Computer Science and Information Systems 22(4):1379–1403 https://doi.org/10.2298/CSIS240910029G

FAP: A Time Series Analysis and Mining Framework for Scientific and Practical Applications

Zoltán Gellér[1], Vladimir Kurbalija[2], and Mirjana Ivanović[2]

[1] University of Novi Sad, Faculty of Philosophy, Department of Media Studies
Dr Zorana Đinđića 2, 21000 Novi Sad, Serbia
zoltang@ff.uns.ac.rs
[2] University of Novi Sad, Faculty of Sciences, Department of Mathematics and Informatics
Trg D. Obradovića 4, 21000 Novi Sad, Serbia
{kurba, mira}@dmi.uns.ac.rs

Abstract. Given the exponential growth of data in modern society, data analysis tools have become increasingly pivotal in a wide range of fields, such as business, advertising, economy, medicine, biology, meteorology, astronomy, agriculture, and others. As the time component often plays an essential role in data analysis, the application and research of different methods for examining temporal data is among the current interests of both practitioners and researchers. This paper presents the main capabilities of the Framework for Analysis and Prediction (FAP), a free and open source Java library designed for processing and mining time series data that has been successfully applied both in research and education since its initial presentation.

Keywords: time series, data mining, open source, software library, Java.

1. Introduction

Past two decades influenced significant changes in processing huge amounts of data. The need to process such ever-growing amounts of data from different sources all over the world, importance of developing and applying different approaches of data mining and machine learning has gained more and more attention. They are unavoidable instruments of applications in computer science, ICT, and education [3, 57].

An emerging sub-field of data mining is temporal data mining that is focused on knowledge discovery from huge amounts of temporal data [51]. Time series are the most common form of temporal data which are composed of real values usually sampled at regular time intervals [28]. The chronologically represented arrays of numbers are collected in different domains and they are used to express the change of the observed phenomena over time, like financial sector, economics, engineering, meteorology, medicine [58, 31] as well as in other areas of natural and social sciences [9].

Statistical analysis of time series [37] is mainly focused on identifying patterns, trend analysis, seasonality and forecasting [17]. On the other hand, data mining of time series is focused on tasks like prediction, classification, clustering, indexing, anomaly detection, data representation, distance measures and others [20, 63]. Laxman and Sastry [45] considered and presented significant differences between statistical analysis and temporal data mining: data-mining approaches effectively analyze much larger volumes of data.

More important is that their field of interest exceeds the scope and limitations of statistical time-series analysis.

The numerous possibility of applying time series for storage, analysis, and visualization of big data collections influenced a significant growth of interest in researching in significant aspects and tasks of time-series data mining. The methods presented in [63] have always claimed a particular superiority over previously achieved results.

Contemporary research in domain of time series data mining inspired authors to produce free and open sourced support and services that could assist and facilitate researching new and comparing existing techniques in this domain. Usefulness of such freely available high-quality services could help in more productive and quality research but also educational processes.

All mentioned highly motivated us to significantly improve our previously developed framework for time series processing and analysis [24]. The extended version of FAP (Framework for Analysis and Prediction) implements significant number of essential algorithms in the field of time-series data mining: time-series representations, distance/similarity measures, preprocessing, classification, classifier evaluation techniques with a focus on efficiency, multi-threading, and resumability of time-consuming tasks.

In this paper, we will give a comprehensive analysis of the newly developed functionalities and capabilities of FAP that are essential for researchers and educators to facilitate their research and applications of time series data mining. All the features provided by FAP were implemented from scratch; it does not utilize any other underlying libraries.

The rest of the paper is organized as follows. The second Section is devoted to an extensive review of related work. Central Section 3 deals with various concepts of time series analysis and data mining and their realization within FAP. Concluding remarks are given in last Section.

2. Related Work

The interest in time series has increased dramatically in the past decade. The main reason is the exponential growth of data available for various machine learning and decision support system. A significant part of this data is available in form of time series. Consequently, a large number of frameworks and systems which can help in time-series analysis was developed and improved recently. This section will give an overview of these systems, and also will elaborate the context and motivation for developing FAP.

The investigation of time series is usually based on two important methodologies: statistical analysis and data mining. Time series statistical analysis is an approach used to analyze data points collected or recorded at specific time intervals using well established methods from statistics and econometrics. This type of analysis helps identify patterns, trends, and other characteristics within the data over time. On the other hand, data mining is a newer discipline focused on analyzing complex, massive datasets to extract useful knowledge. Time-series data mining involves analyzing time-dependent data for tasks like forecasting and anomaly detection. Both approaches (statistical analysis and data mining) have numerous systems which can offer assistance in corresponding time-series analysis. Additionally, several high-level programming languages offer powerful packages and libraries which can considerably help in various time-series tasks.

In the market, there is a considerable number of systems that enable time-series analysis relying on statistical and econometric concepts. Probably the most widely used is SAS. SAS[3] (Statistical Analysis System) is a complete software suite widely used for statistical analysis and data management [38]. It offers several tools for time series modeling and forecasting. Key components for time series analysis include:

1. SAS/ETS (Econometric Time Series): A specialized module designed for time series analysis and forecasting, supporting models such as ARIMA, exponential smoothing, state space models, and multivariate time series analysis. It includes functions for model estimation, diagnostics, forecasting, and simulation.
2. SAS/STAT: Provides a wide range of statistical procedures, including time series decomposition, autocorrelation analysis, spectral analysis, and structural time series models. It also incorporates ARIMA and GARCH model fitting, unit root tests, and handling missing values.
3. SAS Forecast Studio: A graphical interface that simplifies building and evaluating time series forecasting models. It enables visual exploration, model selection, parameter specification, and forecast accuracy assessment, integrating with SAS/ETS and SAS/STAT for seamless model estimation and forecasting.

SPSS[4] is also very widely used and influential statistical software which provides tools for statistical analysis, data management, and documentation. It offers following key features for time series analysis: data management (import, merge, clean, recode, and handle missing values in time series data), descriptive statistics (calculate measures like mean, median, standard deviation, and percentiles to summarize time series data), time series visualization, autocorrelation analysis, and forecasting (methods like ARIMA and exponential smoothing, with accuracy assessment options).

GRETL[5] (GNU Regression, Econometrics and Time-series Library) is a platform-independent, open source software package for econometric analysis [6]. It offers a very intuitive interface, parallelization, and an integrated powerful scripting language. In time-series analysis several concept are provided; ARIMA, GARCH-type models, VARs and VECMs (including structural VARs), unit-root and cointegration tests, Kalman filter, etc.

Stata[6] is a widely used statistical software developed by StataCorp for data manipulation, visualization, statistics, and automated reporting. It is widely used by researchers in various fields, such as: economics, epidemiology, biomedicine, and sociology. Stata also offers comprehensive tools for time series analysis: time series data management, descriptive statistics, graphical analysis, and time series modeling.

On the other side of the spectrum of available time-series software there is general data mining software with support for time-series. Since the number of these systems is huge, we will limit our analysis only on non-commercial, research oriented software.

Weka[7] (Waikato Environment for Knowledge Analysis) [64] is an open-source software suite designed for machine learning and data mining tasks. It provides a collection of algorithms and tools for data preprocessing, classification, regression, clustering, association rules, and visualization. Weka is particularly popular in educational and research

[3] http://www.sas.com
[4] https://www.ibm.com/spss
[5] http://gretl.sourceforge.net/
[6] https://www.stata.com/
[7] https://ml.cms.waikato.ac.nz/weka

communities due to its ease of use and comprehensive documentation. It has good capabilities for time series analysis and forecasting through a dedicated environment that can be accessed via its graphical user interface. This environment allows users to develop, evaluate, and visualize forecasting models. Additionally, there are packages like `TS-Classification` that facilitate time series classification tasks in Weka.

RapidMiner[8] is a Java-based data mining and machine learning platform offering features like data loading, transformation, preprocessing, visualization, predictive analytics, statistical modeling, evaluation, and deployment. It provides a graphical user interface to design and execute analytical workflows called "Processes," which consist of multiple "Operators" each performing a specific task. The output of one operator serves as input for the next. RapidMiner can also be accessed via an API or command line, and its functionality can be extended using R and Python scripts for custom operations. It provides strong capabilities for time series analysis through its integrated time series extension.

ELKI[9] (Environment for DeveLoping KDD-Applications Supported by Index Structures) is an open-source data mining software written in Java. It is primarily designed for research in algorithms, with a strong focus on unsupervised methods such as cluster analysis and outlier detection. ELKI facilitates time series analysis through its ability to evaluate various distance measures and algorithms specifically designed for time series data. Several time-series concepts are implemented in ELKI: distance measures, various time-series algorithms and visualization tools.

KNIME[10] (KoNstanz Information MinEr) [8] is a free, open-source platform for data analytics, reporting, and integration. It uses a modular, "Building Blocks of Analytics" concept for machine learning and data mining, allowing users to blend data sources and perform tasks like preprocessing, modeling, and visualization through a graphical interface, minimizing the need for programming. KNIME provides comprehensive tools for time series analysis through its various components and extensions.

The third large group of time-series software is the group of programming languages with powerful libraries for time-series analysis. Here we will give an overview of modern and actively used languages with this property.

Python is a general-purpose programming language [69] which can be used for time series analysis due to its extensive ecosystem of libraries and tools. These libraries offer a variety of tools and models that can help in analysis and forecasting time series. Libraries with functionalities for time-series forecasting, anomaly detection, and feature extraction include: `Tsfresh`, `Darts`, `Kats`, `GreyKite` and `AutoTS`.

R[11] is a comprehensive language that is well-suited for a wide variety of statistical analyses. It also offers a variety of packages designed for handling time series data. Key packages include: `forecast` (methods like exponential smoothing, ARIMA, and state space models for time series forecasting), `tseries` (tools for unit root tests, seasonality tests, time series decomposition, detrending, and differencing), and `zoo` (support for irregularly spaced time series, with efficient data structures for manipulation, subsetting, merging, handling missing values, and aggregating data over irregular intervals).

[8] https://altair.com/altair-rapidminer
[9] https://elki-project.github.io/
[10] https://www.knime.org/
[11] https://www.r-project.org

MATLAB is a high-level programming and numeric computing platform developed by MathWorks [25]. It is widely used by engineers and scientists for a variety of applications, including data analysis, algorithm development, and modeling. Key features for time-series manipulation include: time series objects (`Timeseries` and `timetable` objects for efficient manipulation, indexing, and visualization of time series data), signal processing toolbox (functions for filtering, spectral analysis, Fourier and wavelet analysis, and time-frequency analysis on time series data), econometrics toolbox (tools for econometric time series analysis, including model estimation, forecasting, unit root tests, panel data analysis, and multivariate time series models), and financial toolbox (focuses on financial time series analysis, offering tools for analyzing market data, portfolio optimization, and risk measurement).

Julia [59] is a high-level, high-performance programming language designed for technical computing, particularly in areas like data science, machine learning, and numerical analysis. It also has strong capabilities for time series analysis. Two popular packages are: (1) `TimeSeries.jl` - A comprehensive package for handling time series data with efficient structures like TimeArray and TS. It provides tools for data manipulation, visualization, resampling, merging, differencing, and rolling window calculations, models such as AR, MA, and ARIMA; (2) `Econometrics.jl` - Focused on econometric modeling, it offers functions for time series models like ARDL, VAR, GARCH, and structural time series. The package includes tools for model estimation, hypothesis testing, diagnostic checking, and forecasting in econometrics.

Clearly, three types of software packages for time-series analysis and mining can be distinguished:

1. Statistical and econometric software systems that provide methods and tools for time-series data.
2. General-purpose data mining and machine learning systems that have extensions for time-series tasks.
3. General-purpose programming languages with libraries for time-series analysis.

Evidently, all of these packages, frameworks and systems have some disadvantages. Some of them are not free or open sourced, many of them are not primarily made for time series and the systems from the third group can't be used by non-programmers.

The system FAP, presented in this paper, tries to overcome all of these disadvantages. It is free and open-sourced. It is designed to work with time series and encompasses all main concepts for time-series analysis (pre-processing tasks, distance measures, time-series representations); and for time-series mining (indexing, classification, prediction). Finally, it can be used by experts from various fields since it can be used without any programming experience. In the past 15 years we have developed and constantly upgraded FAP system making it up-to-date with modern findings in time-series mining field. Furthermore, we successfully applied FAP in various domains both for research [42] and educational [41] purposes.

3. Essential functionalities of FAP for high quality time-series data mining

The `core` sub-packages of the FAP library (Fig. 1) define basic interfaces and classes for implementing various time series analysis and data mining concepts such as data

points, time series, datasets, representations (`data`), distance measure (`distance`), classifiers (`classifier`), classifier performance evaluators (`evaluator`), classifier trainers and distance measure tuners (`trainer`), predictors (`predictor`), as well as for loading data points from strings (`input`), and basic classes for checked and unchecked exceptions thrown within the library (`exception`).

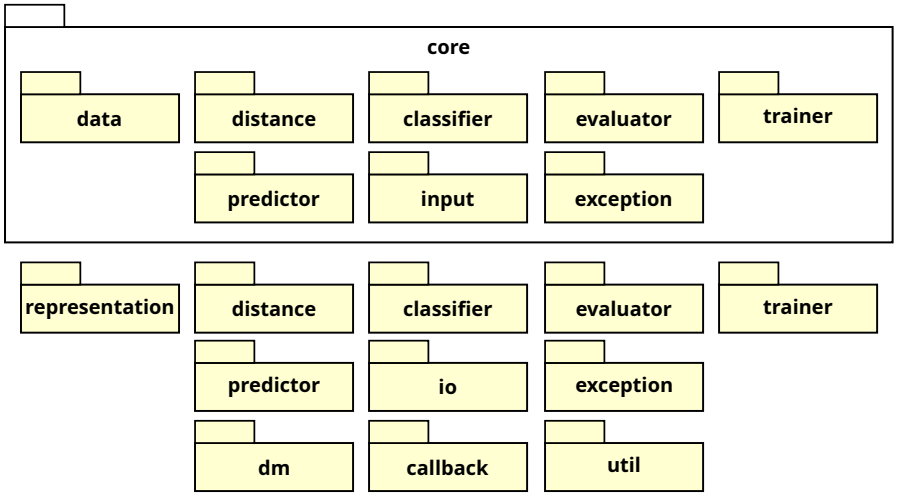

Fig. 1. Sub-packages of the FAP library

Specific implementations of the various time series processing tasks are provided in the appropriate first-level sub-packages. For example, the `fap.core.distance` package defines the `Distance` interface along with the auxiliary abstract class `AbstractDistance`, and classes that represent various distance measures by implementing that interface or extending the auxiliary class are placed in the `fap.distance` sub-package.

Since all base interfaces extend the `Serializable` interface and all classes offer parameterless constructors as well as public getter and setter methods for their properties, the FAP library also supports the JavaBeans standard.

In the rest of this section, we will provide an overview of the capabilities of the FAP library. The review will not cover the predictors (since they are in early stage of development) nor the `fap.core.input`, `fap.io`, and `fap.dm` sub-packages (since they deal with technical issues that are not necessary to understand the main functionalities of the framework). Furthermore, for brevity, the utility classes of the `fap.util` sub-package, which contain mathematical, statistical, and accessory methods intended to facilitate working with threads, strings, files, datasets, and time series (including preprocessing algorithms such as shifting, scaling, z-normalization, mean normalization, min-max normalization, maximum absolute normalization, and decimal scaling), will also be omitted.

To provide a more comprehensive view, the UML diagrams show only a representative subset of the constructors and methods of the classes, and only the types of their parameters.

3.1. Time Series and Representations

Time series are implemented in the form of a list of two-dimensional data points (Fig. 2) where the y coordinate describes the observed phenomenon at the timestamp specified by the x coordinate. A general assumption of all FAP classes that perform tasks related to time series processing (for example, calculating the distance between them) is that the data points are chronologically ordered (i.e., the x coordinate of the i-th element of the time series is less than the x coordinate of the $(i + 1)$-th element).

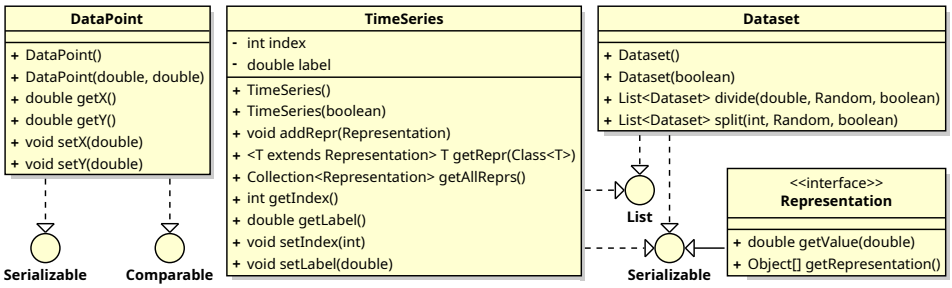

Fig. 2. Basic classes and interfaces

Each time series can be assigned a class label, an index, and a collection of representations. The index represents the unique identifier of the time series within the dataset to which it belongs (providing unique indices is the user's responsibility). It is used by distance measures to store calculated distances in memory (an optional feature that, when performing certain tasks, allows avoiding multiple calculation of distances between the same pairs of time series and thereby speeding up execution) and by kNN classifiers in combination with distance and neighbor matrices. A more detailed insight into how the index is used is given in the corresponding subsections.

To create an (indexed) time series, it is sufficient to specify the class label, index, and chronological list of its values. The x coordinates will be automatically initialized with values from 0 to n-1, where n is the number of elements in the list. For example, the following line of code creates a new time series with label 1.0, index 0, and values 2.0, 3.0, and 4.0 at timestamps 0.0, 1.0, and 2.0:

```
TimeSeries ts = new TimeSeries(1.0, 0, 2.0, 3.0, 4.0);
```

By default, time series are stored in `ArrayLists`. If we want to use `LinkedLists` instead, we just need to add `true` as the first parameter of the constructor:

```
TimeSeries ts = new TimeSeries(true, 1.0, 0, 2.0, 3.0, 4.0);
```

Datasets represent lists of time series, where, analogously to time series, `ArrayLists` are used for storage by default. By specifying the `boolean` value `true` as the first parameter of the constructor, the data structure for storing the time series will be `LinkedList`.

The `Dataset` class offers several methods for partitioning datasets into two (`divide`) or two or more subsets of approximately the same size (`split`), with the ability to control shuffling and stratification. Thus, by applying the `divide(20.0)` method, a list containing

two stratified subsets of the given dataset is obtained: the first subset will contain 20% and the second 80% of its time series. Similarly, a list of 10 stratified subsets of approximately the same size is obtained utilizing the `split(10)` method. The obtained subsets will be stored in the same type of list as the original dataset.

According to the definition given by Esling and Agon [20], a representation of a time series A of length n is a model $\bar{A}$ of length m (where $m \ll n$) that closely approximates A. FAP's interface requires classes implementing time series representations to be able to return the value of the time series at a given timestamp according to their model, and also the representation itself in the form of an array (Fig. 2).

The `fap.representation` sub-package offers implementation of several time-series representations: based on discrete Haar wavelet transform [36, 23, 1, 62, 12], discrete Fourier transform [4, 21, 54, 55, 66], spline [40], Piecewise Linear Approximation (PLA) [34], Piecewise Aggregate Appoximation (PAA) [32, 67, 35, 47], Indexable Piecewise Linear Approximation (IPLA) [15], Piecewise Aggregate Appoximation (PAA) [32, 67, 35, 47], Adaptive Piecewise Constant Approximation (APCA) [33], and Symbolic Aggregate Approximation (SAX) [47, 46].

3.2. Distance Measures

Distance measures should implement the `Distance` interface, declaring a single method whose task is to return the distance between two time series passed as its parameters (Fig. 3).

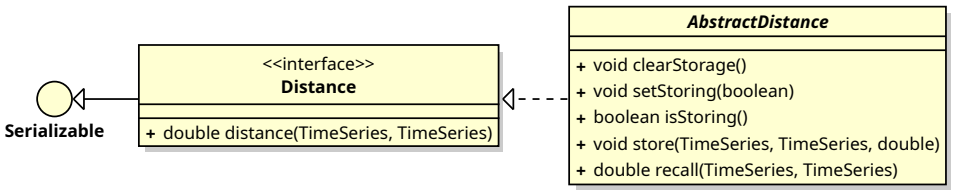

Fig. 3. Distance measures

All the distance measures in the `fap.distance` sub-package inherit the abstract class `AbstractDistance` offering the ability to store calculated distances between pairs of time series for reuse, which can speed up the execution of some tasks. The basic condition for using this mechanism is that the time series are assigned unique indices (see the previous subsection), namely, distances are stored in a (thread-safe) hash map whose keys are the indices of the time series. In addition, all of the distance measures implement the `Copyable` auxiliary interface discussed in subsection 3.6.

Listing 1 shows the implementation of the Manhattan distance as an example of utilizing this mechanism. Before calculating the distance between two time series, it should be checked whether the distance has already been calculated and saved in memory. This is achieved by relying on the `recall` method which returns `NaN` if the requested distance is not yet stored in the hash map. Memorizing new distances is achieved by calling the `store` method, and the utilization of the storage mechanism is controlled via the `setStoring`

Listing 1. Implementation of the Manhattan distance relying on the mechanism for storing calculated distances

```
double distance = recall(series1, series2);
if (!Double.isNaN(distance))
        return distance;

int len = IncomparableTimeSeriesException.checkLength(series1, series2);

distance = 0;

for (int i = 0; i < len; i++) {

        double y1 = series1.getY(i);
        double y2 = series2.getY(i);

        distance += Math.abs(y1 - y2);

}

store(series1, series2, distance);

return distance;
```

method. Additionally, any distance measure that uses this feature must clear the contents of the underlying hash map by calling the `clearStorage` method if the value of any of its parameters that affect the distance between time series changes.

Currently, the following distance measures based on linear matching of time series data points are implemented [18, 11, 2]: Euclidean, Manhattan, Chebyshev, Minkowski, Canberra, Kulczynski, Lorentzian, Soergel, Sørensen (Bray-Curtis), and Wave-Hedges. Within them, 0/0 is treated as 0, and the zero denominator is replaced with the value provided by the `getZeroDenominator` method of the `MathUtils` auxiliary class of the `fap.util` sub-package, as recommended in [11].

The list of implemented elastic distance measures includes Dynamic Time Warping (DTW) [7], Longest Common Subsequence (LCS) [61], Edit distance with Real Penalty (ERP) [13], Edit Distance on Real sequence (EDR) [14], and Time Warp Edit Distance (TWED) [49]. Their elasticity can be adjusted by applying the Sakoe-Chiba [56] or Itakura [30] global constraints.

3.3. Classifiers

In order for a class to be used to classify time series, it must implement the `Classifier` interface (Fig. 4), which declares two methods. The `initialize` method serves for (optional) initialization of the classifier and is not intended for its training. Classification is realized through the `classify` method, which should return the predicted class label.

Distance-based classifiers should implement the `DistanceBasedClassifier` interface that extends the base interface with getter/setter methods to access and update the distance measure to rely on. The abstract convenience class `AbstractDistanceBasedClassifier` stores the distance measure in the `distance` field with protected access level.

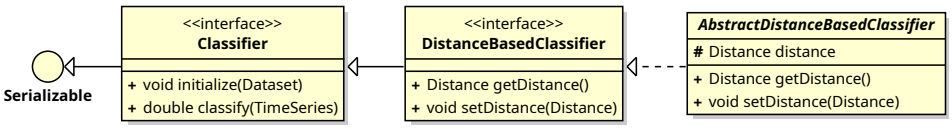

Fig. 4. Classifiers and distance-based classifiers

Presently, the `fap.classifier.NN` sub-package offers the implementation of the nearest neighbor (1NN) rule [16], the majority-voting kNN classifier [22, 50] and several of its weighted variants relying on the inverse of the distances [19], the inverse of the squared distances [50, 44, 60], Dudani's weighting scheme [19], the dual distance-weighted function [26], the uniform and dual-uniform weighting techniques [27], Zavrel's weighting scheme [68], Macleod's weighting function [48], the neighbours' ranks [19], and the Fibonacci weighting function [52]. In the case of weighted kNN variants based on the inverse and inverse of the squared distances, to avoid division by zero, a small value is added to the denominator. By default, it is initialized with the value returned by the `getZeroDenominator` method of the utility class `MathUtils` of the `fap.util` sub-package.

By implementing the `Multithreaded` and `Copyable` auxiliary interfaces, all NN classifiers enable multi-threaded execution of the classification process and making copies of themselves (see subsection 3.6 for details).

For additional acceleration of classification, NN classifiers also support the use of pre-generated distance and neighbor matrices which can be set and accessed through the `setDistances`, `setNeighbours`, `getDistances`, and `getNeighbours` methods.

A distance matrix is a (diagonal) matrix that in the intersection of the i-th row and the j-th column contains the distance between the time series with indices i and j of the given dataset. As an example, part of the distance matrix generated by applying the DTW distance measure to the *SyntheticControl* dataset from the UCR Time Series Classification Archive [5] is given in Table 1. Sub-package `fap.dm` contains classes for (multi-threaded) generation of distance matrices.

The intersection of the i-th row and the j-th column of the neighbor matrix contains the index of the j-th nearest neighbor of the time series with index i in the given dataset. Table 2 shows part of the neighbor matrix generated by applying the DTW distance measure to the *SyntheticControl* dataset.

3.4. Evaluators

Classifier evaluators should implement the `Evaluator` interface depicted in Fig. 5. The classifier performance evaluation algorithm should be implemented within the `evaluate` method, which has three parameters: the trainer (see the next subsection), the classifier, and the whole dataset. The evaluator should split the dataset into test and training subsets, train the classifier using the training set and the trainer, and evaluate the performance of the trained classifier on the test set. As the result, the method should return the classification error rate. This same value should be returned by the `getErrorRate` method, and the result of the call to the `getMisclassified` method should be the number of misclassified time series.

Table 1. The first ten rows and columns of the distance matrix of the *SyntheticControl* dataset generated by applying the DTW distance measure

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0									
2	24.42	0								
3	25.33	22.43	0							
4	22.90	26.85	25.74	0						
5	31.99	23.16	30.41	30.67	0					
6	35.06	22.19	33.38	31.65	23.53	0				
7	34.63	25.50	31.03	28.95	25.81	24.12	0			
8	26.52	32.72	30.17	24.70	34.85	35.59	31.54	0		
9	32.69	26.65	25.22	33.74	41.46	41.56	31.07	30.77	0	
10	21.22	26.66	31.62	23.26	32.28	32.32	28.81	26.14	27.32	0

Table 2. Ten nearest neighbors of the first ten time series of the SyntheticControl dataset obtained by applying the DTW distance measures

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	322	305	320	10	12	14	26	49	321	4
2	342	322	338	17	348	313	324	49	12	31
3	310	323	38	347	333	2	340	315	349	316
4	12	310	305	22	28	343	21	1	10	322
5	309	345	12	311	47	48	50	330	319	2
6	44	2	36	5	30	7	346	339	311	50
7	41	324	330	348	44	12	48	311	309	47
8	18	12	28	16	27	305	328	350	4	341
9	303	334	38	329	307	337	315	350	306	37
10	305	317	1	49	20	28	17	4	320	350

The abstract convenience class `AbstractEvaluator` stores the classification error in the `errorRate` and the number of missclassified time series in the `misclassified` field with protected access level.

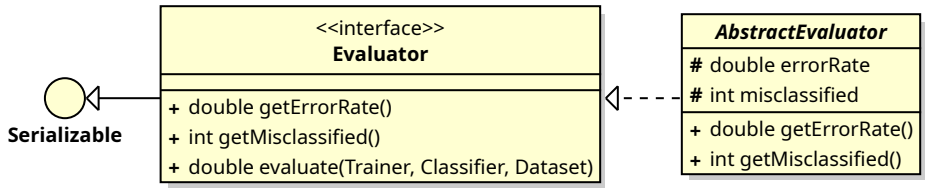

Fig. 5. Classifier evaluators

The `fap.evaluator` sub-package provides the following classes that implement the most common evaluation techniques [1, 60, 29]:

- `HoldoutEvaluator` - applies the Holdout method: a given percentage of the dataset constitutes the training set, and the rest is used as the test set.
- `CrossValidationEvaluator` - performs the cross-validation algorithm: the dataset is divided into k approximately equal subsets of which the union of $k-1$ subsets is used for training and one for testing. The procedure is repeated until each of the k subsets has been used as a test set (exactly) once. The classification error is calculated as the average of the errors obtained over the k test subsets.
- `LeaveOneOutEvaluator` - executes the leave-one-out procedure: the classifier is trained on a training set that contains all the time series of the original dataset except for one that is reserved for testing. The procedure is repeated until each time series of the initial dataset is excluded from the training process (and used for testing) exactly once.

The parameters of the `HoldoutEvaluator` and `CrossValidationEvaluator` classes allow the choice between stratified and non-stratified partitioning. In addition, by specifying an array of random seed values, a repeated variant of these two methods can be applied, where before each application, random shuffling of the initial dataset is performed based on the corresponding seed value. The final error rate is obtained by averaging over all repeated evaluations.

All three classes implement the `Callbackable`, `Resumable`, `Multithreaded`, and also the `Copyable` auxiliary interfaces (see subsection 3.6). Evaluator multi-threading takes precedence over trainer and classifier multi-threading, which means that in the case of multi-threaded evaluation, the number of threads of the trainer (when the evaluator performs the training multi-threaded) and the classifier will be set to 1 (if they implement the `Multithreaded` interface).

In the case of the `LeaveOneOutEvaluator` class multi-threaded execution requires that both the trainer and the classifier implement the `Copyable` interface (otherwise it will revert to single-threaded execution). This is necessary because the evaluation of the classifier in each iteration is reduced to the classification of a single time series, while all other

time series of the dataset are used for training the classifier, i.e. multi-threaded evaluation requires parallelization of training: each thread must have its own trainer and classifier.

The leave-one-out and cross-validation methods also offer the possibility of sequential evaluation of the classifier over individual splits of the dataset into training and test subsets. In each iteration, the classifier is tested multi-threaded on the test set after training on the training set. This approach does not require the implementation of the `Copyable` interface by either the trainer or the classifier.

In Figures 6 and 7, which illustrate the difference between full and partial parallelization on the example of m-fold cross-validation, `C` denotes the classificator, `C(i)` the i-th copy of `C` trained on the training set `trainset(i)` constructed in the i-th iteration, and `testset(i, j)` the j-th time series of the test set corresponding to the i-th iteration. If both the trainer and the classifier implement the `Copyable` interface (or if no trainer is specified and the classifier implements it), the holdout and cross-validation evaluators apply full parallelization by default.

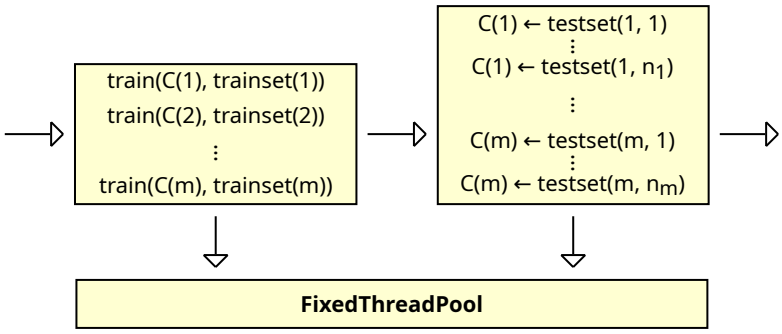

Fig. 6. Full parallelization of m-fold cross-validation

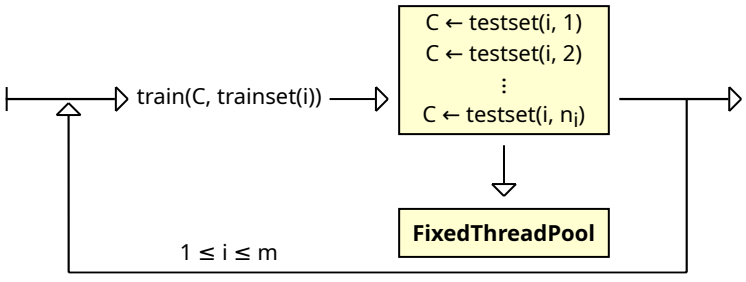

Fig. 7. Partial parallelization of m-fold cross-validation

Listing 2 demonstrates the evaluation of the 1NN classifier paired with the DTW distance measure over the *FiftyWords* dataset using 3 times repeated 10-fold cross-validation. The first parameter of the constructor of the evaluator determines the number of folds (10),

Listing 2. Evaluating the 1NN classifier paired with the DTW distance measure on the FiftyWords dataset with 3 times repeated 10-fold stratified cross-validation

```
Dataset dataset = DatasetUtils.loadDataset("FiftyWords");
Distance distance = new DTWDistance();
Classifier classifier = new NNClassifier(distance);
Trainer trainer = null;
Evaluator evaluator =
          new CrossValidationEvaluator(10, new long[] {1, 2, 3}, 0);

double error = evaluator.evaluate(trainer, classifier, dataset);
ThreadUtils.shutdown(evaluator);
```

the second parameter is an array of seed values (1, 2, 3) that should be used to initialize the random number generator utilized for shuffling the dataset before each run, and the last parameter defines the number of threads (0 means that it should use as many threads as processors are available to the Java Virtual Machine). Due to optimization, the underlying executor service is not automatically shut down after the evaluation is completed. Currently, shutdown should be initiated by the user.

The `getResults()` method of the `HoldoutEvaluator` and `CrossValidationEvaluator` classes returns an array of `FoldResult` objects (Fig. 8) that describe the results of individual iterations. In the case of the Holdout evaluator, iterations represent runs, and in the case of cross-validation, they correspond to folds. The `FoldResult` class defines only public fields that store the training and test sets, the number of misclassified time series of the test set along with the error rate, as well as the expected error and the list of the optimal parameter values found by the trainer (provided that the trainer supports retrieving them by implementing the `ParameterTrainer` interface presented in the next subsection).

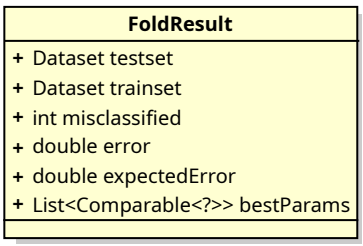

Fig. 8. A class for storing the results of individual iterations of the holdout and cross-validation evaluators

3.5. Trainers

Training a classifier on a given training set is the task of the `train` method declared by the `Train` interface shown in Fig. 9. Its result should be the expected classification error,

which should also be returned by the `getExpectedError` method. In the case of distance-based classifiers, through the `affectsDistance` method, the trainer should report whether it changes the parameters of the distance measures (this information is used by the evaluators described in the previous subsection to optimize the evaluation process when storing calculated distances between pairs of time series is enabled). Its default return value is `false`.

For convenience, the `AbstractTrainer` class stores the expected error and information about whether the training affects the distance measure in fields with protected access level.

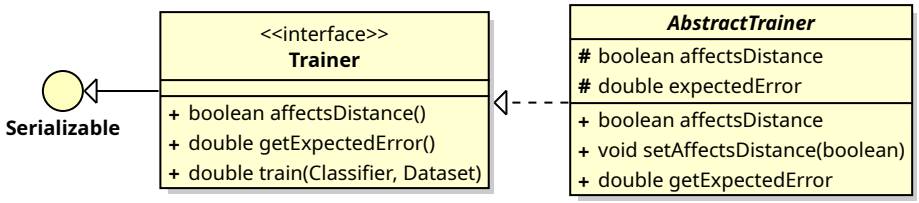

Fig. 9. Classifier trainers and distance measure tuners

The `ParameterTrainer` interface (Fig. 10) of the `fap.trainer` sub-package extends the `Trainer` interface by declaring methods for trainers that tune the value of a single parameter of a classifier or a distance measure (the type `T` of the parameter must implement the `Comparable` interface). Such trainers should provide getter and setter methods for the list of possible parameter values, the evaluator that evaluates their impact on classifier performance, and the sub-trainer that tunes some other parameter of the classifier or distance measure. In this way, specifying sub-trainers opens up the possibility of chaining a series of trainers.

After completing the training, the `getBestValue` method should return the optimal value of the parameter (the one that generated the smallest classification error). The return value of the `getParameters` method should be a list of optimal values of all the parameters tuned by the chained trainers: the first element of the list is the optimal value of the parameter tuned by the given trainer (and returned by the `getBestValue` method), the second element is the optimal value of the parameter tuned by the sub-trainer, and so on. Furthermore, when the `setParameters` method is called, the parameter trainer should set the parameter value to the first value of the specified list, and pass the rest of the list via the same method to the sub-trainer.

The `AbstractParameterTrainer` abstract class (Fig. 10) provides basic fields and methods for parameter trainers (it implements the `ParameterTrainer` interface and extends the `AbstractTrainer` class), including both sequential and parallel finding the optimal value. Since such a general implementation has no knowledge of which parameter it is tuning, nor whether it is a classifier parameter or a distance measure parameter, it is necessary to provide an auxiliary object that will assign the current value with the corresponding parameter. Such an object should implement the `Modifier` interface (Fig. 11), which defines two methods: `set` for assigning a given value to the parameter, and `affectsDistance`, which should report whether the parameter belongs to a distance measure (`true`) or a clas-

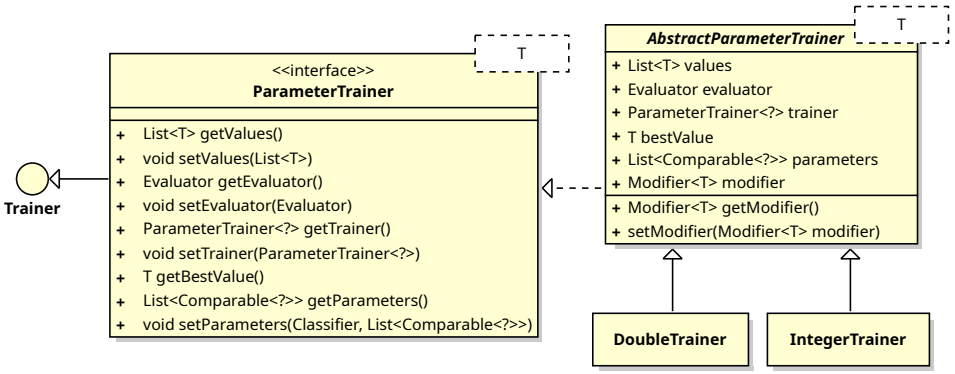

Fig. 10. Types for tuning the values of individual parameters of classifier or distance measures

sifier (`false`). The `DistanceModifier` and `ClassifierModifier` sub-interfaces contain only the corresponding (default) implementation of the `affectsDistance` method.

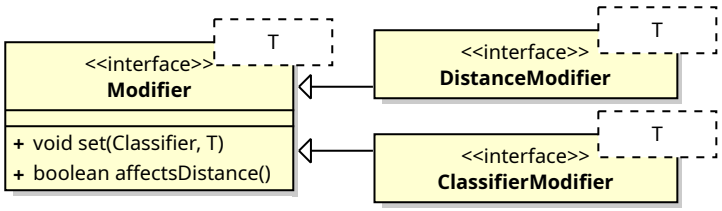

Fig. 11. Interfaces for parameter-modifier classes

An example of using trainers and modifiers to evaluate the weighted kNN classifier based on the Dudani's weighting function [19] and paired with the Sakoe-Chiba [56] constrained DTW [7] distance measure using nested cross-validation [65] is given in Listing 3. The optimal combination of the number of nearest neighbours and the width of the warping window is determined by applying 9-fold cross-validation within each iteration of the 10-fold cross-validation algorithm used to evaluate the classifier performance. The number of nearest neighbors is chosen from the interval between 1 to 10, and the (relative) width of the warping window from the interval between 0% to 25% of the time series length, both values are increased in unit steps.

Based on the result shown in Listing 4, it can be seen that the (average) classification error (rounded to 3 decimal places) was 0.050, i.e. approximately 5% of the time series of the *MoteStrain* dataset were misclassified (the dataset was preprocessed using Paparizzo's script [53]). In the first iteration of the evaluation process, the smallest error (0.046) over the training subset was obtained by a combination of one (1) nearest neighbor and a warping-window width that was 14% of the length of the time series. The actual classification error was 0.023 (calculated over the test subset by applying the classifier and distance measure trained over the training subset).

Listing 3. Evaluation of the weighted kNN classifier using nested cross-validation (utilizing Dudani's weighting function in combination with the Sakoe-Chiba constrained DTW distance measure)

```
Dataset dataset = DatasetUtils.loadDataset("MoteStrain");
Distance distance = new SakoeChibaDTWDistance(true);
Classifier classifier = new DudaniKNNClassifier(distance);

Evaluator subEvaluator = new CrossValidationEvaluator(9);
DoubleTrainer subTrainer =
             new DoubleTrainer(Modifiers.ELASTICITY, 0d, 25d);
subTrainer.setEvaluator(subEvaluator);

IntegerTrainer trainer = new IntegerTrainer(Modifiers.KNN, 1, 10);
trainer.setTrainer(subTrainer);

Evaluator evaluator = new CrossValidationEvaluator(10, 0);
double error = evaluator.evaluate(trainer, classifier, dataset);

ThreadUtils.shutdown(evaluator);

System.out.format("%.3f\n", error);

System.out.println("error, expected, parameters");
for (FoldResult fr : ((CrossValidationEvaluator) evaluator).getResults())
  System.out.format("%.3f, %.3f, " + fr.bestParams + "\n",
                    fr.error, fr.expectedError);
```

The `fap.trainer.Modifiers` class defines a modifier for each parameter of each classifier of the sub-package `fap.classifier.NN` and each distance measure of the sub-package `fap.distance` that relies on parameters. Listing 3 uses two of these modifiers: `KNN` to set the number of nearest neighbors and `ELASTICITY` to set the relative width of the warping window. Their source codes are given in Listing 5.

3.6. Auxiliary Interfaces

The `fap.util` and `fap.callback` sub-package auxiliary interfaces and classes briefly described in this subsection are intended to support mechanisms for monitoring, terminating, and resuming long-running processes, as well as their parallelization.

By implementing the `Resumable` interface (Fig. 12), classes that perform long-running tasks indicate that their execution can be interrupted and resumed from near the breakpoint. Via the `isDone` method, they should report whether the task has already been completed, and the `isInProgress` method should report whether it is still in progress (if the result of both methods is `false`, it means that the execution has not yet started). The function of the reset method is to reset the internal state of the object for reuse (for example, if the same trainer is to be used to train another classifier after finishing training the previous one).

Classes supporting multi-threaded execution must implement the `Multithreaded` interface (Fig. 12), which declares getter/setter methods to set and read the number of threads,

Listing 4. The output of the code shown in Listing 3

```
0.050
error, expected, parameters
0.023, 0.046, [1, 14.0]
0.055, 0.042, [6, 23.0]
0.039, 0.043, [4, 17.0]
0.071, 0.041, [6, 17.0]
0.055, 0.045, [1, 25.0]
0.039, 0.045, [4, 21.0]
0.063, 0.049, [6, 6.0]
0.063, 0.033, [4, 16.0]
0.024, 0.045, [4, 16.0]
0.063, 0.039, [6, 21.0]
```

Listing 5. Implementation of the modifiers of the number of nearest neighbors of the kNN classifier, and the width of the warping window of constrained elastic distance measures

```
public static final ClassifierModifier<Integer> KNN =
                    new ClassifierModifier<>() {

  @Override
  public void set(Classifier classifier, Integer value) {
    ((KNNClassifier) classifier).setK(value);
  }

};

public static final DistanceModifier<Double> ELASTICITY =
                    new DistanceModifier<>() {

  @Override
  public void set(Classifier classifier, Double value) {
    Distance distance =
             ((DistanceBasedClassifier) classifier).getDistance();
    ((ConstrainedDistance) distance).setR(value);
  }

};
```

and for stopping them (threads might not be stopped automatically after completing a task in order to optimize resource usage in case of reusing the same object for executing multiple tasks).

Parallel execution of some tasks requires that each thread be provided with a copy of the objects involved in the process of the task realization. For example, for each parallel partitioning of the dataset into testing and training subsets within repeated holdout evaluation, it is necessary to provide a copy of the trainer and classifier, and if the trainer changes the parameters of the distance measure used by the classifier, then also a copy of the distance measure. The ability of a class to make copies of objects of its type is indicated by implementing the `Copyable` interface (Fig. 12). Whether it is necessary to make a deep copy of the object is indicated by the value of the `boolean` parameter of the `makeACopy` method. For example, if a trainer changes the parameters of a distance measure, different copies of a classifier cannot share the same distance measure, and when copying a classifier, a copy of the associated distance measure must also be made. The parameterless form of this method is a shortcut for deep copying. The result of calling these methods should be a copy of the corresponding object.

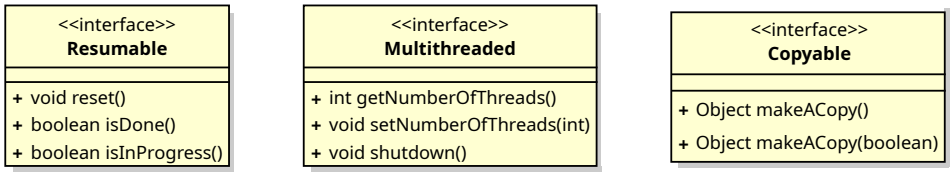

Fig. 12. Interfaces for resumable, multi-threaded and copyable tasks

Supporting monitoring the progress of task execution is indicated by implementing the `Callbackable` interface (Fig. 13) and it should be realized by regularly calling the `callback` method of the provided `Callback` object. The `Callback` object should report the desired number of callbacks via the `getDesiredCallbackNumber` method, and through `setPossibleCallbackNumber`, the `Callbackable` object can indicate the maximum number of callbacks it can perform. For example, if a `Callback` object requests 100 callbacks, but the task in question consists of only 60 steps, the number of possible callbacks should be reported as 60. Initialization and reading of the current number of callbacks should be enabled via the `setCallbackCount` and `getCallbackCount` methods. The purpose of the `getProgress` methods is to map the number of callbacks from the range of the possible number to the range of the desired number of callbacks.

The `AbstractCallback` abstract class provides basic data structures and a basic implementation of the methods of the `Callback` interface. The `fap.callback` sub-package besides it (and the `Callback` and `Callbackable` interfaces) also contains two concrete implementations: the `SystemOutCallback` class prints a specified character on the standard output with each callback, and the `ProgressBarCallback` class displays the progress through a given progress bar.

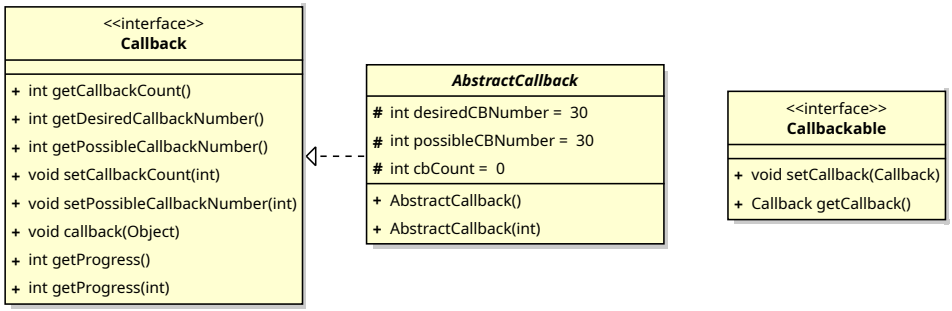

Fig. 13. Interfaces for callbackable tasks

4. Conclusions

In this paper, we presented the basic interfaces and classes around which our FAP library was built and demonstrated the ease of its utilization through examples of applying repeated and nested stratified cross-validation [65] to evaluate the performance of the 1NN and the variant of the weighted kNN classifier based on Dudani's scheme [19], paired with the unconstrained and the Sakoe-Chiba [56] constrained Dynamic Time Warping [7] (dis)similarity measure. In addition, we gave an insight into some more advanced capabilities of the framework, such as storing calculated distances between time series in memory to avoid multiple calculations, multi-threaded classification, evaluation and training of classifiers, and tuning of distance measure parameters for more efficient use of modern, multi-core processors, using pre-generated distance and neighbour matrices to speed up NN classifiers, as well as mechanisms for monitoring, interrupting and continuing interrupted long-term processes (considering environments where the availability of computers to run long-term experiments is not continuous - for example, university computer centers and classrooms).

Motivated by the need to develop a new representation of time series based on cubic splines [40, 43], the library was gradually expanded with new capabilities that enabled its application in both research [42, 39, 24, 10] and education [41]. Currently, the FAP library contains implementations of a number of distance measures based on linear matching of time series data points, the basic elastic measures whose elasticity can be constrained by applying either the Sakoe-Chiba band or the Itakura parallelogram [30], various variants of the NN classifier, multiple representations of time series, the main techniques for evaluating classifier performance (holdout, leave-one-out, cross-validation) with the possibility of multiple repetitions and nested evaluation, as well as classes for training classifiers and tuning parameters of distance measures.

In the future, we plan not only to expand the already existing sub-packages of the FAP library with additional capabilities, but also to implement solutions related to other areas of time series analysis and mining (such as, for example, clustering, anomaly detection, and prediction). Furthermore, believing that it may also be useful to other researchers and practitioners, the FAP library is open source and freely available via GitHub (https://github.com/zgeller/FAP.git).

Acknowledgments. The authors would like to thank Eamonn Keogh for collecting and making available the UCR time-series datasets, as well as everyone who contributed data to the collection. The authors would also like to thank the contribution of Brankica Bratić, Miklós Kálózi, and Aleksa Todorivić to the development of certain parts of the library. Vladimir Kurbalija and Mirjana Ivanović gratefully acknowledge the financial support of the Ministry of Science, Technological Development and Innovation of the Republic of Serbia (Grants No. 451-03-66/2024-03/200125 & 451-03-65/2024-03/200125) and the Science Fund of the Republic of Serbia, project #7462, Graphs in Space and Time: Graph Embeddings for Machine Learning in Complex Dynamical Systems – TIGRA.

References

1. Abonyi, J.: Adatbányászat a hatékonyság eszköze. ComputerBooks, Budapest, 1st edn. (2006)
2. Abu Alfeilat, H.A., Hassanat, A.B., Lasassmeh, O., Tarawneh, A.S., Alhasanat, M.B., Eyal Salman, H.S., Prasath, V.S.: Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review. Big Data 7(4), 221–248 (dec 2019), https://www.liebertpub.com/doi/10.1089/big.2018.0175
3. Aggarwal, C.C.: Data Mining: The Textbook. Springer Publishing Company, Incorporated (2015)
4. Agrawal, R., Faloutsos, C., Swami, A.: Efficient similarity search in sequence databases. In: David B. Lomet (ed.) Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO '93), Lecture Notes in Computer Science, vol. 730, pp. 69–84. Springer Berlin Heidelberg (1993), http://link.springer.com/10.1007/3-540-57301-1_5
5. Anh Dau, H., Keogh, E., Kamgar, K., Michael Yeh, C.C., Zhu, Y., Gharghabi, S., Ann Ratanamahatana, C., Chen, Y., Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G., Hexagon-ML: The UCR Time Series Classification Archive (2019), https://www.cs.ucr.edu/∼eamonn/time_series_data_2018/
6. Baiocchi, G., Distaso, W.: Gretl: Econometric software for the gnu generation. Journal of Applied Econometrics 18(1), 105–110 (2003), http://www.jstor.org/stable/30035190
7. Berndt, D., Clifford, J.: Using dynamic time warping to find patterns in time series. In: Usama M. Fayyad, R.U. (ed.) Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop. vol. 10, pp. 359–370. AAAI Press, Seattle, Washington (1994), http://dblp.uni-trier.de/rec/bib/conf/kdd/BerndtC94
8. Berthold, M.R., Cebron, N., Dill, F., Gabriel, T.R., Kötter, T., Meinl, T., Ohl, P., Sieb, C., Thiel, K., Wiswedel, B.: Knime: The konstanz information miner. In: Preisach, C., Burkhardt, H., Schmidt-Thieme, L., Decker, R. (eds.) Data Analysis, Machine Learning and Applications. pp. 319–326. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
9. Box, G.E.P., Jenkins, G.M., Reinsel, G.C., Ljung, G.M.: Time Series Analysis: Forecasting and Control. John Wiley & Sons, Inc., Hoboken, New Jersey, 5th editio edn. (2015)
10. Bratić, B.: Approximation algorithms for k-NN graph construction. Phd thesis, University of Novi Sad, Serbia (2021), https://nardus.mpn.gov.rs/handle/123456789/18059
11. Cha, S.H.: Comprehensive Survey on Distance/Similarity Measures between Probability Density Functions. International Journal of Mathematical Models and Methods in Applied Sciences 1(4), 300–307 (2007), http://www.gly.fsu.edu/∼parker/geostats/Cha.pdf
12. Chaovalit, P., Gangopadhyay, A., Karabatis, G., Chen, Z.: Discrete wavelet transform-based time series analysis and mining. ACM Computing Surveys 43(2), 1–37 (jan 2011), https://dl.acm.org/doi/10.1145/1883612.1883613
13. Chen, L., Ng, R.: On The Marriage of Lp-norms and Edit Distance. In: Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B.

(eds.) Proceedings 2004 VLDB Conference, vol. 04, pp. 792–803. Elsevier (2004), https://linkinghub.elsevier.com/retrieve/pii/B978012088469850070X
14. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05. pp. 491–502. SIGMOD '05, ACM Press, New York, New York, USA (2005), http://doi.acm.org/10.1145/1066157.1066213
15. Chen, Q., Chen, L., Lian, X., Liu, Y., Yu, J.X.: Indexable PLA for Efficient Similarity Search. In: Proceedings of the 33rd International Conference on Very Large Data Bases. pp. 435–446. VLDB '07, VLDB Endowment (2007), http://dl.acm.org/citation.cfm?id=1325851.1325903
16. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory 13(1), 21–27 (jan 1967), http://ieeexplore.ieee.org/document/1053964/
17. Das, G., Gunopulos, D.: Time Series Similarity and Indexing. In: Ye, N. (ed.) The Handbook of Data Mining, chap. 11, pp. 279–304. Human factors and ergonomics, Lawrence Erlbaum Associates, Mahwah, N.J. (2003)
18. Deza, M.M., Deza, E.: Encyclopedia of Distances. SpringerLink : Bücher, Springer Berlin Heidelberg, Berlin, Heidelberg (2016), http://link.springer.com/10.1007/978-3-662-52844-0
19. Dudani, S.A.: The Distance-Weighted k-Nearest-Neighbor Rule. IEEE Transactions on Systems, Man, and Cybernetics SMC-6(4), 325–327 (apr 1976), http://ieeexplore.ieee.org/document/5408784/
20. Esling, P., Agon, C.: Time-series data mining. ACM Computing Surveys 45(1), 12:1–12:34 (nov 2012), http://doi.acm.org/10.1145/2379776.2379788
21. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases. ACM SIGMOD Record 23(2), 419–429 (jun 1994), http://dl.acm.org/citation.cfm?id=191843.191925
22. Fix, E., Hodges, J.L.: Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties. International Statistical Review / Revue Internationale de Statistique 57(3), 238–247 (dec 1989), http://www.jstor.org/stable/1403797?origin=crossref https://www.jstor.org/stable/1403797?origin=crossref
23. Fu, A.W.c., Leung, O.T.W., Keogh, E., Lin, J.: Finding Time Series Discords Based on Haar Transform. pp. 31–41 (2006), http://link.springer.com/10.1007/11811305_3
24. Geler, Z.: Role of Similarity Measures in Time Series Analysis. Phd thesis, University of Novi Sad, Serbia (2015), https://nardus.mpn.gov.rs/handle/123456789/1703
25. Gilat, A.: MATLAB: An Introduction with Applications. John Wiley & Sons, Inc. (2017)
26. Gou, J., Du, L., Zhang, Y., Xiong, T.: A New distance-weighted k-nearest neighbor classifier. Journal of Information & Computational Science 9(6), 1429–1436 (2012)
27. Gou, J., Xiong, T., Kuang, Y.: A Novel Weighted Voting for K-Nearest Neighbor Rule. Journal of Computers 6(5), 833–840 (may 2011), http://ojs.academypublisher.com/index.php/jcp/article/view/4056
28. Grossmann, W., Rinderle-Ma, S.: Data Mining for Temporal Data. In: Fundamentals of Business Intelligence SE - 6, pp. 207–244. Data-Centric Systems and Applications, Springer Berlin Heidelberg (2015), http://dx.doi.org/10.1007/978-3-662-46531-8_6
29. Han, J., Pei, J., Tong, H.: Data Mining: Concepts and Techniques. The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufmann, 4 edn. (2022)
30. Itakura, F.: Minimum prediction residual principle applied to speech recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing 23(1), 67–72 (feb 1975), http://ieeexplore.ieee.org/document/1162641/
31. Ivanovic, M., Autexier, S., Kokkonidis, M., Rust, J.: Quality medical data management within an open AI architecture – cancer patients case. Connection Science 35(1) (dec 2023), https://www.tandfonline.com/doi/full/10.1080/09540091.2023.2194581
32. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. Knowledge and Information Systems 3(3), 263–286 (aug 2001), http://link.springer.com/10.1007/PL00011669

33. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Locally adaptive dimensionality reduction for indexing large time series databases. In: Proceedings of the 2001 ACM SIGMOD international conference on Management of data. pp. 151–162. SIGMOD '01, ACM, New York, NY, USA (2001), http://doi.acm.org/10.1145/375663.375680
34. Keogh, E.J., Chu, S., Hart, D., Pazzani, M.: Segmenting Time Series: A Survey and Novel Approach. In: Last, M., Kandel, A., Bunke, H. (eds.) Data Mining In Time Series Databases, Series in Machine Perception and Artificial Intelligence, vol. 57, chap. 1, pp. 1–22. World Scientific Publishing Company (2004)
35. Keogh, E.J., Pazzani, M.J.: Scaling up dynamic time warping for datamining applications. In: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 285–289. ACM, New York, NY, USA (aug 2000), https://dl.acm.org/doi/10.1145/347090.347153
36. Kin-Pong Chan, Ada Wai-Chee Fu: Efficient time series matching by wavelets. In: Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337). pp. 126–133. IEEE (1999), http://ieeexplore.ieee.org/document/754915/
37. Kirchgässner, G., Wolters, J., Hassler, U.: Introduction to Modern Time Series Analysis. Springer Texts in Business and Economics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2 edn. (2013), http://link.springer.com/10.1007/978-3-642-33436-8
38. Konasani, V.R., Kadre, S.: Practical Business Analytics Using SAS: A Hands-on Guide. Apress Berkeley, CA (2015), https://link.springer.com/book/10.1007/978-1-4842-0043-8
39. Kurbalija, V.: Time series analysis and prediction using case based reasoning technology. Phd thesis, University of Novi Sad, Novi Sad (oct 2009), http://www.doiserbia.nb.rs/phd/university.aspx?theseid=NS20091005KURBALIJA
40. Kurbalija, V., Ivanović, M., Budimac, Z.: Case-based curve behaviour prediction. Software: Practice and Experience 39(1), 81–103 (jan 2009), http://doi.wiley.com/10.1002/spe.891
41. Kurbalija, V., Ivanović, M., Geler, Z., Radovanović, M.: Two Faces of the Framework for Analysis and Prediction, Part 1 - Education. Information Technology And Control 47(2), 249–261 (jun 2018), http://itc.ktu.lt/index.php/ITC/article/view/18746
42. Kurbalija, V., Ivanović, M., Geler, Z., Radovanović, M.: Two Faces of the Framework for Analysis and Prediction, Part 2 - Research. Information Technology And Control 47(3), 489–502 (sep 2018), http://itc.ktu.lt/index.php/ITC/article/view/18747
43. Kurbalija, V., Radovanović, M., Geler, Z., Ivanović, M.: A Framework for Time-Series Analysis. In: Dicheva, D., Dochev, D. (eds.) Artificial Intelligence: Methodology, Systems, and Applications SE - 5. Lecture Notes in Computer Science, vol. 6304, pp. 42–51. Springer Berlin Heidelberg (2010), http://link.springer.com/10.1007/978-3-642-15431-7_5
44. Larose, D.T., Larose, C.D.: Discovering Knowledge in Data. Wiley, 2 edn. (jun 2014), https://onlinelibrary.wiley.com/doi/book/10.1002/9781118874059
45. Laxman, S., Sastry, P.S.: A survey of temporal data mining. Sadhana 31(2), 173–198 (apr 2006), http://dx.doi.org/10.1007/BF02719780 http://link.springer.com/10.1007/BF02719780
46. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. pp. 2–11. ACM, New York, NY, USA (jun 2003), https://dl.acm.org/doi/10.1145/882082.882086
47. Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing SAX: a novel symbolic representation of time series. Data Mining and Knowledge Discovery 15(2), 107–144 (2007), http://dx.doi.org/10.1007/s10618-007-0064-z
48. Macleod, J., Luk, A., Titterington, D.: A Re-Examination of the Distance-Weighted k-Nearest Neighbor Classification Rule. IEEE Transactions on Systems, Man, and Cybernetics 17(4), 689–696 (jul 1987), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4075685
49. Marteau, P.F.: Time Warp Edit Distance with Stiffness Adjustment for Time Series Matching. IEEE Transactions on Pattern Analysis and Machine Intelligence 31(2), 306–318 (feb 2009), https://ieeexplore.ieee.org/document/4479483

50. Mitchell, T.M.: Machine Learning. McGraw-Hill, Inc., New York, NY, USA (1997)
51. Mitsa, T.: Temporal Data Mining. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, Taylor & Francis (2010), http://books.google.rs/books?id=4P_7ydvW7cAC
52. Pao, T.L., Chen, Y.T., Yeh, J.H., Cheng, Y.M., Lin, Y.Y.: A Comparative Study of Different Weighting Schemes on KNN-Based Emotion Recognition in Mandarin Speech. In: Huang, D.S., Heutte, L., Loog, M. (eds.) Advanced Intelligent Computing Theories and Applications. With Aspects of Theoretical and Methodological Issues, Lecture Notes in Computer Science, vol. 4681, pp. 997–1005. Springer Berlin Heidelberg, Berlin, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-74171-8_101 http://link.springer.com/10.1007/978-3-540-74171-8_101
53. Paparrizos, J.: 2018 UCR Time-Series Archive: Backward Compatibility, Missing Values, and Varying Lengths (2019), https://github.com/johnpaparrizos/UCRArchiveFixes
54. Rafiei, D.: On similarity-based queries for time series data. In: Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337). pp. 410–417. IEEE (1999), http://ieeexplore.ieee.org/document/754957/
55. Rafiei, D., Mendelzon, A.: Similarity-based queries for time series data. ACM SIGMOD Record 26(2), 13–25 (jun 1997), https://dl.acm.org/doi/10.1145/253262.253264
56. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics, Speech, and Signal Processing 26(1), 43–49 (feb 1978), http://ieeexplore.ieee.org/document/1163055/
57. Savaglio, C., Ganzha, M., Paprzycki, M., Bădică, C., Ivanović, M., Fortino, G.: Agent-based Internet of Things: State-of-the-art and research challenges. Future Generation Computer Systems 102, 1038–1053 (jan 2020), https://linkinghub.elsevier.com/retrieve/pii/S0167739X19312282
58. Savic, M., Kurbalija, V., Ilic, M., Ivanovic, M., Jakovetic, D., Valachis, A., Autexier, S., Rust, J., Kosmidis, T.: The application of machine learning techniques in prediction of quality of life features for cancer patients. Computer Science and Information Systems 20(1), 381–404 (2023), https://doiserbia.nb.rs/Article.aspx?ID=1820-02142200061S
59. Sherrington, M.: Mastering Julia - Second Edition: Enhance your analytical and programming skills for data modeling and processing with Julia. Packt Publishing, USA (2024)
60. Tan, P.N., Steinbach, M., Kumar, V., Karpatne, A.: Introduction to Data Mining. Pearson Education, 2 edn. (2019)
61. Vlachos, M., Kollios, G., Gunopulos, D.: Discovering similar multidimensional trajectories. In: Proceedings 18th International Conference on Data Engineering. pp. 673–684. IEEE Comput. Soc (2002), http://ieeexplore.ieee.org/document/994784/
62. Vlachos, M., Lin, J., Keogh, E.J., Gunopulos, D.: A Wavelet-Based Anytime Algorithm for K-Means Clustering of Time Series. In: Workshop on Clustering High Dimensionality Data and Its Applications, at the 3rd SIAM International Conference on Data Mining. San Francisco, CA, USA (2003), https://api.semanticscholar.org/CorpusID:18338443
63. Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E.: Experimental comparison of representation methods and distance measures for time series data. Data Mining and Knowledge Discovery 26(2), 275–309 (mar 2013), http://link.springer.com/10.1007/s10618-012-0250-5
64. Witten, I.H., Frank, E., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edn. (2011)
65. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, 4 edn. (2017)
66. Yi, B.K., Jagadish, H.V., Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In: Data Engineering, 1998. Proceedings., 14th International Conference on. pp. 201–208 (1998)

67. Yi, B.K., Faloutsos, C.: Fast Time Sequence Indexing for Arbitrary Lp Norms. In: Proceedings of the 26th International Conference on Very Large Data Bases. pp. 385–394. VLDB '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
68. Zavrel, J.: An Empirical Re-Examination of Weighted Voting for k-NN. In: Proceedings of the 7th Belgian-Dutch Conference on Machine Learning. pp. 139–148 (1997)
69. Zelle, J.: Python Programming: An Introduction to Computer Science 2nd Edition. Franklin, Beedle & Associates Inc., USA (2010)

Zoltán Gellér is an Associate Professor at the Department of Media Studies, Faculty of Philosophy, University of Novi Sad, Serbia. He authored or co-authored 2 textbooks, 1 international monograph, and over 30 publications in data mining, machine learning, computer literacy, and related fields. He was a member of Program Committees of several international conferences and a reviewer in several international journals.

Vladimir Kurbalija holds the position of Full Professor from 2021 at the Department of Mathematics and Informatics, Faculty of Sciences, University of Novi Sad, Serbia, where he received his B.Sc., M.Sc. and Ph.D. degrees. He was/is a member of several international projects supported by DAAD, TEMPUS, Horizon and bilateral and national programs. Vladimir (co)authored over 60 papers in Case-Based Reasoning, Time-Series Analysis, Medical decision-support systems, and related fields. He was a member of Program Committees of numerous international conferences, and a reviewer in more than 30 international journals.

Mirjana Ivanović is a Full Professor at the Faculty of Sciences, University of Novi Sad, Serbia, since 2002, and a corresponding member of the Serbian Academy of Sciences and Arts since 2024. She is a member of the Board of Directors of the Institute for Artificial Intelligence Research and Development of Serbia. Mirjana has authored or co-authored 17 textbooks, 30 edited proceedings, 4 monographs, and more than 540 research articles on multi-agent systems, e-learning and web-based learning, applications of intelligent techniques (CBR, data and web mining), software engineering education, most of which are published in international journals and proceedings of high-quality international conferences. She has served as a member of program committees for more than 500 international conferences and has chaired numerous international conferences as general chair and program committee chair. Additionally, she has been an invited speaker at numerous international conferences and a visiting lecturer in Australia, Thailand, and China. As a leader and researcher, she has participated in highly regarded international projects.

Received: September 10, 2024; Accepted: January 08, 2025.