A Feature Extraction Framework for Time Series Analysis

An Application for EEG Signal Processing for Epileptic Seizures Detection

Angelo Martone, Gaetano Zazzaro
Italian Aerospace Research Centre, CIRA
Capua (CE), Italy
e-mail: {a.martone, g.zazzaro}@cira.it

Luigi Pavone
Neuromed, IRCCS
Pozzilli (IS), Italy
e-mail: bioingegneria@neuromed.it

Abstract—**With the raise of smart sensors and of the Internet of Things paradigm, there is an increasing demand for performing Data Mining tasks (classification, clustering, outlier detection, etc.) on data stream produced by these inter-connected devices. In particular, Data Mining for time series has gained a relevant importance in the last decade. For these temporal data, feature extraction can be performed using various algorithms and decomposition techniques for time series analysis. In addition, features can also be obtained by sequence comparison techniques, such as dynamic time warping or other measures of similarity. For these reasons, we have designed and implemented a multipurpose and extendable tool for window-based feature extraction from time series data. This paper describes the architecture of the designed tool, named Training Builder, and the current version of its multi-language implementation, which focuses on time series feature extraction, parametric windowing task and data pre-processing. The framework has been applied in the neurological domain where very good results have been achieved for epileptic seizures detection; the case study shows how the Training Builder tool may be very helpful for the next Data Mining tasks.**

Keywords-data mining; time series analysis; feature extraction; sliding window; similarity measures; pre-processing.

I. INTRODUCTION

In many application fields, such as production lines in factories or stock quotes analysis, it is quite usual to create and process high amounts of data at high rates. Such continuous data flows with unknown size and end are called data streams [1]. When elements of a data stream have a temporal ordering, we talk about time series data. Today, the primary source of data streams are smart sensors that are ubiquitous devices crucial for a multitude of monitoring applications. Important examples are weather observation and environment monitoring in general, health monitoring, Radio-Frequency IDentification (RFID) monitoring, or road monitoring. There are several important tasks that have to be considered when dealing with time series data; among these are: signal pre-processing transformation, time-based windowing and feature extraction process. All these different tasks are usually separately implemented in the freely available tools, see Section II, and it is often hard to combine them to achieve the desired workflow. Being motivated by this observation, we have designed and developed a multipurpose and extensible tool called Training Builder, with the aim of supporting Data Mining process on time series data, implementing data representations, similarity measures and pre-processing modules. It also makes possible to easily change some existing or to add new concrete implementation of any module or algorithm. We have implemented many features and similarity measures, and we have performed a set of experiments to validate their advantages.

In Section II, we examine some time series data analysis tools that exist in the literature. In Section III, we present time series analysis general outlines, including main definitions, its scope and its role in Data Mining (DM). In Section IV, the Training Builder Tool is presented, including the main definitions, feature extraction process, and Graphical User Interface (GUI). In Section V, we show how the application has been applied to a case study in neurological domain. Finally, in Section VI, our general considerations and future works are shown.

II. RELATED WORK

Many tools and applications deal with time series data, each of which differs by the type of approach.

There is a category of tools specialized in the implementation of DM algorithms, such as Waikato Environment for Knowledge Analysis (WEKA) [2] and RapidMiner [3]. WEKA tool supports a great number of DM and machine learning techniques, including data pre-processing, classification, regression and visualization. However, WEKA is a general-purpose DM library, not specialised for time series. Instead, the time series support within WEKA is based on Massive Online Analysis (MOA) [3] tool, which is an open source framework for data stream mining, with a very active growing community. It includes a collection of machine learning algorithms (classification, regression, clustering, outlier detection, etc.) and tools for evaluation. Another system similar to WEKA is RapidMiner. It is also an open source (only the Community Edition) collection of data-mining and machine-learning techniques. RapidMiner has a very sophisticated graphical user interface, and it is also extensible with the user's implementations. Time series support is demanded to the Time Series Extension package that is in alpha version at this moment.

In addition, there are several tools specialised for monitoring and visualisation of time series. Kibana [5] is an open source analytics and visualization platform designed to work with Elasticsearch [6]. It enables near real-time analysis and visualization of streaming data. It allows interactive data exploration, supports cross filtering and provides multiple chart types, such as bar chart, line and scatter plots, histograms, pie charts, and maps. It is open source and has a number of plug-in extensions that can further enhance its functionality. Grafana [6] is another open source visualization tool that can be used on the top of a variety of different data stores, especially on the top of Time Series DataBases (TSDB), such as OpenTSDB [8] and KairosDB [9].

Lastly, there are several massive data stream processing frameworks specialized in manipulation of time series data produced at high rate and with a large volume, such as Apache Spark [10] and Apache Flink [11]. Apache Spark is a batch-processing framework with stream processing capabilities. Built using many of the same principles of Hadoop's MapReduce engine, Spark focuses primarily on speeding up batch processing workloads by offering full in-memory computation and processing optimization. Spark has a specific module, Spark Streaming, which supplies stream processing capabilities, making use of the so-called micro-batches. Apache Flink is a stream processing framework that can also handle batch tasks. It considers batches as data streams with finite boundaries, and thus treats batch processing as a subset of stream processing. This stream-first approach has been called the Kappa architecture [12], in contrast to the more widely known Lambda architecture [13].

Currently, however, there is no freely available standalone system or framework that, at the same time, provides efficient implementations of features extraction process and data pre-processing techniques for time series data and supports the necessary concepts of data representation, similarity measures and signal filtering tasks. In this paper, we propose a software application that tries to combine all the different aforementioned approaches (algorithms, visualization, storage, and parallel/distributed computation) in order to facilitate the user in the DM step.

The implemented tool, called Training Builder, covers all the data preparation tasks, ranging from the signal pre-processing step to the data labeling one, by using the sliding window paradigm and the features calculation algorithms, enriched by functionalities of data storage and data visualization. Training Builder has also been developed to be as extensible as possible, allowing to easily add algorithms for feature calculation to the existing core implementation, thanks to an extremely flexible and modular architecture, and the algorithms can be developed with different programming languages. Lastly, a user-friendly and Web-oriented GUI allows the user to select the temporal parameters and the features to be extracted from the input time series and, showing the charts of features over time, it allows to quickly evaluate and optimize the temporal parameters by mutual comparisons.

III. TIME SERIES ANALYSIS & MINING

Time series analysis is composed of methods that attempt to extract meaningful statistics and other characteristics from data points, to understand the underlying context, and to make forecasts. Time series data are popular in many applications, such as stock market analysis, process control, observation of natural phenomena, scientific and engineering experiments, medical treatments, etc. Therefore, in the last decade, there has been increased interest in querying and mining such data. The purpose of time series mining is to try to extract all meaningful knowledge from the shape of the data. Even if humans have a natural capacity to perform these tasks, it remains a complex problem for computers. In recent years, many efforts have been made to find new methodologies for different time series mining [14] task types including indexing, classification, clustering, prediction, segmentation, anomaly detection, motif discovery, etc.

There are several important concepts that should be taken into account when dealing with time series: pre-processing transformation, time-based parametric windowing, feature extraction, and visualization.

A. Pre-Processing Transformation

"Raw" time series usually contain some distortions, which could be consequences of bad measurements or just a property of the underlying process that generated the time series. The presence of distortions can seriously deteriorate the indexing problem because the distance between two "raw" time series could be very large even though their overall shape is very similar.

The task of the pre-processing transformations is to remove different kinds of distortions. Some of the most common pre-processing tasks are: offset translation, amplitude scaling, removing linear trend, removing noise, etc. [15].

Pre-processing transformations can greatly improve the performance of time series applications by removing different kinds of distortions.

B. Parametric Windowing Technique

Windowing is one of the most frequently used processing methods for data streams. An unbounded stream of data (events) is split into finite sets, or windows, based on specified criteria, such as time. A window can be conceptualized as an in-memory table in which events are added and removed based on a set of policies.

This subsection describes how sliding and tumbling windows work. Both types of windows move across continuous streaming data, splitting the data into finite sets. Finite windows are helpful for operations, such as aggregations, joins, feature extraction, and pattern matching.

1) Tumbling Window

In a tumbling window, tuples are grouped in a single window based on time or count. A tuple belongs to only one window.

For example, consider a time-based tumbling window like the one shown in Fig. 1 with a length of five seconds. The first window (*w1*) contains events that arrived between the zeroth and fifth seconds. The second window (*w2*) contains events that arrived between the fifth and tenth seconds, the third window (*w3*) contains events that arrived between tenth and fifteenth seconds, and finally the fourth window (*w4*) contains events that arrived between fifteenth and twentieth seconds. The tumbling window is evaluated every five seconds, with no overlap between different time windows; each segment represents a distinct time segment.

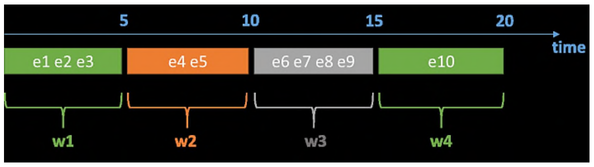


Figure 1. A tumbling windowing process.

This method can be applied, for example, for the computation of the average of a price of a stock over the last five minutes, repeated every five minutes.

2) *Sliding Window*

In a sliding window, tuples are packed within a window that moves across the stream of data according to a fixed interval. A time-based sliding window with a length of x seconds and a sliding interval of y seconds contains tuples that arrive within an x-second window. The tuples within the window are evaluated every y seconds. Sliding windows can contain overlapping data and the same event can belong to more than one sliding window.

An example is shown in Fig. 2. The first window (*w1*, the green box) contains events occurring between the zeroth and tenth seconds. The second window (*w2*, the orange box) contains events between the fifth and fifteenth seconds. Note that events *e4* through *e5* are in both windows. When window w2 is evaluated at time *t* = 15 seconds, events *e1*, *e2*, and *e3* are dropped from the event queue.

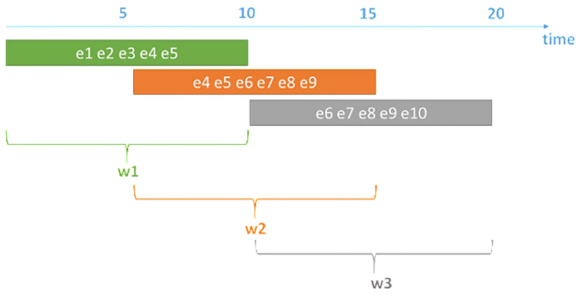


Figure 2. A sliding windowing process.

The time windows w1, w2, w3 contain overlapping data.

C. *Features Extraction Task*

Feature extraction aims to explain the underlying phenomena of interest from a set of raw data by simplifying the amount of resources required to accurately describe it. In various fields, such as image processing or bio-informatics, raw data are corrupted with undesired variations, or noise, that should be discarded. Thus, feature extraction methods usually consist of a combination of noise removal algorithms (also called de-noising), structure detection, and dimensionality reduction techniques. Generally, an optimal balance is required to be found between fineness and complexity of the extracted features. The desired output should use a minimal amount of resources while being able to accurately describe the underlying phenomena of interest of the data. Once the relevant part of the signal has been extracted, detailed analysis may be conducted, hypotheses may be drawn, and further applications may be considered by the end-user.

Features could be extracted either from one signal (univariate) or from two or more signals (multivariate). In particular, bivariate features are based on a similarity measure that compares two time series objects and returns a value that encodes how similar they are. Distance metrics represent a kind of similarity measures commonly used to define if two time series are similar. Many algorithms are used to compute these metrics, such as Lp distance (Lp) [16], Dynamic Time Warping (DTW) [17], distance based on Longest Common Subsequence (LCSS) [18], Edit Distance (ED) [19], also known as Levenshtein Distance, etc.

D. *Data Visualization*

Data visualization is a general term that describes any effort to help people understand the significance of data by placing it in a visual context. A graphical visualization of time series could help Data Analyst to better understand time series evolution and to find patterns, correlations or trends.

Usually, a time series is represented by a line graph or a stacked area chart, where the observations are plotted against the corresponding sampling time. A line graph is the simplest way to represent time series data and it uses points connected by lines (also called trend lines). For temporal time series, it represents how the signal changes across time, so how the dependent variable (the signal) changes according to the independent variable (the time).

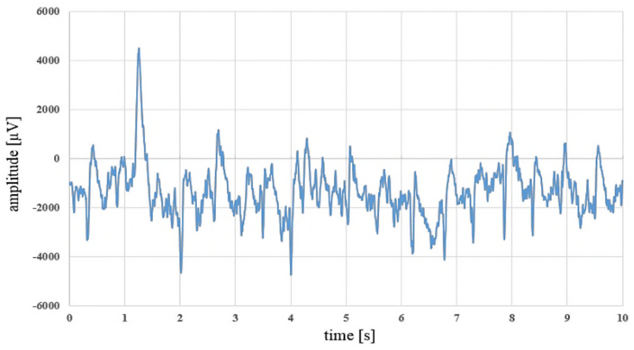


Figure 3. A line graph chart reporting ten seconds of an EEG recording.

The graph in Fig. 3 shows the first ten seconds of an ElectroEncephaloGram (EEG) that measures human brain's electrical activity; along the y-axis is plotted the amplitude of the measured signal in microvolt, while the x-axis has the time in seconds.

IV. TRAINING BUILDER TOOL

The Training Builder is a modular software application for the massive extraction of features from time series, provided as input, by changing of the temporal analysis parameters and the band-pass filters.

The final output of the tool is to create the training sets that will be used as input for the DM techniques. Therefore, each set of training varies depending on:

- Time series (or better the recording of them).
- Temporal analysis parameters: L, R, and S.
- Band-pass filters: [8][12], [13][20], etc.
- Features to be computed (Hjorth Parameters, Statistical Moments, etc.).
- Bivariate calculation method: bivariate algorithms can be used to compute similarity distance between the under examination signal and a "reference" signal.

Each training set consists of a comma-separated values (csv) file, where features are recorded as vectors.

A. Software Architecture and GUI

The software application architecture has been designed following the Client / Server architectural model, in which the Server part is composed of the algorithm for massively extracting features, pre-processing functions, and other support utilities, while the Client part is composed of a browser-based application, responsible for visualizing output results and submitting a form for input selection and validation.

Fig. 4 shows the high-level diagram of the designed software architecture, including the input data sources and the outputs delivered; accordingly, two possible time series data sources are provided:

- Recorded in text format (txt or csv).
- Stored in a TSDB (OpenTSDB or KairosDB).

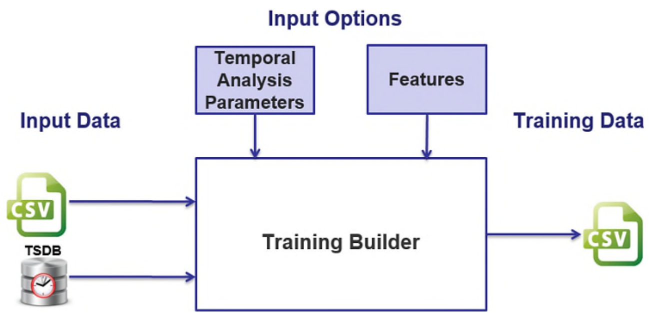
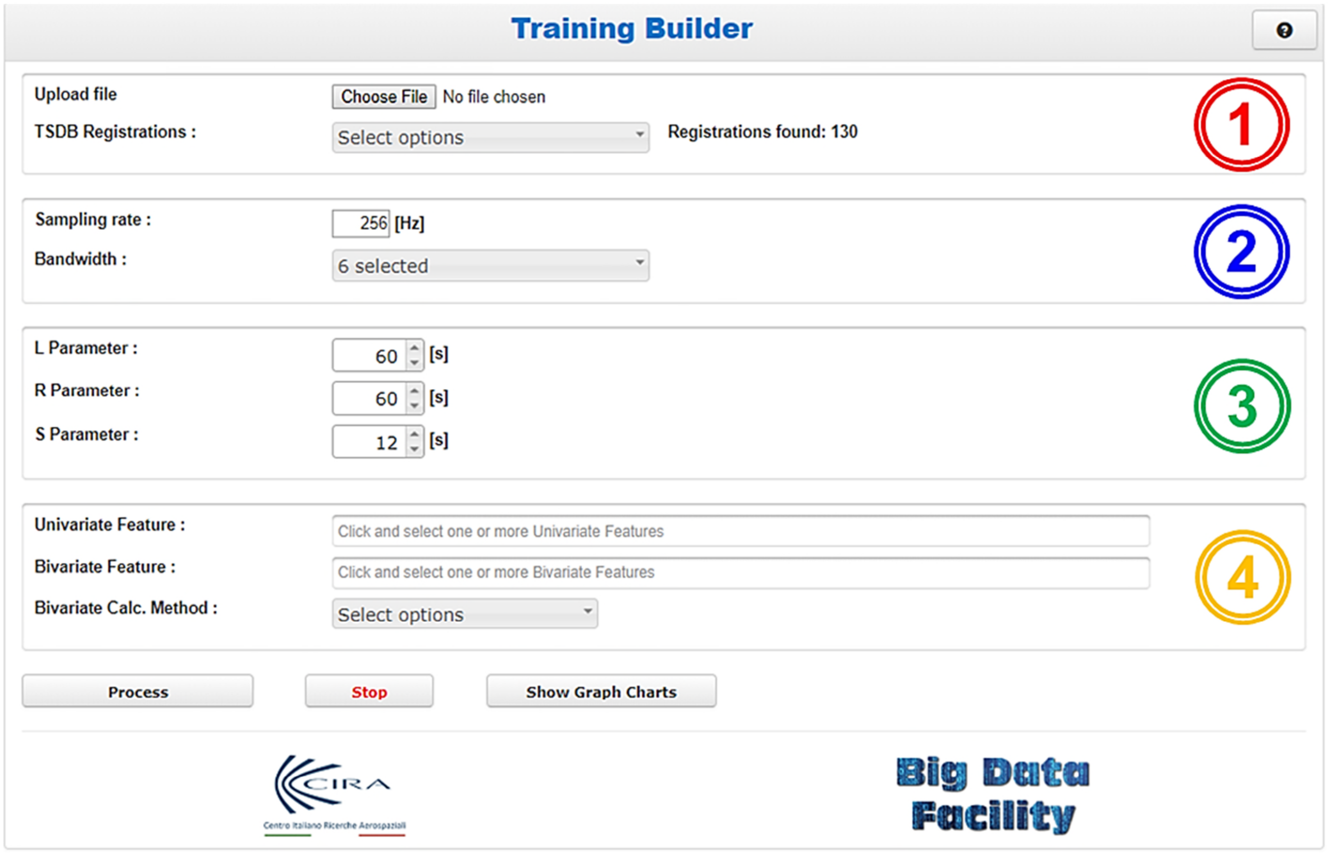


Figure 4. Application logic scheme.

The use of a time series database, instead of formatted files, allows an optimization in the management of time series, as regards their storage and recovery, while ensuring high reliability and availability.

Currently, the application can store and retrieve time series data from the OpenTSDB and KairosDB time series databases, which in turn store data on the NoSQL databases Apache HBase [21] and Apache Cassandra [22], respectively. This double possibility allows the Training Builder to adapt to different software configurations.

In output, instead, the results of the application of features to these time series are provided in csv format. The csv file can be saved by the client or stored on a distributed file system.



Upload file: No file chosen
TSDB Registrations : Select options — Registrations found: 130

Sampling rate : 256 [Hz]
Bandwidth : 6 selected

L Parameter : 60 [s]
R Parameter : 60 [s]
S Parameter : 12 [s]

Univariate Feature : Click and select one or more Univariate Features
Bivariate Feature : Click and select one or more Bivariate Features
Bivariate Calc. Method : Select options

Figure 5. Training Builder GUI.

By using the responsive Web-oriented GUI, shown in Fig. 5, the input sources, all the temporal parameters, the bandwidths, and the features can be chosen and selected by the user. A direct interface to the time series visualization browser, provided by the above-mentioned TSDB, is also provided.

The GUI can be divided into four functional blocks, as highlighted by numbered circles in Fig. 5. The first block enables user to select the time series that has to be analysed, selecting a local csv file or choosing a stored time series in the TSDB. In the second block, the user can specify the time series sampling rate and can select the bandwidth intervals. In the third block, the temporal analysis parameters are listed (see Section IV.D). In the fourth block, the user can select the univariate and bivariate features to be computed from the selected time series data (see Section IV.E) and the calculation methods useful to extract the bivariate ones (see Section IV.F for deeper details). Lastly, a set of buttons allows the following operations:

- Process: starts data analysis process.
- Stop: stops the running computation.
- Show Graph Charts: shows the plot charts of the raw data and filtered time series (an example is reported in Fig. 7) in a separate browser window.

This client component was developed using the jQuery JavaScript library [20]; in particular, it was used to manipulate the Document Object Model (DOM) interface of the Hyper Text Markup Language (HTML) page and for asynchronous communications with the Server part, by using Asynchronous JavaScript and XML (AJAX) technology.

The Server component is instead divided into two layers:

- The Application Layer, which includes the logic that implements the analysis algorithms and how they are used for the massive extraction of the features, according to the chosen temporal parameters and other user selected inputs.
- The Data Layer, which is represented by the time series database chosen (OpenTSDB and KairosDB) or text/csv files.

Each of these layers could be instantiated on a dedicated workstation improving the overall performances of the software application.

Furthermore, Training Builder has been developed to be as extensible as possible, with the aim of being able to execute algorithms for feature computation developed with different programming languages; currently Java, C/C++ and Matlab are natively supported, but compatibility with other languages like R and Python, which are widely used for time series analysis tasks, can be easily configured. This capability is achieved thanks to Java Native Interface (JNI) and Service Provider Interface technologies (SPI) offered by Java Virtual Machine (JVM).

B. *Core System Funcionalities*

The core of the system consists of a set of algorithms for features implementation and routines for the definition of temporal analysis parameters. The choice of which parameters and which features to apply to the input files is delegated to the user and is simplified through a Web-oriented graphical user interface. The application is also compliant with the architectural pattern Representational State Transfer (REST) [23]: using a stateless protocol and standard operations, REST systems provide high performance, reliability, and scalability, reusing components that can be managed and updated without affecting the system as a whole, even while it is running. The REST APIs, which act as wrappers for the developed algorithms, can be called as services from external applications; in this way, for example, the application can be integrated into existing software platforms or recalled by other remote Web services. The Web application can be run on any standard Java Application Server; for our tests Apache Tomcat [24] has been used, because it is the most widespread and used in the Open Source community.

The algorithmic component (that is, the component that contains the features algorithms) has been coded using Java, Matlab and C programming languages. The choice of the language to be used is related to the complexity of the algorithm (for example, in Matlab it is much easier to work on matrices and vectors) and the availability of built-in functions that can simplify the coding of the algorithm itself. Consider, for example, the Log-Energy Entropy feature, which requires the wavelet transform of the input signal: its implementation is easier in Matlab environment as it provides a series of utility functions to obtain the wavelet transform of a signal (both discrete and continuous).

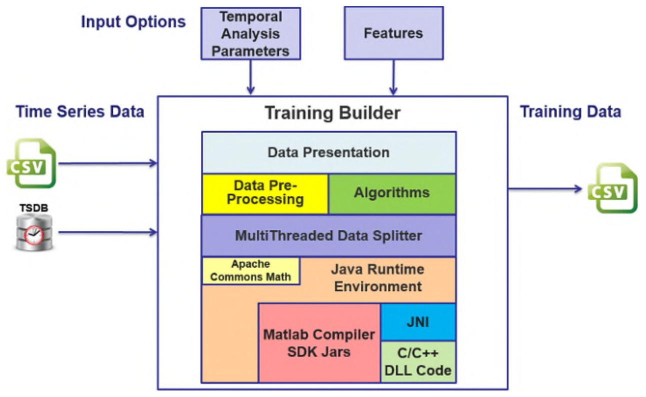


Figure 6. Application components scheme.

In order to execute algorithms implemented in Matlab environment, a series of utility classes were developed in Java (using the Matlab Compiler SDK tool), which are able to interoperate with the Matlab environment through the Matlab Runtime [25]. The Matlab Runtime is a standalone set of shared libraries that allows running applications compiled by Matlab or Matlab components on computers where Matlab is not installed. The Matlab Runtime behaves very similar to a JVM, allowing code portability on Windows, Linux and Mac machines. Where possible, the equivalent C code of the algorithm developed in Matlab was generated automatically using the Matlab Coder toolbox. This eliminates the dependency on the Matlab Runtime, but the corresponding library must be generated according to the host Operating System (DLL for Windows, Shared Object for Linux). Unfortunately, this operation is not always supported; in fact, in the case of Log-Energy Entropy this was not possible because the wavelet functions cannot be generated in C, so the code requires the Matlab Runtime to be installed on the host machine. The Multithreaded Data Splitter component, shown in purple in Fig. 6, is responsible of splitting the dataset to be analysed, into sub-blocks of data, in order to exploit the multithreading capabilities of modern processors, parallelizing the execution of the program and consequently increasing its performance. The input to the component can be one or more files to be analysed or even one or more streams retrieved from the time series database; both types of input are converted into a standard internal format so, for simplicity, we will now use the generic term source to indicate one of the two input types. The component implements an algorithm to define the number of threads to be used and then calculates how to split the data provided by the source between the different threads. Indicating with *MaxNumThread* the number of threads manageable by the processor and *NumSources* the number of sources to be analysed, the algorithm follows these steps:

1. One thread is reserved for each source to be analysed (at most equal to MaxNumThread).
2. Each of the threads of the ith source can in turn launch a number of secondary threads equal to:

$$\text{NumThread}_i = \text{MaxNumThread} / \text{NumSources} \qquad (1)$$

3. For each jth thread of the ith source, a block of data is assigned equal to:

$$\text{DataBlock}_{ji} = \text{SourceDataLength} / \text{NumThread}_i \qquad (2)$$

For example, suppose you have an Intel^{TM} processor with 8 cores and each core can handle 2 threads using Hyper-Threading technology, obtaining a total number of 16 threads that can be managed simultaneously (*MaxNumThread*); if we wanted to process 4 sources in parallel (*NumSources*) of data length equal to 10000 values, we would have a thread for each single source (1). Each source is then associated with a number of threads ($NumThread_i$) equal to 16/4 = 4. Each of these four threads is assigned a portion of data equal to 10000/4 = 2500 values (2).

The Java implementation of this component makes use of the concurrency APIs, where the Executor framework as a layer of higher level in thread management has been implemented. Executors replace the direct execution mode of threads, allowing the implementation of asynchronous tasks and thread pools. Each thread inside the pool is reusable: an Executor does not autonomously terminate its execution but waits for the execution of new tasks. In our tests, we have seen an almost linear performance speedup, by increasing the number of threads used.

C. Signal Pre-Processing

Data pre-processing is made up of a set of techniques able to transform the raw data into some meaningful and understandable format. It is advisable to identify the main frequency components of a signal in order to eliminate the so-called out-of-band noise. The choice of the frequency bands to be used for processing the signals under examination also depends on the type of the signal and on the frequency at which it was acquired. The upper limit is dictated by the sampling frequency (as stated by the Nyquist–Shannon sampling theorem). The bandpass filters, encapsulated in the yellow block of Fig. 6, have been implemented in the Matlab environment by using the Fast Fourier Transform (FFT) function.

D. Parametric Windowing

Parametric Windowing in Training Builder is achieved by using three temporal analysis parameters:

- *L*: it represents the length of the signal to be analysed, expressed in seconds [s].
- *S*: it represents the slippage of the signal to be analysed (i.e., how often the algorithm is applied), expressed in seconds [s].
- *R*: it represents the forecast radius, expressed in seconds [s].

If the sliding step size *S* is smaller than the window size *L*, the windows overlap, while if $S = L$ we get a tumbling window. *R* parameter is helpful to tag each computed feature with a target class (this is helpful for the next DM).

E. Implemented Features

The Algorithms green block, in Fig. 6, is the component responsible of features algorithms computation.

TABLE I. COMPUTED FEATURES ALGORITHMS

Id	*Feature Name*	*Code*	*UB*	*Coding*
1	Mean	SM1	U	Java
2	Standard Deviation	SM2	U	Java
3	Variance	SM3	U	Java
4	Skewness	SM4	U	Java
5	Kurtosis	SM5	U	Java
6	Hjorth Mobility	HP1	U	Java
7	Hjorth Complexity	HP2	U	Java
8	Shannon Entropy	EB1	U	Java
9	Log-Energy Entropy	EB2	U	Matlab
10	Kolmogorov Complexity	CB1	U	Matlab/C
11	Upper Limit Lempel-Ziv Complexity	CB2	U	Matlab/C
12	Lower Limit Lempel-Ziv Complexity	CB3	U	Matlab/C
13	Peak Displacement	SE1	U	Java
14	Predominant Period	SE2	U	Java
15	Averaged Period	SE3	U	Java
16	Squared Grade	SE4	U	Java
17	Squared Time to Peak	SE5	U	Java
18	Inverted Time to Peak	SE6	U	Java
19	Conditional Entropy	MC1	B	Java
20	Joint Entropy	MC2	B	Java
21	Mutual Information	MC3	B	Java
22	Cross Correlation Index	MC4	B	Java
23	Euclidean Distance	DB1	B	Java
24	Levenshtein Distance	DB2	B	Java
25	Dynamic Time Warping	DB3	B	Java
26	Longest Common Sub-Sequence	DB4	B	Java

Currently, 26 algorithms have been implemented, that could be divided into 7 classes and can be of Univariate (U) or Bivariate type (B):

- SM: Statistical Moments.
- HP: Hjorth Parameters.
- EB: Entropy Based.
- CB: Complexity Based.
- SE: Seismic Evaluators.
- MC: Mutual Conditioned.
- DB: Distance Based.

In Table I, a list of all implemented features is reported, and it is also specified with which programming language the algorithm has been coded.

A description of the more relevant implemented features is reported below.

1) Statistical Moments

In mathematics, a moment is a specific quantitative measure of the shape of a function. In our framework, the first four statistical moments have been calculated, plus standard deviation measure. All algorithms where developed in Java by using the Apache Commons Math library [26].

2) Hjorth's parameters

Hjorth's parameters (normalized slope descriptors) of mobility and complexity [27] quantify the root-mean-square frequency and the root-mean-square frequency spread of a given signal, respectively.

3) Shannon Entropy

In Information Theory, the Shannon's Entropy represents the average amount of information produced by a stochastic source of data. Formally, it is defined as the expected value of self-information. The latter represents the information contained in a given event x, emitted by the source X and it is defined as follows:

$$I(x) = -\log_2 P(x) \tag{3}$$

Thus, the entropy of a source X turns out to be:

$$H(X) = E[I(X)] = E[-\log_2 P(x)] \tag{4}$$

where $P(X)$ is a probability mass function for a discrete random variable X.

4) Log-Energy Entropy

The Log-Energy Entropy is a feature closely related to Shannon's Entropy and to Wavelet Transform. In fact, after an appropriate wavelet decomposition, it is possible to calculate the Log-Energy Entropy by using the following relation:

$$E(s) = \sum_{i=1} \log_2 (s_i^2) \tag{5}$$

where s_i are the N coefficients of the wavelet transform for the signal s emitted.

5) Kolmogorov Complexity

In Algorithmic Information Theory, the Kolmogorov Complexity of an object, such as a piece of text, is the length of the shortest computer program (in a predetermined programming language) that produces the object as output. It is a measure of the computational resources needed to specify the object and it is also known as descriptive complexity.

6) Lempel-Ziv Complexity

The Lempel-Ziv Complexity of a given finite binary sequence is an index associated with the number of sub-sequences that can be identified. In particular, this process can take place through methods that tend to highlight the greater or lesser complexity of the given sequence. Therefore, taking into account the two extremes, it is possible to calculate those that are interpreted as the upper and lower limit of this index.

7) Seismic Evaluators

The seismic evaluators have been calculated by considering [28] and [29] because there is an analogy between earthquakes and epileptic seizures.

8) Dynamic Time Warping

Dynamic Time Warping is a technique that uses dynamic programming to compare two sequences of different lengths and allows non-linear alignments, one-to-many, or vice versa, thanks to a temporal distortion. A nonlinear (elastic) alignment produces a more intuitive measure of similarity and favours those cases in which the sequences are similar but locally out of phase.

F. Bivariate Features Calculation Methods

Bivariate algorithms have been used to compute similarity distance between the under examination signal and a "reference" signal. This reference signal could be of three different types:

- W.r.t. Previous L: with respect to the same signal taken at a previous L interval.
- W.r.t. Zero: with respect to the zero constant signal.
- W.r.t. Different Synchronous Signal: with respect to a synchronous signal happening in the same instant but originated from a different positioning.

V. Case Study in Neurological Domain

Epilepsy is a neurological disorder characterized by recurrent seizures caused by abnormal electrical discharges from the brain cells, which extremely affect patient quality of life. The worldwide recognized standard for epilepsy monitoring and diagnosing is ElectroEncephaloGram (EEG) recorded from scalp or intracranially (iEEG). The former is the most commonly used ambulatory method, mainly due to its low invasiveness, while the latter is mainly used to help patients in which classical EEG monitoring is not able to identify epileptic area. A lot of interest there is in finding automated seizure detection methods from EEG/iEEG, to help clinicians to identify seizures on EEG/iEEG recordings and also to embed them in closed-loop systems for epilepsy control. Feature extraction method for epileptic EEG/iEEG plays a crucial role in detection algorithms, since it seriously affects the performance of these algorithms.

A. *Training Builder on Working*

We used Training Builder to analyse iEEG signals for the detection of epileptic seizures. In particular, by analysing the fraction of the iEEG recordings immediately preceding the beginning of the epileptic seizure (PreIctal recordings) and the ones belonging to the seizure itself (Ictal recordings), a classifier is trained in order to determine the anomalous signals, using the numerous computed predictive features.

A public iEEG dataset, the Freiburg Seizure Prediction EEG database (FSPEEG) [30], was used for evaluating the classification performance of extracted features. The database contains iEEG recordings from 21 patients with medically intractable epilepsy. Recordings were made by means of grids, strips, and depth electrodes, and acquired with a 128 channel system at 256 Hz sampling rate. Six iEEG channels were selected by certified epileptologists, three from focal electrodes (InFokus channels), located near to the region where the seizures occurred and three from extra focal electrodes (OutFokus channels), located in areas far from the seizure focus. In our test, we examined twelve recordings of one patient (number 16): five containing seizures (Ictal) and seven without seizures (PreIctal), were observed.

In order to detect the beginning of the epileptic seizure within the iEEG signals, a binary classifier can be trained starting from the training set formed by the computed features and whose target class is *ActualYN*, which assumes the values {*YES*, *NO*}: *YES* if we are in Ictal phase, *NO* otherwise.

B. *iEEG Pre-Processing*

The iEEG signal from each InFokus/OutFokus electrode was filtered through six different frequency bands, 8-12 Hz, 13-20 Hz, 21-30 Hz, 30-45 Hz, 40-70 Hz and 70-120 Hz using band-pass filter, thus obtaining six signals. The upper limit of 120 Hz is dictated by the sampling frequency with which the iEEG signal was acquired at 256 Hz. Moreover, a notch filter at 50 Hz, to minimize power line interferences, has been used.

C. *Feature Extraction Process*

The first step before feature extraction is the selection of the window size *L* and the sliding step *S* for the sliding window calculation task. *R* parameter is used to select the value of the *ActualYN* target class.

TABLE II. TEMPORAL PARAMETER VALUES

L	*R*	*S*
5 [s]	0 [s]	1 [s]

For this case study, we selected temporal parameter values (in seconds) as listed in Table II; with $S < L$ we had choose an overlapped window and we set $R = 0$ because we wanted to detect the onset of the epileptic seizure.

For this case study, we decided to compute all features provided by the Training Builder tool for all possible combination of electrodes, bandwidths and type of reference signal; the size of the final feature dataset is then:

$$(a + b * c + b * d) * e * f \qquad (6)$$

where *a* are the univariate features, *b* the bivariate features, *c* bivariate modality calculation, *d* the type of reference signal, *e* the bandwidths and *f* the electrodes.

TABLE III. FEATURE DATASET VARIABLES VALUE

a	*b*	*c*	*d*	*e*	*f*
18	8	3	3	6	6

In this case study, we have 2376 variables, as reported in Table III.

Training Builder tool tags every extracted vector of features with the corresponding value of the target class *ActualYN.*

D. *iEEG Data Visualization*

Training Builder tool provides also a graphical user interface to visualize and analyse the input time series. This is achieved using the time series visual editors that both OpenTSDB and KairosDB provide. Otherwise, Grafana can be used, which deals well with the two TSDB. In this case study, the GUI made available by KairosDB has been used.

In Fig. 7, the first 30 seconds of the patient's registration 001 have been displayed, recorded from the electrode 1 and filtered in the band [13,20] Hz, for a total of 7680 samples (considering the sampling freq. of 256 Hz).

These charts are also very useful for visually detecting the various seizures phases.

E. *Modeling*

In the classification step, we hypothesized that the different features extracted over time can be separated into two classes corresponding to two different cerebral states (Ictal and PreIctal).

By analysing the fraction of PreIctal and Ictal recordings, a classifier model has been trained in order to determine the anomalous signals, using the calculated features. We chose as classifier a multilayer neural network (Multilayer Perceptron) with 20 hidden layers (H = 20). From our tests, it is able to correctly classify the 99.27% of records, including 95% of records of the *YES* class.

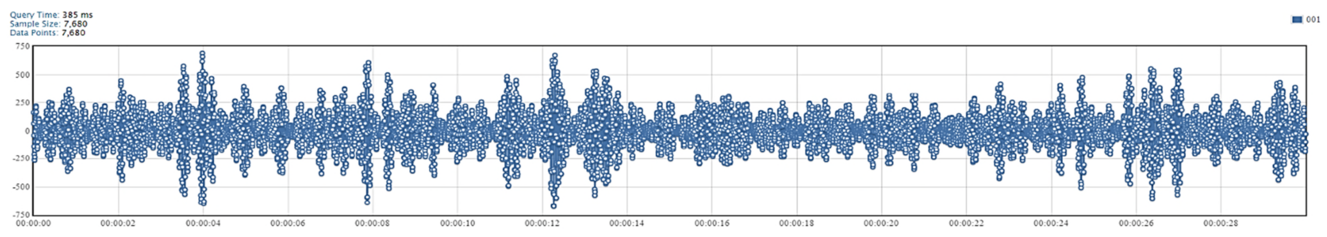


Figure 7. iEEG time series visualization.

To get further details of the Modeling phase of DM process and additional interesting methods and results, you can see [31], where Support Vector Machines have been trained in order to detect epileptic seizures in the iEEG signals.

VI. CONCLUSIONS AND FUTURE WORKS

In this work, we proposed a time-based windowed framework for time series analysis that allows Data Analysts to easily set all different combination of temporal parametric values, bandwidth intervals, and features to extract from a time series. By using a user-friendly software application, we tested a case study in the neurological domain, in order to understand how this approach helps to analyse the dataset, to optimize the feature extraction task and to help the following modelling task of the target dataset, by applying the sliding window paradigm.

As future works, we are going to integrate in our tool some representation techniques that can reduce the dimensionality of time series. These techniques have been proven to limit time and memory consuming, especially when there is a need to compute a similarity distance between time series. Moreover, in the future studies, we are going to use one of the massive data stream processing frameworks, mentioned in Section II.

ACKNOWLEDGMENT

The authors would mention the Big Data Facility project, funded by the Italian Aerospace Research Program (PRORA), in which the tool has been designed and developed.

REFERENCES

[1] S. Geisler, “Data Stream Management Systems,” In: Data Exchange, Information, and Streams, 2013.

[2] M. Hall et al., “The WEKA Data Mining Software: An Update,” SIGKDD Explorations, vol. 11, no 1, pp. 10–18, 2009.

[3] M. Hofmann and R. Klinkenberg, “RapidMiner: Data Mining Use Cases and Business Analytics Applications,” Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, CRC Press, October 25, 2013.

[4] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: Massive Online Analysis,” Journal of Machine Learning Research, vol. 11, pp. 1601–1604, 2010.

[5] *Kibana: Explore, Visualize, Discover Data*. [Online]. Available from: https://www.elastic.co/products/kibana, 2019.01.22.

[6] *Open Source Search & Analytics Elasticsearch*. [Online]. Available from: https://www.elastic.co, 2019.01.22.

[7] *Grafana - The open platform for analytics and monitoring*. [Online]. Available from: https://grafana.com, 2019.01.22.

[8] *OpenTSDB, The Scalable Time Series Database*. [Online]. Available from: http://opentsdb.net, 2019.01.22.

[9] *KairosDB, Fast Time Series Database on Cassandra*. [Online]. Available from: https://kairosdb.github.io, 2019.01.22.

[10] M. Zaharia, “An Architecture for Fast and General Data Processing on Large Clusters,” PhD Dissertation, 2013.

[11] P. Carbone et al., “Apache Flink: Stream and Batch Processing in a Single Engine,” Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 36, no. 4, pp. 28–38, 2015.

[12] J. Kreps, *Questioning the Lambda Architecture*. [Online]. Available from: https://www.oreilly.com/ideas/questioning-the-lambda-architecture, 2019.01.22.

[13] N. Marz and J. Warren, “Big Data: Principles and best practices of scalable realtime data systems,” Manning Publications, 2013.

[14] P. Esling and C. Agon, “Time-series data mining,” ACM Computing Surveys (CSUR), vol. 45, no. 1, 2012.

[15] E. Keogh and M. Pazzani, “Relevance Feedback Retrieval of Time Series Data,” Proc. 22nd Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, pp. 183–190, 1999.

[16] R. Agrawal, C. Faloutsos, and A. N. Swami, “Efficient similarity search in sequence databases,” Proc. 4th Int. Conf. of Foundations of Data Organization and Algorithms, pp. 69–84, 1993.

[17] D. Berndt and J. Clifford, “Using dynamic time warping to find patterns in time series,” Proc. of the 3rd Int. Conf. on Knowledge Discovery and Data Mining, pp 359-370, 1994.

[18] M. Vlachos, D. Gunopulos, and G. Kollios, “Discovering similar multidimensional trajectories,” In: ICDE, pp. 673–684, 2002.

[19] R. A. Wagner and M. J. Fischer, “The string-to-string correction problem,” Journal of the ACM, vol. 21, no. 1, pp. 168–173, 1974.

[20] *jQuery*. [Online]. Available from: https://jquery.com/, 2019.01.22.

[21] *Apache HBase*. [Online]. Available from: https://hbase.apache.org, 2019.01.22.

[22] *Apache Cassandra*. [Online]. Available from: http://cassandra.apache.org, 2019.01.22.

[23] R. T. Fielding and R. N. Taylor, “Principled design of the modern Web architecture”, ACM Transactions on Internet Technology, vol. 2, no. 2, pp. 115–150, 2002.

[24] *Apache Tomcat*. [Online]. Available from: http://tomcat.apache.org, 2019.01.22.

[25] *MATLAB Runtime, Run compiled MATLAB applications or components without installing MATLAB*. [Online]. Available from: https://www.mathworks.com/products/compiler/matlab-runtime.html, 2019.01.22.

[26] *Commons Math: The Apache Commons Mathematics Library*. [Online]. Available from: http://commons.apache.org/proper/commons-math, 2019.01.22.

[27] B. Hjorth, “EEG analysis based on time domain properties,” Electroencephalogr. Clinical Neurophysiology, vol. 29, no. 3, pp. 306–310, 1970.

[28] I. Osorio, H. P. Zaveri, M. G. Frei, and S. Arthurs, “Epilepsy: The Intersection of Neurosciences, Biology, Mathematics, Engineering, and Physics,” in Rationales for Analogy between Earthquakes, Financial Crashes, and Epileptic Seizures, CRC Press, Taylor & Francis Group, 2011.

[29] G. Zazzaro, F. M. Pisano, and G. Romano, “Bayesian Networks for Earthquake Magnitude Classification in a Early Warning System,” International Journal of Environmental, Chemical, Ecological, Geological and Geophysical Engineering, vol. 6, no. 4, 2012.

[30] “The Freiburg Seizure Prediction EEG database,” [Online]. Available from: http://epilepsy.uni-freiburg.de/freiburg-seizure-prediction-project/eeg-database, 2019.01.22.

[31] G. Zazzaro et al., “EEG Signal Analysis for Epileptic Seizures Detection by Applying Data Mining Techniques,” Internet of Things: Engineering Cyber Physical Human Systems, Elsevier, in press.