code

- GitHub
- GitLab
- Bitbucket
- SourceForge

Hosting your (Python) package

- Some programming languages (e.g., Python, R, TeX) have an **official archive** to upload and download packages.

- PyPI: Python Package Index
  - Over 330 thousand projects
  - Over 3 million releases
  - Over 500k users

```
pip install pyts
```

```
conda install -c conda-forge pyts
```

- conda: package, dependency and environment management:
  - **Limitation**: Only a few packages are available in the default channel; anyone can create their own channel to host their packages (but this has several disadvantages).
  - conda-forge is a community effort that provides conda packages for a wide range of software in a single channel.

Semantic versioning

- Website: https://semver.org
- Summary:

> *Given a **version number MAJOR.MINOR.PATCH**, increment the:*
>
> - *MAJOR version when you make **incompatible API changes**,*
> - *MINOR version when you add **functionality in a backwards compatible manner**, and*
> - *PATCH version when you make **backwards compatible bug fixes**.*
>
> *Additional labels for pre-release and build metadata are available as extensions to the MAJOR.MINOR.PATCH format.*

Linting

- **Definition**: Process of checking the source code for programmatic and stylistic errors.

- Examples of stylistic errors:
  - Lines too long
  - Defining variables that are never used
  - Missing (or too many) whitespaces (or blank lines)

Linting in Python

- Mainly defined by two **Python Enhancement Proposals** (PEP):
  - PEP 8: Style Guide for Python Code
  - PEP 257: Docstring Conventions

- Main Python package: `flake8`
  - `flake8` itself does not implement checks but builds a strong foundation for a plugin ecosystem.
  - Popular plugins:
    - `pyflakes`: checks Python code for errors.
    - `pycodestyle`: checks Python code against some PEP 8 style conventions.
    - `mccabe`: checks McCabe complexity using Ned's script.
    - `pep8-naming`: checks Python code against PEP 8 naming conventions.
    - `flake8-docstrings`: is an extension for `pydocstyle` to `flake8`.

Code style (in Python)

- Even when abiding by PEP 8 style conventions, there are still **many ways to write the same piece of code**.

- Black: The uncompromising code formatter:
  - Blackened code looks the same regardless of the project you're reading.
  - Formatting becomes transparent after a while and you can focus on the content instead.
  - Black makes code review faster by producing the smallest diffs possible.

Testing

- Would you state a new theorem without giving its proof?

Testing

- Would you state a new theorem without giving its proof?

- Would you apply a theorem without checking if the hypotheses are satisfied?

Testing

- Would you state a new theorem without giving its proof?
- Would you apply a theorem without checking if the hypotheses are satisfied?
- Would you trust anyone's code (including yours) without it being tested?

Testing

Objective: Testing that your code **works** and **does what it is supposed to do**.

- **Unit testing**: Testing individual modules of an application in isolation to confirm that the code is doing things right.
- **Integration testing**: Checking if different submodules of your project are working fine when combined together.
- **Functional testing**: Testing a functionality in the project (may interact with dependencies) to confirm that the code is doing the right things.

Testing in Python

- `unittest`: Python package from the standard library.
- ~~`nose`~~: deprecated Python package.
- `pytest`: the most popular Python package (easier, more flexible).

Code coverage

- **Definition**: a measure used to describe the degree to which the source code of a program is executed when a particular test suite is run.
- Common metric: percentage of lines that have been executed at least once. Available at any level:
  - in the whole module,
  - in any submodule,
  - in any file.
- Reliant on the report of the testing tool used to run the test suite.

Code coverage in Python

- `coverage`: general tool (initially developed to be used with `unittest`).

- `pytest-cov`: plugin for `pytest`.

Code coverage (online)

- Reporting the code coverage results online has several upsides:
  - **Information easily available to anyone** (no need to run a command)
  - **User-friendly report** (sunburst graph, code coverage at any level, etc.)
  - Can be included in the **continuous integration** pipeline (e.g., monitoring the change in code coverage in a pull request)

- Available tools:
  - Codecov
  - Coveralls

Codecov

COVERALLS

Documentation

- A software (and more generally any source code) **without its corresponding documentation** is almost **useless**.

- **Key elements** of any documentation:
    - Installation instructions
    - User guide
    - API documentation
    - Examples

- Other useful elements: getting started, tutorials, changelog, glossary, developer guide, etc.

Documentation in Python

- `Sphinx`: Python documentation generator
  - Originally created for the Python documentation
  - Expanded to other programming languages (C, PHP, Ruby, JavaScript, etc.)
  - Many useful extensions, including:
    - `sphinx.ext.autodoc`: Include documentation from docstrings
    - `sphinx.ext.autodoc`: Generate autodoc summaries
    - `sphinx.ext.viewcode`: Add links to highlighted source code
    - `sphinx.ext.doctest`: Test snippets in the documentation
    - `sphinx_gallery`: Build an HTML gallery of examples from any set of Python scripts

- `MkDocs`: project documentation with Markdown

Documentation (online)

- A **website** dedicated to the documentation is much more user-friendly than a PDF file with hundreds or even thousands of pages.

- ReadTheDocs: Simplify software documentation by automating building, versioning, and hosting of your docs for you.

- GitHub Pages: Websites for you and your projects.
  - Hosted directly from your GitHub repository.
  - Just edit, push, and your changes are live.

- Automatically redirect to another website if you own a dedicated domain.

Continuous integration

- **Rationale**: Making sure that any version of the remote source code always works.
- **Content**: linting, testing, code coverage, documentation, etc.
- **Workflow**: Before changing the remote source code:
  1. Run the continuous integration locally.
  2. Run the continuous integration remotely (several operating systems, several versions of dependencies, etc.).
  3. If successful, the changes can be merged.

Continuous integration (online)

Many services available, all of them being free for open source projects (with reasonable restrictions), including:

- Azure Pipelines
- GitHub workflows
- Travis CI
- CircleCI
- AppVeyor
- Jenkins

Outline

What is pyts?

- Python package dedicated to time series classification.

- **Objective**: Make working on time series classification easy:
  - Data loading utilities, preprocessing tools, implementations of many algorithms,
  - Under a unified application programming interface,
  - Compatible with `scikit-learn` tools such as cross-validation and pipelines.

- Published in the *Open Source Section* of *Journal of Machine Learning Research* in 2020 [FJ20].

Concrete example

Let's see how the tools presented in the second section are applied in this package.

Thanks

Thank you for your attention

References I

[Bag+15] Anthony Bagnall et al. "Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles". In: *IEEE Transactions on Knowledge and Data Engineering* 27.9 (Sept. 2015), pp. 2522–2535.

[Bag+20] Anthony Bagnall et al. "On the Usage and Performance of the Hierarchical Vote Collective of Transformation-Based Ensembles Version 1.0 (HIVE-COTE v1.0)". In: *Advanced Analytics and Learning on Temporal Data*. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 3–18.

[BRT13] Mustafa Gokce Baydogan, George Runger, and Eugene Tuv. "A Bag-of-Features Framework to Classify Time Series". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11 (Nov. 2013), pp. 2796–2802.

[CB17] Marco Cuturi and Mathieu Blondel. "Soft-DTW: a Differentiable Loss Function for Time-Series". In: *Proceedings of the 34th International Conference on International Conference on Machine Learning*. PMLR. 2017, pp. 894–903.

[Cut11] Marco Cuturi. "Fast Global Alignment Kernels". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. June 2011, pp. 929–936.

[Den+13] Houtao Deng et al. "A time series forest for classification and feature extraction". In: *Information Sciences* 239 (Aug. 2013), pp. 142–153.

References II

[DPW20] Angus Dempster, François Petitjean, and Geoffrey I. Webb. "ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels". In: *Data Mining and Knowledge Discovery* 34.5 (Sept. 2020), pp. 1454–1495.

[DSW21] Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. "MiniRocket: A Very Fast (Almost) Deterministic Transform for Time Series Classification". In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. Aug. 2021, pp. 248–257.

[EKR87] J.-P. Eckmann, S. Oliffson Kamphorst, and D. Ruelle. "Recurrence Plots of Dynamical Systems". In: *Europhysics Letters (EPL)* 4.9 (Nov. 1987), pp. 973–977.

[FJ20] Johann Faouzi and Hicham Janati. "pyts: A Python Package for Time Series Classification". In: *Journal of Machine Learning Research* 21.46 (2020), pp. 1–6.

[Gra+14] Josif Grabocka et al. "Learning Time-Series Shapelets". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining.* 2014, pp. 392–401.

[Ism+20] Hassan Ismail Fawaz et al. "InceptionTime: Finding AlexNet for time series classification". In: *Data Mining and Knowledge Discovery* 34.6 (Nov. 2020), pp. 1936–1962.

References III

[Ita75] F. Itakura. “Minimum prediction residual principle applied to speech recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1 (Feb. 1975), pp. 67–72.

[Lin+07] Jessica Lin et al. “Experiencing SAX: a novel symbolic representation of time series”. In: *Data Mining and Knowledge Discovery* 15.2 (Oct. 2007), pp. 107–144.

[Lin+12] Jason Lines et al. “A Shapelet Transform for Time Series Classification”. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2012, pp. 289–297.

[LKL12] Jessica Lin, Rohan Khade, and Yuan Li. “Rotation-invariant similarity in time series using bag-of-patterns representation”. In: *Journal of Intelligent Information Systems* 39.2 (Oct. 2012), pp. 287–315.

[LTB18] Jason Lines, Sarah Taylor, and Anthony Bagnall. “Time Series Classification with HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles”. In: *ACM Transactions on Knowledge Discovery from Data* 12.5 (July 2018), 52:1–52:35.

[Luc+19] Benjamin Lucas et al. “Proximity Forest: an effective and scalable distance-based classifier for time series”. In: *Data Mining and Knowledge Discovery* 33.3 (2019), pp. 607–635.

References IV

[Mid+21] Matthew Middlehurst et al. “HIVE-COTE 2.0: a new meta ensemble for time series classification”. In: *arXiv:2104.07551 [cs]* (Apr. 2021).

[MMK06] Meinard Müller, Henning Mattes, and Frank Kurth. “An efficient multiscale approach to audio synchronization”. In: *In Proceedings of the 6th International Conference on Music Information Retrieval*. 2006, pp. 192–197.

[SC07] Stan Salvador and Philip Chan. “Toward Accurate Dynamic Time Warping in Linear Time and Space”. In: *Intelligent Data Analysis* 11.5 (Oct. 2007), pp. 561–580.

[SC78] H. Sakoe and S. Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (Feb. 1978), pp. 43–49.

[Sch15] Patrick Schäfer. “The BOSS is concerned with time series classification in the presence of noise”. In: *Data Mining and Knowledge Discovery* 29.6 (Nov. 2015), pp. 1505–1530.

[Sch16] Patrick Schäfer. “Scalable time series classification”. In: *Data Mining and Knowledge Discovery* 30.5 (Sept. 2016), pp. 1273–1298.

[SH12] Patrick Schäfer and Mikael 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 - EDBT ’12*. Berlin, Germany: ACM Press, 2012, p. 516.

References V

[Shi+20] Ahmed Shifaz et al. "TS-CHIEF: a scalable and accurate forest algorithm for time series classification". In: *Data Mining and Knowledge Discovery* 34.3 (May 2020), pp. 742–775.

[SL17] Patrick Schäfer and Ulf Leser. "Fast and Accurate Time Series Classification with WEASEL". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17* (2017), pp. 637–646.

[SM13] P. Senin and S. Malinchik. "SAX-VSM: Interpretable Time Series Classification Using SAX and Vector Space Model". In: *2013 IEEE 13th International Conference on Data Mining*. Dec. 2013, pp. 1175–1180.

[Tan+21] Chang Wei Tan et al. "MultiRocket: Effective summary statistics for convolutional outputs in time series classification". In: *ArXiv* (2021).

[WO15] Zhiguang Wang and Tim Oates. "Imaging Time-series to Improve Classification and Imputation". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. AAAI Press, 2015, pp. 3939–3945.