BEATS: Blocks of Eigenvalues Algorithm for Time Series Segmentation

Aurora González-Vidal, Payam Barnaghi, *Senior Member, IEEE*, and Antonio F. Skarmeta, *Member, IEEE*

Abstract—The massive collection of data via emerging technologies like the Internet of Things (IoT) requires finding optimal ways to reduce the observations in the time series analysis domain. The IoT time series require aggregation methods that can preserve and represent the key characteristics of the data. In this paper, we propose a segmentation algorithm that adapts to unannounced mutations of the data (i.e., data drifts). The algorithm splits the data streams into blocks and groups them in square matrices, computes the Discrete Cosine Transform (DCT), and quantizes them. The key information is contained in the upper-left part of the resulting matrix. We extract this sub-matrix, compute the modulus of its eigenvalues, and remove duplicates. The algorithm, called BEATS, is designed to tackle dynamic IoT streams, whose distribution changes over time. We implement experiments with six datasets combining real, synthetic, real-world data, and data with drifts. Compared to other segmentation methods like Symbolic Aggregate approXimation (SAX), BEATS shows significant improvements. Trying it with classification and clustering algorithms it provides efficient results. BEATS is an effective mechanism to work with dynamic and multi-variate data, making it suitable for IoT data sources. The datasets, code of the algorithm and the analysis results can be accessed publicly at: https://github.com/auroragonzalez/BEATS.

Index Terms—BEATS, SAX, data analytics, data aggregation, segmentation, DCT, smart cities

1 Introduction

Less than 1 percent of the data that are nowadays captured, stored, and managed by means of the Internet of Things (IoT) and Big Data technologies is being analysed [1]. There exist several challenges in the analysis of data such as high dimensionality, high volume, noise, and data drifts. Data provided by IoT sources (sensory devices and sensing mechanisms) are multi-modal and heterogeneous. Since all of the above mentioned features hinder the execution and generalization of the algorithms, many higher-level representations or abstractions of the raw data have been proposed to address these challenges.

In this paper, we attempt to aggregate and represent large volumes of data in efficient and higher-granularity form. The latter is an attempt to create sequences of patterns and data segments that occur in large-scale IoT data streams. The contribution of our approach is to do such representation on-the-fly since usually data treatment has to be done very quickly, adapting to unpredictable changes in the data or even without prior knowledge.

A use case where large and dynamic datasets are present is smart cities. Data aggregation and pattern representation enables us to find underlying patterns, providing further understanding of *the city data*. Big Data analytics, machine learning and statistical techniques are used to predict, classify and extract information that empowers machines with decision-making capabilities.

IoT data is usually related to physical objects and their surrounding environment. Normally, IoT data is collected together with a timestamp. The collection of several points spaced in time, having a temporal order is known as time series data. Time series can be analysed using various techniques such as clustering, classification and regression (as inputs of models) in the fields of data mining, machine learning, signal processing, communication engineering, and statistics.

Our proposed method is based on splitting time series data into blocks. These blocks can be either overlapping or non-overlapping and they represent subsets of the whole data structure. The method synthesizes independently the information that the blocks contain. It reduces the data points while still preserving their fundamental characteristics (loosing as little information as possible). We propose a novel technique using matrix-based data aggregation, Discrete Cosine Transform (DCT) and eigenvalues characterization of the time series data. The algorithm is called Blocks of Eigenvalues Algorithm for Time series Segmentation (BEATS). We compare BEATS with the state-of-the art segmentation and representation algorithms. We also compare and evaluate the approaches in two of the most common machine learning tasks, classification and clustering, by comparing metrics between each of the transformed datasets. We also present a

- *A. González-Vidal and A. F. Skarmeta are with the Department of Information and Communications Engineering, University of Murcia, Murcia 30100, Spain. E-mail: {aurora.gonzalez2, skarmeta}@um.es.*
- *P. Barnaghi is with the Institute for Communication Systems, University of Surrey, Guildford, Surrey GU2 7XH, United Kingdom. E-mail: p.barnaghi@surrey.ac.uk.*

Manuscript received 21 June 2017; revised 21 Jan. 2018; accepted 9 Mar. 2018. Date of publication 0 . 0000; date of current version 0 . 0000.
(Corresponding author: Aurora González-Vidal.)
Recommended for acceptance by E. Terzi.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2018.2817229

use case that is related to smart cities showing the suitability of BEATS for real time data stream analysis. This is shown by explaining how to apply it within a Big Data framework.

The remainder of the paper is organized as follows: Section 2 describes the related work. Section 3 motivates the need of a new approach. Section 4 details the algorithm and briefly explains the mathematical background of the work. Section 5 includes the evaluations in several scenarios using different datasets and a use-case related to smart cities. Section 6 discusses the results of the experiments and Section 7 concludes the paper and describes the future work.

2 Related Work

There are several approaches to represent a numeric time-dependent variable (i.e., a time series). The most basic one is to compute the mean and standard deviation among other statistical measures (e.g., variance, mode). Using those statistics it is not possible to represent all the information that the time series contains. A classical example that supports this claim is the Anscombe's Quartet, [2] that shows how four very different datasets have identical simple statistical properties: mean, variance, correlation and regression coefficients.

In order to reduce the number of data points in a series and create a representation, segmentation methods can be used as a pre-processing step in data analytics.

Definition 1 (Segmentation). *Given a time series T containing n data points, segmentation is defined as the construction of a model $\bar{T}$, from l piecewise segments ($l < n$) such that $\bar{T}$ closely approximates T [3].*

The segmentation algorithms that aim to identify the observation where the probability distribution of a time series changes are called change-point detection algorithms. Sliding windows, bottom-up, and top-down methods are popular change-point detection based approaches. For sliding windows, each segment is grown until it exceeds an error threshold. The next block starts with the new data point not included in the newly approximated segment and so on. In the bottom-up methods, the segments of data are merged until some stopping criteria is met and top-down methods partition the time series recursively until a stopping criteria is met [4].

Another way of classifying the algorithmic methods for segmentation is considering them as online and offline solutions [5]. While offline segmentation is used when the entire time series is previously given, the online segmentation deals with points that arrive at each time interval. In offline mode, the algorithm first learns how to perform a particular task and then it is used to do it automatically. After the learning phase is completed, the system cannot improve or change (unless we consider incremental learning or retraining). On the other hand, online algorithms can adapt to possible changes in the environment. Those changes are known as "drifts". Whereas top-down and bottom-up methods can only be used offline, sliding windows are applicable to both circumstances.

After segmentation, the representation of the time series based on the reduction can be regarded as an initial step that reduces the load and improves the performance of tasks such as classification and clustering. The use of such algorithms can be generally regarded in two ways:

- Representation methods: Extracting features from the whole time series or its segments and applying machine learning algorithms (Support Vector Machines, Random Forest, etc) in order to classify them or compute the distance between the time series representation for clustering.
- Instanced based methods (similarities): Computing the distance matrix between the whole series and using it for clustering or classification applying a k-nearest neighbour approach [6] by finding the most similar (in distance) time series in the training set.

BEATS is based on the first perspective since as stated in *Bagnall et al The greatest improvement can be found through choice of data transformation, rather than classification algorithm* [7]. However, we review the work made using both approaches since the ultimate goal of our time series representation is to make the time series data more aggregated and better represented for further processing.

2.1 Whole Series Similarities

Similarity measures are used to quantify the distance between two raw time series. The list of approaches is vast and the comparison between well-known methods has lead to the conclusion that the benchmark for classification is dynamic time warping (DTW) since other techniques proposed before 2007 were found not significantly better [8].

Similar results have been stated in [9] when comparing DTW with more recent distance measures as: Weighted DTW [10], Time warp edit (TWE) [11] and Move-split-merge (MSM) [12] together with a slight accuracy improvement (1 percent) when using Complexity invariant distance (CID) [13] and Derivative transform distance (DTD_C) [14].

When computation time is not a problem, the best approach is to use a combination of nearest neighbour (NN) classifiers that use whole series elastic distance measures in the time domain and with first order derivatives: Elastic ensemble (EE) [15]. However, if a single measure is required a choice between DTW and MSM is recommended, with MSM preferred because of its overall performance.

In the clustering domain, the number of evaluated similarity distances is even higher, due to the nature of the problem. An extensive description of similarity measures can be found in [16]. DTW and CID are also used in clustering the raw time series [17], [18].

2.2 Intervals

Various algorithms focus on deriving features from intervals of each series. For a series of length m, there are $m(m-1)/2$ possible contiguous intervals.

Piecewise Linear Representation (PLR) [19] methods are based on the approximation of each segment in the form of straight lines and include the perceptually important points (PIP), Piecewise Aggregate Approximation (PAA) [20], and the turning point (TP) method [21].

The state-of-the-art models Time Series Forest (TSF) [22] and Learned pattern similarity (LPS) [23] generate many different random intervals and classifiers on each of them, ensembling the resulting predictions.

TSF trains several trees in a random forest fashion but each tree uses as data input the $3\sqrt{m}$ statistics features (mean, standard deviation and slope) of the $\sqrt{m}$ randomly selected intervals.

LPS can be regarded as an approximation of an autocorrelation function. For each series, they generate a random number l of series by randomly selecting a fixed number w of elements of the primitive one. A column of the generated $l * n \times w$ matrix is chosen as the class and a regression tree is built (autocorrelation part). After that, for every series the number of rows of the matrix (originated by the raw series) that reside in each leaf node is counted. Concatenating these counts the final representation of the series is formed. Then, a 1-NN classifier is applied to process the time series data.

2.3 Symbolic Aggregate Approximation (SAX)

Among all the techniques that have been used to reduce the number of points of a time series data, SAX has specially attracted the attention of the researchers in the field. SAX has been used to asses different problems such as finding time series discords [24], finding motifs in a database of shapes [25], and to compress data before finding abnormal deviations [26] and it has repeatedly been enhanced [27], [28], [29].

SAX allows a time series of length n to be reduced to a string of length l ($l < n$). The algorithm has two parameters: window length w and alphabet size α, and it involves three main steps [30]:

- Normalization: standardizes the data in order to have a zero mean and a standard deviation of one;
- Piecewise Aggregation Approximation (PAA): divides the original data into the desired number of windows and calculates the average of data falling into each window; and
- Symbolization: discretizes the aggregated data using an alphabet set with the size represented as an integer parameter α, where $\alpha > 2$.

  As normalized time series data assumes a Gaussian distribution for the data, the discretization phase allows to obtain a symbolic representation of the data by mapping the PAA coefficients to a set of equiprobable breakpoints that are produced according to the alphabet size α. The breakpoints determine equal-sized areas under the Gaussian curve [31] in which each area is assigned to an alphabet character.

Since SAX representation does not consider the segment trends, different segments with similar average values may be mapped to the same symbols. Among the multiple enhancements done to SAX (see related work section of [28] and [29]) we highlight the following works:

- Extended SAX (ESAX) [27]: adds maximum and minimum along with the original SAX representation.
- SAX Trend Distance (SAX_TD) [28]: defines the trend distance quantitatively by using the starting and ending point of the segment and improved the original SAX distance with the weighted trend distance.
- SAX with Standard Deviation (SAX_{SD}) [29]: adds the standard deviation of the segment to its SAX representation.

The Vector Space Model (VSM) is combined with SAX in [32] in order to discover and rank time series patterns by their importance to the class. Similarly to shapelets, SAX-VSM looks for time series subsequences which are characteristic representatives of a class. The algorithm converts all training time series into bags of SAX words and uses *tf-idf* weighting and cosine similarity in order to rank by importance the subsequences of SAX words according to the classes.

2.4 Shapelets

Shapelets are subsequences of time series that identify with the class that the time series belongs to.

The Fast shapelets (FS) [33] algorithm discretises and approximates shapelets using SAX. The dimensionality of the SAX dictionary is reduced through masking randomly selected letters (random projection).

Learned shapelets (LS) [34] optimizes a classification loss in order to learn shapelets whose minimal euclidean distances to the time series are used as features for a logistic regression model. An improvement of such model is the use of DTW instead of euclidean distance [35].

The Fused LAsso Generalized eigenvector method (FLAG) [36] is a combination of the state-of-the-art feature extraction technique of generalized eigenvector with the fused LASSO that reformulates the shapelet discovery task as a numerical optimization problem instead of a combinatorial search.

Finally, we take into consideration the clustering algorithm k-shape [37], a centroid-based clustering algorithm that can preserve the shapes of time-series sequences. They capture the shape-based similarity by using a normalized version of the cross-correlations measure and claims to be the only scalable method that significantly outperforms k-means.

2.5 Ensembles

So far we have reviewed how data transformation techniques are applied to different algorithms in order to improve their accuracy and to reduce the computation time. COTE algorithm [38] uses a collective of ensembles of classifiers on different data transformations.

The ensembling approach in COTE is unusual because it adopts a heterogeneous ensemble rather than resampling schemes with weak learners. COTE contains classifiers constructed in the time, frequency, change (autocorrelations), and shapelet transformation domains (35 in total) combined in alternative ensemble structures. Each classifier is assigned a weight based on the cross validation training accuracy, and new data are classified with a weighted vote.

The results of evaluations in COTE show that the simple collective formed by including all classifiers in one ensemble is significantly more accurate than any of its components.

3 Motivation and Contributions

As it can be seen among the segmentation techniques that we referenced in section 2, we have mentioned not only the representation techniques but also how the whole classification and clustering procedure is performed by combining representation with machine learning algorithms. We intended to show that our representation method is an efficient alternative segmentation method to be employed in time series data processing.

One commonality of the several studies that we have reviewed is that most of the existing algorithms use normalization that re-scales the data.

However, there are few studies that do not apply re-scaling and normalization. BEATS uses a non-normalized algorithm for constructing the segment representation.

The concept *drift* appears when a model built in the past is no longer fully applicable to the current data. Concept drift is due to a change in the data distribution according to a single feature, to a combination of features or in the class boundaries, since the underlying source generating the data is not stationary.

The potential changes in the data might happen in:

- The prior probability $\mathrm{P}(y_i)$;
- The conditional probability $\mathrm{P}(x|y_i)$;
- The posterior probability $\mathrm{P}(y_i|x)$; and
- A combination of the above.
  Where x is the predicted data and y_i is the observed data.

These changes can cause two kinds of concept drift: real and virtual [39].

If only the data distribution changes without any effect on the output, i.e., changes in $\mathrm{P}(y_i)$ and/or $\mathrm{P}(x|y_i)$ that does not affect $\mathrm{P}(y_i|x)$, it is called virtual drift.

When the output, i.e., $\mathrm{P}(y_i|x)$, also changes it is called real concept drift.

In the IoT domain and especially in smart city data analysis, we are interested in the second type of drift which will be referred as *data drift* in this paper [40]. Some examples where a data drift may occur in smart cities are related to the replacement of sensors (different calibration), sensor wear and tear [41] or drastic changes to the topics of discussion in social media used for crowdsensing [42].

There are several existing methods and solution addressing the concept drift for supervised learning [41], and some recent ones also for unsupervised learning [40]. However, we focus on the initial step of the analysis (i.e., pre-processing). We claim that not only the model has to be adaptive but also the way that we segment the inputs has to take into account the dynamics of the data and be able to efficiently deal with the changes in the structure of the data.

A considerable challenge in segmentation is to find a common way to represent the data. This is due to the variety of ways to formulate the problem in terms of defining the key parameters (number of segments, segmentation starting point, length of segments, error function, user-specified threshold, etc.).

The first step in SAX algorithm is assuming that for a particular problem that we deal with, the data follows a normal distribution or at least we have a sufficiently large number of samples in order to say that the distribution of the data is approximately normal, appealing to the central limit theorem [43]. Nevertheless, this is a strong assumption because there are many scenarios in which this might not be the case; for example:

- Outliers and noise: data from physical devices usually contains noise and outliers that affect the identification of the correct parameters of the distribution.
- Data follows different distribution.
- Fast data: two of the V's from the 7V's Big Data challenges [44] are *velocity* and *variety*. Traditionally in data mining, batch data is processed in an offline manner using historical data. However, in IoT applications we need to consider short-term snapshots of the data which are collected very quickly. Thus, we need adaptive methods that catch up with the changes during their operation.

All mentioned algorithms lack of at least one of such 3 problems too. We have developed an algorithm that does not require normalization of the data. The latter will also help to preserve the value of the data points (i.e., magnitude of the data). The lack of sensitivity to magnitude in the algorithms that make assumptions about the normalized distribution and use Z-normalization makes them less efficient in analysing correlation and regression. Another requirement is the application of the algorithm in an online way and using sliding windows. Nonetheless, we have to be able to compute the distance between the aggregated time series. Considering these requirements we have designed the BEATS algorithm.

4 BEATS PRESENTATION

This section describes our proposed algorithm and discusses its mathematical and analytical background. We present BEATS and show the effect of each step of the algorithm in a block of data.

4.1 BEATS Construction

Transforms, in particular integral transforms, are used to reduce the complexity in mathematical problems. In order to decorrelate the time features and reveal the hidden structure of the time series, they are transformed from the time domain into other domains. Well-known transformations are the Fourier Transform, which decomposes a signal into its frequency components, and the Karhunen-Loeve Transform (KLT) which decorrelates a signal sequence.

Discrete Cosine Transform (DCT) is similar to Discrete Fourier Transform (DFT) but uses cosines obtained from the discretization of the kernel of the Fourier Transform. DCT transfers the series to the frequency domain. Among the four different cosine transformations classified by Wang [45], the second one (i.e., DCT-II) is regarded as one of the best tools in digital signal processing [46] (times series can be regarded as a particular case of signals). Due to its mathematical properties such as unitarity, scaling in time, shift in time, the difference property, and the convolution property, DCT-II is asymptotically equivalent to the KLT where under certain (and general) conditions KLT is an optimal but impractical tool to represent a given random function in the mean square error sense (MSE). KLT is said to be an optimal transform because:

- It completely decorrelates the signal in the transform domain;
- It minimizes the MSE in bandwidth reduction or data compression;
- It contains the most variance (energy) in the fewest number of transform coefficients; and
- It minimizes the total representation entropy of the sequence.

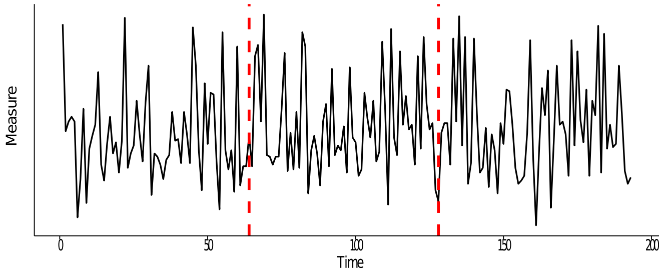

Fig. 1. An example of a time series divided into blocks of 64 observations.

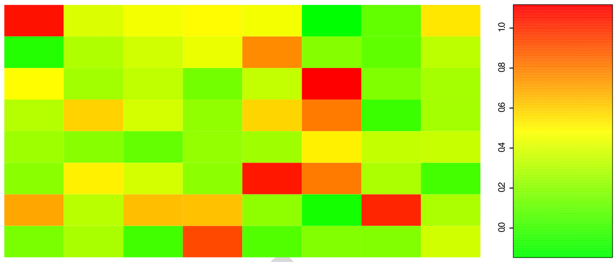

Fig. 2. The heatmap of the matrix obtained from the first block of time series data.

The details of the proof of the above statements can be found in [46]. Understanding the properties of the DCT, we use it to transform our time series data.

We apply the transformation essentially by using the compression of a stream of square 8x8 blocks, taking reference from the standards in image compression [47] where DCT is widely used (e.g., JPEG). Since 8 is a power of 2, it will ease the performance of the algorithm.

As an illustration, we provide an example. We have divided the time series shown in Fig. 1 as blocks of 64 observations that are shown using a dashed red line. If we arrange the first block row-wise into a squared matrix M, we can visualize that the information is spread through the matrix as the heatmap shown in Fig. 2.

It should be noted that while our raw time series data is represented in value/time, a 2D transformation is applied to the data. This is based on the assumption that in each block, the neighbour values of a selected observation m_{ij} (eg. $m_{i-1j}, m_{ij-1}, m_{i-1j-1}$ are correlated. In time series with very rapid changes in the data, small block sizes will be more suitable and if the changes are not very rapid size block can be larger. In this paper, we use a common 8×8 block size for our description.

Intuitively, each 8×8 block includes 64 observations of a discrete signal which is a function of a two-dimensional (2D) space. The DCT decomposes this signal into 64 orthogonal basis signals. Each DCT coefficient contains one of the 64 unique *spatial frequencies* which comprise the *spectrum* of the input series. The DCT coefficient values can be regarded as the relative amount of the spatial frequencies contained in the 64 observations [47].

Let M be the 8×8 input matrix. Then, the transformed matrix is computed as $D = UMU^{\mathsf{T}}$, where U is an 8×8 DCT matrix. U coefficients for the $n \times n$ case are computed as shown in Eq. 1:

$$U_{ij} = \begin{cases} \frac{\sqrt{2}}{2} & i, j = 1 \\ \cos\left(\frac{\pi}{n}(i-1)(j-\frac{1}{2})\right) & i, j > 1. \end{cases} \quad (1)$$

The formula of Eq. (1) is obtained using Eq. (5) (Appendix 8). Finally, we multiply the first term by $\frac{1}{\sqrt{2}}$ in order to make the DCT-II matrix orthogonal. After applying DCT, the information is accumulated in its upper-left part, as it is shown in the heatmap in Fig. 3.

Each of the 64 entries of the matrix D is quantized by pointwise division of the matrices D and Z, where the elements of the quantization matrix Z are integer values ranging from 1 to 255.

Quantization is the process of reducing the number of bits needed to store an integer value by reducing the precision of the integer. Given a matrix of DCT coefficients, we can divide them by their corresponding quantizer step size and round it up depending on its magnitude, normally 2 decimals. If the maximum of the DCT matrix is small, the number of decimals is selected by the operation $|\lfloor \log_{10} max \rfloor - 4|$, where $\lfloor \log_{10} max \rfloor$ returns the position of the first significant figure of the maximum number in the transformed matrix D. This step is used to remove the high frequencies or to discard information which is not very significant in large-scale observations.

The selected matrix Z is the standard quantization matrix for DCT [48].

After the quantization process, a large number of zeroes appears in the bottom-right position of the matrix $Q = \frac{D}{Z}$, i.e., it is a sparse matrix.

We extract the 4×4 upper-left matrix that contains the information of our 64 raw data and compute the eigenvalues, which in our case are: $0.18605, 0.02455, 0.00275 + 0.00843i, 0.00275 - 0.00843i$.

Using BEATS so far we have significantly reduced the number of points of our time series from 64 to 4 but we have also converted its components into complex numbers. These complex numbers (eigenvalues vector) represent the original block in a lower dimension. This eigenvalues vector is used in BEATS to represent the segments and hence, it is the potential input for the machine learning models. However, it is not always possible to feed machine learning algorithms with complex numbers and the eigenvalues could be complex numbers. To solve this problem, we compute the modulus of the eigenvalues and remove the repeated ones (they are presented in pairs so the information would be repeated).

In case that there are no complex numbers in the output of BEATS, we will conserve the first three values, since the latter values are sorted in a descending order. This means that we have represented the original 64 observations as

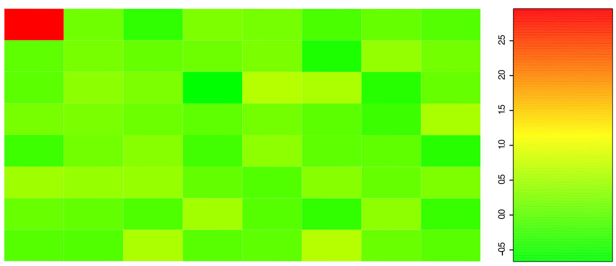

Fig. 3. The heatmap of the DCT matrix.

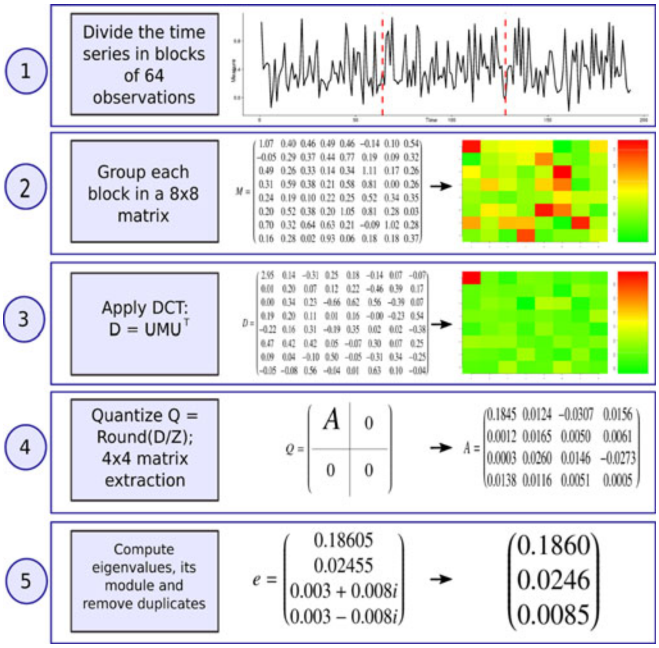

Fig. 4. BEATS is shown step by step with an example.

three values. In our example, the final representation (modulus of the eigenvalues) consists of $0.1860, 0.0246, 0.0085$.

The BEATS process is summarized in Fig. 4.

We also consider the relevance of the direct computation of the eigenvalues of the 8×8 matrix M in order to assure that the DCT and its quantization contribute to the aggregation of the information. We refer to this method throughout the paper as Eigen.

4.2 Complexity Analysis of BEATS

The time complexity is represented as a function of the input time series size (n). Regarding the different steps of BEATS, the processes that have a key impact on the run time are DCT, which is a double matrix multiplication, i.e., $O(n^3)$; pointwise matrix division for the quantization, i.e., $O(n^2)$ and eigenvalue computation, i.e., $O(\beta^3)$, where n is the size of the matrix block (square root of the amount of data that compounds each block), and β $(\leq n)$ is the size of the extracted matrix from which we compute the eigenvalues. Although we have set the values to $n = 8$ and $\beta = 4$, we compute the complexity in general terms.

So far, the dominant task regarding the complexity is the DCT function. For about the past 40 years, many fast algorithms have been reported to enhance the computation of discrete cosine transforms [49]. In order to improve the efficiency of the algorithm, we have implemented a popular way of computing the DCT of our N-points time series. We use a 2N-points Fast Fourier Transform (FFT). This has reduced the complexity to $O(n^2 log(n))$ [50].

Hence, for each block we have a complexity of $O(n^2 log(n) + \beta^3)$. Let N be the size of our time series data; if we do not use sliding windows, we will apply the algorithm $\frac{N}{n \times n}$ times, so the complexity is $\frac{N}{n \times n} O(n^2 log(n) + \beta^3)$. As we can see, the complexity of the algorithm grows linearly depending on the number of blocks where we have to apply the computations.

By applying multiple processing architectures, the complexity problem nowadays can also depend on how efficiently we can parallelize the processing load. Parallelising the BEATS algorithm is very simple since the computations are *block dependent* and no information out of the block is required for each individual calculation. This makes the process ideal to be done using graphics processing units (GPUs), and thereby minimising the latency of the computation.

5 Experimental Evaluation

We perform two data mining processes: classification and clustering. Following our approach the data is going to be transformed by the two methods: BEATS and Eigen, summarized as follows:

- BEATS: 8×8 matrix blocks of the data, discrete cosine transformation, and quantization of each of the matrices, reduction to a 4×4 matrix, removal of the duplicated modulus of the complex eigenvalues and selection of the first three values.
- Eigen: 8×8 matrix blocks of the data, computation of the eigenvalues of the matrices, removal of the duplicated modulus of the complex eigenvalues, and selection of the first three values.

Having introduced several algorithms in Section 2, we compare BEATS and Eigen with common existing state-of-the-art methods that show an improvement in comparison with the primitive ones.

The algorithms' code has been accessed from the authors' public repositories when available. When not, R software and Python have been used in order to program them.

We perform each of the techniques using several datasets in order to analyse the type of problems that our algorithm performs better than other methods. It is possible to use sliding windows for our method. In the experiment, we consider a slide of 8 observations. The evaluations also include a cross validation step in order to find their parameters.

A smart cities use case where we cluster traffic data is also presented. The intention is to see how BEATS is suitable for different scenarios including online smart cities applications.

5.1 Datasets

We give a short explanation of the datasets that are used to evaluate the algorithm. Four of the datasets are obtained from the UCR Time Series Classification Archive [51]: ArrowHeads, Coffee, FordA, Lightning7 and ProximalPhalanxOutlineAgeGroup. For each dataset we use, when provided, the train sample in order to find the hyperparameters of the model and then, we test their classification performance with the test set. For clustering we use only the training set. When the split is not provided, which is the case in one of the datasets (the randomly generated by us), we use 75 percent of the samples for the training set and 25 percent of the samples for testing.

The datasets that are used in the experiments are briefly described below.

Arrow Heads (Real and Without Drifts). The Arrow Heads dataset[1] contains 211 series having 192 observations classified into three different classes. The arrowhead data consists

1. http://www.cs.ucr.edu/~eamonn/time_series_data/

of outlines of the images of arrowheads [52]. The shapes of the projectile points are converted into a time series using the angle-based method and they are classified based on shape distinctions such as the presence and location of a notch in the arrow. The classification of projectile points is an important topic in anthropology. According to our method, we reduced the dataset to 72 observations.

Lightning7 (Real and Long). We use the Lightning7 dataset that gathers data related to transient electromagnetic events associated with the lightning natural phenomenon. Data is gathered with a satellite with a sample rate of 800 microseconds and a transformation is applied in order to produce series of length 637.

The classes of interest are related to the way that the lightning is produced.[2]

Initially, each measurement (time series) carries 320 variables. Using our method, we have reduced the dataset to 96 variables.

Random LHS Generator Lift (Synthetic and with Drifts). A dataset with data drifts is also used in our experiments. In this case, we have evaluated the algorithms with the data generated by using the code from the Repository[3] described in [53], which was first used in [40]. The drift is introduced both by shifting the centroids in randomized intervals and by changing the data distribution function used to randomly draw the data from the centroids that are selected through Latin Hypercube Sampling (LHS). This dataset is created for smart cities data analysis and allows to create sample datasets that simulate dynamic and multi-variate data streams in a smart environment. The data generator is developed in the context of the CityPulse smart city project.[4]

The number of centroids is set to ten and we generated 300 series that follow three different distributions (triangular, Gaussian and exponential). Initially, each set (time series) carries 192 variables. Using our method, we reduced the dataset to 51 variables.

Coffee (Real-World Data). The Coffee dataset[1] contains 56 series having 286 observations classified into two different classes. The Coffee data consists of the series generated by the Fourier transform infrared spectroscopy of two species of coffee: Arabica and Robusta. Originally, such method intended to serve as an alternative to wet chemical methods for authentication and quantification of coffee products [54]. Using BEATS, we reduced the dataset to 57 observations which represent the patterns that occur in the dataset. This can be used for further analysis and classification of coffee types.

FordA (Real-World Data). The FordA dataset[1] contains 4921 series having 500 observations each classified into two different classes. The data was generated on the context of a classification competition. The problem is to diagnose whether a certain symptom exists in a automotive subsystem using the engine noise as a measurement. Both training and test data set were collected in typical operating conditions, with minimal noise contamination. Using BEATS, we reduced the dataset to 100 observations. The BEATS observations are more resilient to noise and provide an efficient way to discover and extract patterns from real-world raw data.

ProximalPhalanxOutlineAgeGroup (Real-World Data from Images). The ProximalPhalanxOutlineAgeGroup dataset[1] contains 605 series having 80 observations each classified into three different classes. The dataset was created [55] for testing the efficacy of hand and bone outline detection and whether these outlines could be helpful in bone age prediction. The problem involves using the outline of one of the phalanges of the hand in order to predict whether the subject is one of three age groups . Using BEATS, we reduced the dataset to 9 observations per subject. This observations provide a reduced feature set that ease the analysis tasks.

5.2 Classification

Classification of time series analysis is a classic problem consisting of building a model based on labelled time series data and using the model to predict the label of unlabelled time series samples.

The applications of this technique are widely extended in many areas, ranging from epilepsy diagnosis based on time series recorded by electroencephalography devices (electrical activity generated by brain structures over the scalp) [56] to uncovering customers' behavior in the telecommunication industry [57], and predicting traffic patterns in a smart city environment.

After transforming our data using BEATS and Eigen, we followed the general data modelling process proposed in [58] to classify the series: standarization, splitting the dataset into training and test sets, choosing the model, selecting the best hyperparameters of each model using 10-fold cross validation on the training set and checking the accuracy of the model using the test set. With respect to the methodology followed in [58], we improve the way of looking for the hyperparemeters of the algorithms using the python package optunity since it contains various optimizers for hyperparameter tuning.

Among other options like grid search, random search and genetic algorithms, we have chosen particle swarm implementation since it is shown to surpass the performance of other solutions [59].

The models that we use to combine with BEATS and Eigen are the widely known Random Forest (RF) and Support Vector Machines (SVM) with Radial Basis Function Kernel.

Whereas Random Forest deals with *small n large p-problems*, high-order interactions and correlated predictor variables, SVMs are more effective for relatively small datasets with fewer outliers. Generally speaking, Random Forests may require more data. Both of the algorithm show better performance when combined with SVM.

The tuning of SVM has been done without deciding the kernel in advance. That means, the kernel (linear, polynomial or RBF) is considered as an hyperparameter.

According to the discussion in Section 2, we compare our method with:

- Original time series (i.e., raw data): DTW with 1-NN classification since, after many trials, it is still the benchmark of comparison for distance based classification. Having a complexity of $O(n^2)$ that under

2. http://www.timeseriesclassification.com/description.php?Dataset=Lightning7
3. https://github.com/auroragonzalez/BEATS/tree/master/data/random_LHS_generator_drift
4. http://www.ict-citypulse.eu

TABLE 1
Accuracy of Each Method Using as Inputs Each of the Segmented Time Series

Model \ dataset	Arrow Heads	Lightning7	Random Generator	Coffee	Ford A	Proximal
BEATS-SVM	**0.81**	0.7	**0.75**	**1**	**0.75**	**0.85**
Eigen-SVM	0.79	0.72	0.73	**1**	0.74	0.8
DTW-1NN	0.67	0.75	0.71	0.87	0.66	0.81
SAX-VSM	0.68	0.59	0.52	0.96	0.09*	0.75
TSF	0.73	0.75	**0.75**	0.97	**0.75**	0.85
FLAG	0.57	0.76	0.67	**1**	0.73	0.64
COTE	0.78	**0.8**	0.7	**1**	**0.75**	0.83

**The bag of words generated by a wide majority of the test subjects is not related to the ones generated by the train step. This implies that their TF*IDF weights are not computed and it is not possible to compute the cosine similarity. In consequence, the method is not valid for many of the cases, producing the reported bad results.*

certain circumstances [60] could be reduced to $O(n)$ using lower bounds such as LB_{Keogh} or $LB_{Improved}$ [61].

- Intervals: We choose TSF in order to make the comparison since it is more modern and quicker than the rest.
  Its complexity is $O(t * m * n * logn)$, where t = number of trees and m = number of splits or segments.
- Symbolic approximations: In the classification task, we use SAX-VSM. The complexity is linear: $O(n)$.
- Shapelets: FLAG is the newest, the quickest and claims to be better than its predecessors.
  Its complexity is $O(n^3)$.
- Ensembles: COTE. It is an ensemble of dozens of core classifiers many of which having a quadratic, cubic or even bi-quadratic complexity. It is the most computationally expensive in this list.

The results are shown in Table 1. It is important to mention that not only accuracy results but also the time that it takes the algorithm to run both training and test phases including input transformation, has improved. This runtime is shown in Fig. 5, where a logarithmic transformation is applied to the data in order to improve visibility.

We have depicted both metrics: accuracy and running time in a plot that summarises the results over all the datasets. Both metrics have been scaled per dataset and we have computed the average performance per model that is represented by the bigger points in the plot.

In order to make a more consistent analysis of the results, we have generated 100 Random LHS Generator Lift datasets and the model accuracy of the models using violin plots (see Fig. 6), which together with the regular statistics that boxplot provide they show the probability density of the data at different values of accuracies. While the differences between BEATS-SVM, TSF and COTE are not statistically significant (p-value = $0.7 > 0.05$), BEATS-SVM is very quick in comparison to COTE and that BEATS is also more versatile than the rest since it can be combined with any classification algorithms.

5.3 Clustering

Clustering is used to identify the structure of an unlabeled dataset by organising the data into homogeneous groups where within-group-object similarity is minimized and between-group-object dissimilarity is maximized. The process is done without consulting known class labels. Clustering is an unsupervised machine learning method. In particular, time series clustering partitions time series data into groups based on similarity or distance; so that time series data in the same cluster are similar.

Clustering has tackled tasks such as the assignment of genes with similar expression trajectories to the same group [62]. The creation of profiles of the trips carried out by tram users [63] or the acquisition of energy consumption predictions by clustering houses [64] are among examples of using clustering methods.

After transforming our data using BEATS and Eigen, we applied the connectivity based algorithm *hierarchical agglomerative clustering* and the centroid based algorithm *k-means* to

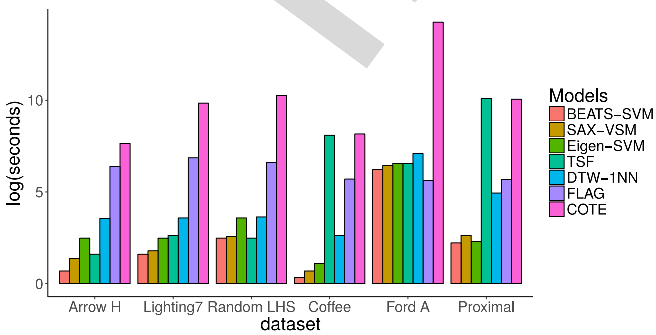

Fig. 5. Running time (log(sec)) and programming language of the algorithms.

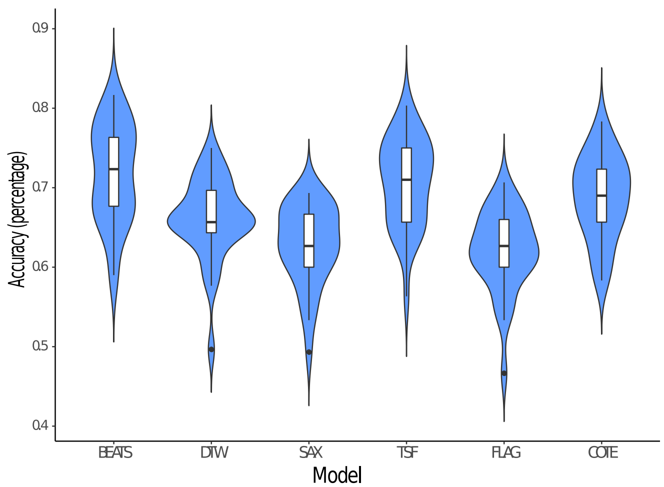

Fig. 6. Classification accuracy on the 100 randomly generated datasets.

TABLE 2
Silhouette Coefficient of Each Method Using as Inputs Each of the Segmented Time Series

Model \ dataset	Arrow Heads	Lightning7	Random Generator	Coffee	Ford A	Proximal
BEATS-HC	**0.6**	0.25	**0.45**	0.25	**0.46**	0.4
Eigen-HC	0.58	**0.31**	0.25	0.26	0.36	0.38
DTW	0.33	0.21	0.44	0.21	0.12	0.31
SAX$_{SD}$- HC	0.53	0.06	0.19	0.13	0	0.33
k-shape	0.44	0.19	0.05	**0.43**	0.38	**0.5**

cluster the time series datasets. In the hierarchical clustering, the selected agglomerative method is *complete linkage*, meaning that the distance between two clusters is the maximum distance between their individual components (in each time series). Hierarchical clustering seems to be a better partner for both of them.

The dissimilarity matrix contains the distances between the pairs of time series. We use the cosine dissimilarity for the rest of the segmentations (BEATS and Eigen). The cosine dissimilarity is calculated as one minus the cosine of the included angle between elements of the time series (see Eq. (2))

$$\text{dissimilarity} = 1 - \frac{\mathbf{XY}}{\|\mathbf{X}\|\|\mathbf{Y}\|} = 1 - \frac{\sum_{i=1}^{n} X_i Y_i}{\sqrt{\sum_{i=1}^{n} X_i^2}\sqrt{\sum_{i=1}^{n} Y_i^2}}. \quad (2)$$

Finally, for both methods we have used a fixed number of clusters. As we were aware of the classification groups (our data is labeled), we applied the algorithms setting apriori the number of clusters k and used the silhouette coefficient as a metric for measuring the cluster quality.

The silhouette coefficient is an internal measure that combines the measurement of cohesion and separation. Cluster cohesion measures how closely related the objects in a cluster are. Cluster separation measures how well separated the clusters are from each other. The silhouette coefficient for a subject i is defined as

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}}, \quad (3)$$

where $a(i)$ is the average distance between i and each of the points of the assigned cluster and $b(i)$ is the average distance between i and each of the points of the next best cluster. This value can be used to compare the quality of different cluster results.

From the definition it is clear that $s(i) \in [-1, 1]$. Meanwhile a silhouette coefficient value closer to 1 means that the clustering is good; a value close to -1 represents less efficiency in the categorization for the clusters. When it is close to 0, it means that the point is in the border between two clusters.

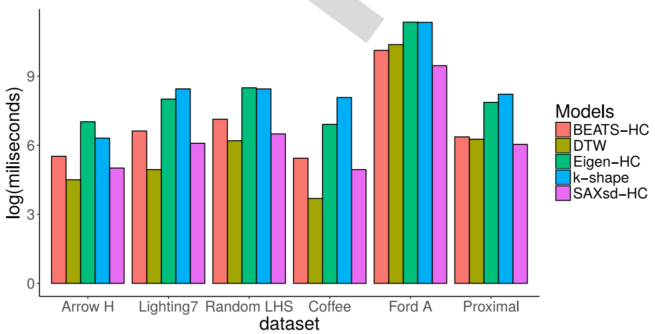

Fig. 7. Running time (log(milisec)) of the clustering algorithms.

According to the discussion in Section 2 we will analyse:

- Original time series: DTW distance using the tight lower bound of [61], that makes it faster.
- Symbolic approximations: We have taken the most modern improvement that SAX has experienced: SAX$_{SD}$. The MINDIST function that returns the minimum distance between the original time series of two words [65] is enhanced with the distance between the standard deviation of each segment.
- Shapelets: k-shape is the model chosen in this direction.

The results of the clustering experiments done in the training sets are shown in Table 2. The run time of the algorithms is shown in Fig. 7. In this case, all the algorithms have been coded using the same programming language so we consider that the graph is enough in order to estimate the different algorithms complexity regarding time.

5.4 Big Data Use Case: Traffic in Smart Cities

In this section we apply BEATS in a smart cities related use-case: traffic data clustering, done in an online and distributed way.

5.4.1 *BEATS Implementation for Big Data*

In contrast to the traditional analysis procedure where data is first stored and then processed in order to deploy models, the major potential of the data generated by IoT is accomplished by the realization of continuous analytics that allow to make decisions in real time.

There are three types of data processing: Batch Processing, Stream Processing and Hybrid Processing.

Batch processing operates over a group of transactions collected over a period of time and reports results only when all computations are done, whereas stream processing produces incremental results as soon as they are ready [66].

Regarding the available Big Data Tools, we have considered Hadoop[5] and Spark[6] Big Data frameworks. Hadoop was designed for batch processing. All data is loaded into HDFS and then MapReduce starts a batch job to process that data. If the data changes the job needs to be ran again. It is step by step processing that can be paused or interrupted, but not changed.

Apache Spark allows to perform analytical tasks on distributed computing clusters. Sparks real-time data

5. http://hadoop.apache.org/
6. https://spark.apache.org/

processing capability provides substantial lead over Hadoops MapReduce and it is essential for online time series segmentation and representation.

The Spark abstraction for a continuous stream of data is called a Discretized Stream or DStream . A DStream is a micro-batch of Resilient Distributed Datasets, RDDs. That means, a DStream is represented as a sequence of RDDs. RDDs are distributed collections that can be operated in parallel by arbitrary functions and by transformations over a sliding window of data (windowed computations).

5.4.2 BEATS Adapted to Spark Technology

For the online implementation of BEATS we have decided to use pyspark, the Spark Python API that exposes the Spark programming model to Python.

There are many works proposing online time series processing but few of them that have implemented it. In [67] is highlighted that MapReduce is not the appropiate techonology for rolling window time series prediction and proposes a index pool data structure.

Pyspark allows us to use the Spark Streaming functionalities that are needed in order to implement BEATS online. In Section 3 we have seen that BEATS algorithm can be separately applied to windows of the data. Therefore we associate the data received within one window to one RDD, that can be processed in a parallel way.

A suitable type of RDDs for our implementation is key/value pairs. In detail, the key is an identifier of the time series (e.g., sensor name) and the value is the sequence of values of our time series that fall in the window. That way the blocks are exposed to operations that give the possibility to act on each key in parallel or regroup data across the network.

The transformations that we use are:

- Window: use for creating sliding window of time over the incoming data.
- GroupByKey: grouping the incoming values of the sliding window by key (for example, same sensor data).
- Map: The Map function applied in parallel to every pair (key, value), where the key is the time series, values are a vector and the function depends on what has to be done.

5.4.3 The Applied Scenario

We use one of the real-world datasets obtained from the collection of datasets of vehicle traffic in the City of Aarhus in Denmark for a period of 6 months.[7] The dataset is provided in the context of the CityPulse smart city project.

The selected dataset gathers 16971 samples of data from sensors situated in lamp posts covering an area around 2345m.[8] The variables considered for the analysis are: flow (numbers of cars between two points) and average speed. Each variable is a time series.

In order to simulate an online application we consider that the BEATS segmentation is carried out on hourly based data.

7. http://iot.ee.surrey.ac.uk:8080/datasets.html#traffic
8. http://iot.ee.surrey.ac.uk:8080/datasets/traffic/traffic_june_sep/index.html

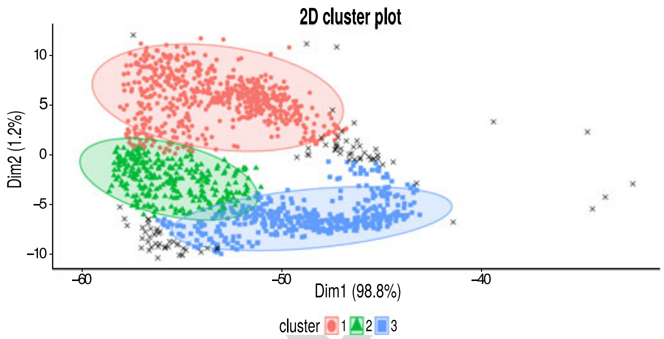

Fig. 8. Plot of the DBSCAN clusters using traffic data.

To achieve this, since the data is collected every 5 mins., a sliding window of size 12 is selected. The goal of the clustering is to determine the status of the road in terms of the traffic flow and occurrences. For every window of 128 observations (64 for each variable) BEATS obtains three flow related representatives and three speed related representatives.

Each observation of the final input dataset for the clustering model represents one window of the raw data. The final dataset has 6 variables and 1409 samples. This means a reduction of around 75 percent of data.

The data is gathered by anonymously collecting Wi-Fi and Bluetooth signals transmitted by travelers' smartphones or in-vehicle systems. This infrastructure provides noisy data in cases such as stopped vehicles in traffic jam, buses with a lot of passengers.

In order to tackle the presence of outliers and noise, the selected clustering technique is density-based spatial clustering (DBSCAN). DBSCAN groups points that are closely packed together. Points that do not fit into any of the main groups because they lie in low-density regions are marked as outliers. The hyper-parameters of DBSCAN are minimum number of points required to form a dense region (MinP) and ϵ in order to find the ϵ-neighborhood of each point. We set that clusters contain at least a 20 percent of the data and $\epsilon = 4.014$. Using such configuration, we obtain 3 different clusters and a 8 percent of data that cannot be classified in any of the previous, i.e., outliers. The description of the clusters, including the number of points n that belong to each of the clusters and the mean μ and standard deviation sd for both flow and speed is:

- Cluster 1 (n = 618): High flow (μ = 30.97, sd= 12.66) and medium speed (μ = 102.5, sd= 10.2);
- Cluster 2 (n = 271): Medium flow (μ = 15.97, sd= 8.4) and high speed (μ = 110, sd= 9.21); and
- Cluster 3 (n = 432): Low flow (μ = 6.1, sd= 5.56) and low/medium speed (μ = 97.8, sd= 14.3).

In order to represent the data in lower dimension, we select the first two principal components of the data using Principal Components Analysis (PCA). The obtained clusters are shown in Fig. 8. Crosses in black colour represent the noise data. We have also projected the clusters in the three flow related components of BEATS, so that clusters can be visualized in a 3D form as presented in Fig. 9.

Regarding this application, we can conclude that clustering methods applied to the segments generated by BEATS are able to characterise the status of the roads by grouping the values in an effective form.

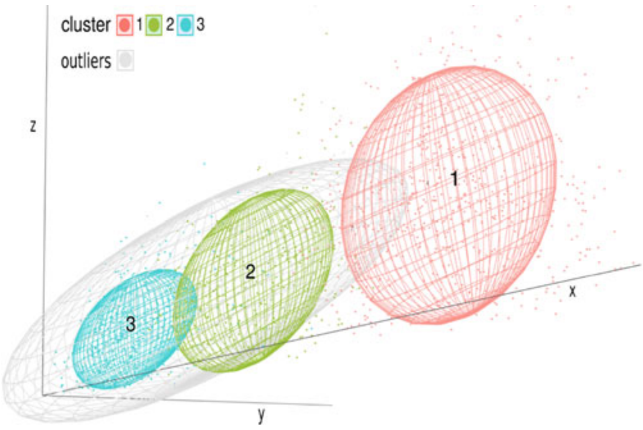

Fig. 9. 3D plot of the DBSCAN clusters using traffic data.

Using a computer with an Intel i5 Processor, 8GB RAM Memory, Ubuntu 16.04 operative system and the statistical software R 3.4.3 [68], the running time of DBSCAN using BEATS segmented data is 0.25 seconds. However, to run the DBSCAN with raw data it takes around 35 seconds. The later confirms again the suitability of BEATS in current IoT scenarios.

6 Discussion

As we have described in the paper, the randomness and predictability of a real-world time series changes over time due to several factors.

The existing solutions for pattern creation and abstraction in time-series data often work based on statistical measures (which have limited representation and granularity), symbolic methods such as SAX (which assumes that the data is normally distributed and requires normalization of the data), or signal processing and stream processing methods such as wavelet or Fourier transforms (which act as filters and can extract features from the data but do not provide a pattern representation/abstraction).

Our proposed model combines a series of methods to create a window based abstraction of time series data and uses a frequency domain function combined with characteristic value measures that represents the overall direction of the dataframe (i.e., an n-dimensional matrix constructed during our windowing/slicing process) as a vector.

BEATS is an algorithm that process data streams whose randomness and predictability varies depending on the segment of data. The proposed algorithm is useful specially in applications such as smart cities where results of the segmentation and processing algorithms are used in order to make fast decisions regarding traffic, energy, light regulation, etc. This can be made by combining various sensory data and other historical data. In general terms, the intention is to predict and manage what is occurring in order to provide informed or automated decisions for repetitive tasks that can be handled by machines. BEATS offers a powerful solution to aggregate and represent large-scale streaming data in a quick and adaptable way. It uses blocks of eigenvalues in a much lower-dimensionality (with a high aggregation rate) which preserves the main information and characteristics of the data. Since BEATS uses eigenvalues, it provides a homogeneous way to represent multi-modal and heterogeneous streaming data. In other words, all different types of numerical streaming data are transformed into vectors of eigenvalues that, in principal, preserve and represent the magnitude and overall direction of the data in a lower-dimensionality space. This not only allows to compare and combine different blocks of data from various data streams, but also provides a unified way to represent the blocks of data as patterns in the form of eigenvalues.

In this paper, we mainly target a key step after collection of the data: aggregation. Aggregation of data becomes a very significant task in order to extract the key characteristics of the data in lower-dimensionality. We segment the time series and make a reduction for each time series at a rate of 60 ~ 70 percent when using overlapping windows. The independence between blocks that our algorithm provides is one of its most important features. BEATS also presents other qualities such as adapting to drifts and low latency.

BEATS reduces the data by using the eigenvalues of a submatrix of the DCT transformation. These eigenvalues represent the key-characteristics of the data.

The evaluation is performed using classification and clustering, two of the classical machine learning tasks using several types of datasets. The inputs of the models are the different representations introduced in the paper: BEATS and Eigen together with raw data for the other models.

Classification is measured by accuracy. This allows us to perform a test for equality of proportions, that is a χ^2 test of independence in order to assure that the differences between accuracies are statistically significant.

For the Arrow Heads dataset we find that BEATS combined with SVM outperforms all the algorithms. However, the differences between COTE and BEATS are not statistically significant ($\chi^2(1) = 0.37$, p-value $= 0.54 > 0.05$). On the other hand, the difference between TSF and BEATS are statistically significant ($\chi^2(1) = 4.8$, p-value $= 0.04 < 0.05$).

In the case of Lightning7, there are several models that outperform BEATS. The winning one is COTE. Nonetheless, COTE is very complicated, time demanding and computationally expensive. The rest only overperforms BEATS by 6 percent at most.

In the case of Random LHS Generator Lift, TSF and BEATS perform similarly.

In the Coffee dataset, we observe that several approaches (including BEATS) achieve a 100 percent accuracy on classification.

In FordA, BEATS, TSF and COTE perform similarly. However, BEATS is the quickest amongst them.

Finally, in the Proximal dataset TSF and BEATS perform similarly in terms of accuracy. However, BEATS is again quicker.

Even though COTE and TSF are strong rivals to BEATS, it should be noted that the computation time and simplicity of BEATS makes it useful to use in rapid analysis having still good results. Also, due to its nature is very adaptable and easy to combine with any other classification algorithm different than SVM.

The clustering experiment is evaluated by comparing the hundredths of the silhouette coefficients, where each hundredth is going to be counted as *a point* in the below description.

BEATS is 7 points above SAX_{SD} for the Arrow Heads dataset, 1 point above DTW in the Random LHS Generator Lift set and 8 points above k-shape in Ford A. Being the most computationally expensive of all the clustering algorithms under study, as it can be seen in Fig. 7, k-shape outperforms BEATS in two datasets: Coffee and Proximal.

It can be said that in clustering, BEATS behaves better when we are using long datasets since it outperforms every algorithms in both metrics: silhouette coefficient and running time in the biggest dataset: $FordA$.

Finally, by applying DBSCAN to cluster traffic data, we noticed that BEATS performs efficiently since the clusters represent different situations of the use-case in terms of traffic flow and speed.

7 Conclusions and Future Work

In this paper, we introduce a novel algorithm called BEATS, which aggregates and represents time series data in blocks of lower-dimensional vectors of eigenvalues. BEATS is not sample dependent so it adapts to data drifts in the underlying data streams.

The BEATS abstractions can be combined with various machine learning models to discover patterns, identify correlations (within or between data streams), extract insights and identify activities from the data. In this paper, we have used several datasets and have shown several use cases that demonstrate how the BEATS abstractions can be used for clustering, analysis and grouping the activities and patterns in time-series data.

Compared to existing segmentation methods, BEATS shows significant improvements in representing datasets with drifts. When combined with classification and clustering methods, we have shown that it can obtain competitive results compared with other state-of-the-art but more complex and time consuming methods.

For the BEATS algorithm evaluation we have fixed the length of the segments at 64; so the only parameter to take into consideration was the slide of the window, that we have kept constantly equal to 8, so the blocks of transformed data intersect. Nevertheless, the optimization of the sliding window is an open issue to be addressed in future work.

For the clustering tasks, it is important to take into account that the definition of similarity is subjective. The similarity depends on the domain of application.

By using BEATS, we are able to restructure the streaming data in a 2D way and then transform it into the frequency domain using DCT. The algorithm finds a smaller sequence that contains the key information of the initial representative. This aggregation provides an opportunity to eliminate repetitive content and similarities that can be found in the sequence of data.

The eigenvalues vectors are a homogeneous representation of the data streams in BEATS that allow us to go one step further in understanding of the sequences and patterns that can be considered as the data structure of a data series in an application domain (e.g., smart cities).

Its applications can be extended to several other domains and various patterns/activity monitoring and detection methods. The future work will focus on applying 3D cosine transform and adaptive block size estimation.

Appendix A

Definition A.1 (Integral transform). *The integral transform of the funtion $f(t)$ with respect to the kernel $K(t,s)$ is*

$$F(t) = \int_{-\infty}^{\infty} K(t,s) f(t) dt, \tag{4}$$

if the integral exists.

The kernel of the Fourier Transformation is $K(t,s) = e^{-its}$, and, in particular for the cosine fourier transformation $K(t,\omega) = cos(t,\omega)$. If we discretize the kernel we can reach that $K_c(j,k) = cos(\frac{jk\pi}{N})$, where N is an integer.

Definition A.2. (Discrete Cosine Transformation (DCT) - II). *DCT is a linear and invertible function*

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}^n$$

where $\mathbb{R}$ denotes the set of real numbers or, equivalently, on a $n \times n$ matrix, defined by:

$$f_j = \sum_{k=0}^{n-1} cos\left(\frac{\pi}{n} j \left(k + \frac{1}{2}\right)\right) \text{ where } j = 0, 1, \ldots, n-1 \tag{5}$$

Acknowledgments

This work has been partially funded by MINECO grant BES-2015-071956, PERSEIDES TIN2017-86885-R project, and ERDF funds, by the European Comission through the H2020-ENTROPY-649849 EU Project, and the H2020 FIESTA Project under grant agreement no. CNECT-ICT-643943.

References

[1] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze Future*, vol. 2007, pp. 1–16, 2012.

[2] D. Abbott, *Applied Predictive Analytics: Principles and Techniques for the Professional Data Analyst*. Hoboken, NJ, USA: Wiley, 2014.

[3] E. J. Keogh and M. J. Pazzani, "An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback," in *Proc. 4th Int. Conf. Knowl. Discovery Data Mining*, vol. 98, pp. 239–243, 1998.

[4] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "Segmenting time series: A survey and novel approach," *Data Mining Time Series Databases*, vol. 57, pp. 1–22, 2004.

[5] H. Aksoy, A. Gedikli, N. E. Unal, and A. Kehagias, "Fast segmentation algorithms for long hydrometeorological time series," *Hydrological Processes*, vol. 22, no. 23, pp. 4600–4608, 2008.

[6] X. Xi, E. Keogh, C. Shelton, L. Wei, and C. A. Ratanamahatana, "Fast time series classification using numerosity reduction," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 1033–1040.

[7] A. Bagnall, L. Davis, J. Hills, and J. Lines, "Transformation based ensembles for time series classification," in *Proc. SIAM Int. Conf. Data Mining*, 2012, pp. 307–318.

[8] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh, "Experimental comparison of representation methods and distance measures for time series data," *Data Mining Knowl. Discovery*, vol. 26, pp. 1–35, 2013.

[9] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, "The great time series classification bake off: A review and experimental evaluation of recent algorithmic advances," *Data Mining Knowl. Discovery*, Springer, vol. 31, no. 3, pp. 606–660, 2017.

[10] Y.-S. Jeong, M. K. Jeong, and O. A. Omitaomu, "Weighted dynamic time warping for time series classification," *Pattern Recognit.*, vol. 44, no. 9, pp. 2231–2240, 2011.

[11] P.-F. Marteau, "Time warp edit distance with stiffness adjustment for time series matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 2, pp. 306–318, Feb. 2009.

[12] A. Stefan, V. Athitsos, and G. Das, "The move-split-merge metric for time series," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 6, pp. 1425–1438, Jun. 2013.

[13] G. E. Batista, X. Wang, and E. J. Keogh, "A complexity-invariant distance measure for time series," in *Proc. SIAM Int. Conf. Data Mining*, 2011, pp. 699–710.

[14] T. Górecki and M. Łuczak, "Non-isometric transforms in time series classification using DTW," *Knowl.-Based Syst.*, vol. 61, pp. 98–108, 2014.

[15] J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," *Data Mining Knowl. Discovery*, vol. 29, no. 3, pp. 565–592, 2015.

[16] S. Aghabozorgi, A. S. Shirkhorshidi, and T. Y. Wah, "Time-series clustering–a decade review," *Inf. Syst.*, vol. 53, pp. 16–38, 2015.

[17] V. Hautamaki, P. Nykanen, and P. Franti, "Time-series clustering by approximate prototypes," in *Proc. 19th Int. Conf. Pattern Recognit*, 2008, pp. 1–4.

[18] G. E. Batista, E. J. Keogh, O. M. Tataw, and V. M. De Souza, "Cid: An efficient complexity-invariant distance for time series," *Data Mining Knowl. Discovery*, vol. 28, no. 3, pp. 634–669, 2014.

[19] Y. Zhu, D. Wu, and S. Li, "A piecewise linear representation method of time series based on feature points," in *Proc. Int. Conf. Knowl.-Based Intell. Inf. Eng. Syst.*, 2007, pp. 1066–1072.

[20] E. J. Keogh and M. J. Pazzani, "A simple dimensionality reduction technique for fast similarity search in large time series databases," in *Proc. Pacific-Asia Conf. Knowledge. Discovery Data Mining*, 2000, pp. 122–133.

[21] N. T. Nguyen, B. Trawiński, R. Katarzyniak, and G.-S. Jo, *Advanced Methods for Computational Collective Intelligence*. Berlin, Germany: Springer, 2012, vol. 457.

[22] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Inf. Sci.*, vol. 239, pp. 142–153, 2013.

[23] M. G. Baydogan and G. Runger, "Time series representation and similarity based on local autopatterns," *Data Mining Knowl. Discovery*, vol. 30, no. 2, pp. 476–509, 2016.

[24] E. Keogh, J. Lin, and A. Fu, "Hot sax: Efficiently finding the most unusual time series subsequence," in *Proc. 5th IEEE Int. Conf. Data Mining*, 2005, pp. 8–pp.

[25] X. Xi, E. J. Keogh, L. Wei, and A. Mafra-Neto, "Finding motifs in a database of shapes," in *Proc. Int. Conf. Data Mining*, 2007, pp. 249–260.

[26] C. D. Stylios and V. Kreinovich, "Symbolic aggregate approximation (SAX) under interval uncertainty," in *Proc. Annu. Conf. North Amer. Fuzzy Inf. Process. Soc. held Jointly 5th World Conf. Soft Comput.. IEEE*, 2015, pp. 1–7.

[27] B. Lkhagva, Y. Suzuki, and K. Kawagoe, "Extended SAX: Extension of symbolic aggregate approximation for financial time series data representation," in *Proc. Data Eng. Workshop*, vol. 4A-i8, 2006.

[28] Y. Sun, J. Li, J. Liu, B. Sun, and C. Chow, "An improvement of symbolic aggregate approximation distance measure for time series," *Neurocomputing*, vol. 138, pp. 189–198, 2014.

[29] C. T. Zan and H. Yamana, "An improved symbolic aggregate approximation distance measure based on its statistical features," in *Proc. 18th Int. Conf. Inf. Integr. Web-Based Appl. Serv.*, 2016, pp. 72–80.

[30] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing sax: A novel symbolic representation of time series," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 107–144, 2007.

[31] S. Kolozali, D. Puschmann, M. Bermudez-Edo, and P. Barnaghi, "On the effect of adaptive and non-adaptive analysis of time-series sensory data," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1084–1098, 2016.

[32] P. Senin and S. Malinchik, "Sax-VSM: Interpretable time series classification using sax and vector space model," in *Proc. IEEE 13th Int. Conf. Data Mining*, 2013, pp. 1175–1180.

[33] T. Rakthanmanon and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *Proc. SIAM Int. Conf. Data Mining*, 2013, pp. 668–676.

[34] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning time-series shapelets," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 392–401.

[35] M. Shah, J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning DTW-shapelets for time-series classification," in *Proc. 3rd IKDD Conf. Data Sci.*, 2016, Art. no. 3.

[36] L. Hou, J. T. Kwok, and J. M. Zurada, "Efficient learning of timeseries shapelets," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 1209–1215.

[37] J. Paparrizos and L. Gravano, "k-shape: Efficient and accurate clustering of time series," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2015, pp. 1855–1870.

[38] A. Bagnall, J. Lines, J. Hills, and A. Bostrom, "Time-series classification with COTE: The collective of transformation-based ensembles," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 9, pp. 2522–2535, Sep. 2015.

[39] M. Sayed-Mouchaweh, *Learning from Data Streams in Dynamic Environments*. Berlin, Germany: Springer, 2016.

[40] D. Puschmann, P. Barnaghi, and R. Tafazolli, "Adaptive clustering for dynamic iot data streams," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 64–74, Feb. 2017.

[41] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surveys* [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6703726, vol. 46, no. 4, 2014, Art. no. 44.

[42] C. Lifna and M. Vijayalakshmi, "Identifying concept-drift in twitter streams," *Procedia Comput. Sci.*, vol. 45, pp. 86–94, 2015.

[43] C. M. Grinstead and J. L. Snell, *Introduction to Probability*. Providence, RI, USA: American Mathematical Society, 2012.

[44] M. Ali-ud-din Khan, M. F. Uddin, and N. Gupta, "Seven v's of big data understanding big data to extract value," in *Proc. Zone 1 Conf. Amer. Soc. Eng. Edu.*, 2014, pp. 1–5.

[45] Z. Wang, "Fast algorithms for the discrete W transform and for the discrete fourier transform," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 32, no. 4, pp. 803–816, Aug. 1984.

[46] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. Cambridge, MA, USA: Academic Press, 2014.

[47] G. K. Wallace, "The jpeg still picture compression standard," *IEEE Trans. Consumer Electron.*, vol. 38, no. 1, pp. xviii–xxxiv, Feb. 1992.

[48] A. C. Bovik, *The Essential Guide to Image Processing*. Cambridge, MA, USA: Academic Press, 2009.

[49] G. Bi and Y. Zeng, *Transforms and Fast Algorithms for Signal Analysis and Representations*. Berlin, Germany: Springer, 2004.

[50] J. Makhoul, "A fast cosine transform in one and two dimensions," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 28, no. 1, pp. 27–34, Feb. 1980.

[51] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The UCR time series classification archive," Jul. 2015, [Online]. Available: www.cs.ucr.edu/ eamonn/time_series_data/

[52] L. Ye and E. Keogh, "Time series shapelets: A new primitive for data mining," in *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 947–956.

[53] D. Puschmann, "Random lhs generator drift," 2016. [Online]. Available: https://github.com/UniSurreyIoT/random-LHS-generator-drift

[54] R. Briandet, E. K. Kemsley, and R. H. Wilson, "Discrimination of arabica and robusta in instant coffee by fourier transform infrared spectroscopy and chemometrics," *J. Agricultural Food Chemistry*, vol. 44, no. 1, pp. 170–174, 1996.

[55] L. M. Davis, "Predictive modelling of bone ageing," Ph.D. dissertation, University of East Anglia, Norwich, Norfolk, 2013.

[56] A. Anguera, J. Barreiro, J. Lara, and D. Lizcano, "Applying data mining techniques to medical time series: An empirical case study in electroencephalography and stabilometry," *Comput. Structural Biotechnology J.*, vol. 14, pp. 185–199, 2016.

[57] Y. Yang, Q. Yang, W. Lu, J. Pan, R. Pan, C. Lu, L. Li, and Z. Qin, "Preprocessing time series data for classification with application to crm," in *Proc. Australasian Joint Conf. Artifi. Intell.*, 2005, pp. 133–142.

[58] A. González-Vidal, V. Moreno-Cano, F. Terroso-Sáenz, and A. F. Skarmeta, "Towards energy efficiency smart buildings models based on intelligent data analytics," *Procedia Comput. Sci.*, vol. 83, pp. 994–999, 2016.

[59] S.-W. Lin, K.-C. Ying, S.-C. Chen, and Z.-J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector machines," *Expert Syst. Appl.*, vol. 35, no. 4, pp. 1817–1824, 2008.

[60] C. A. Ratanamahatana and E. Keogh, "Three myths about dynamic time warping data mining," in *Proc. SIAM Int. Conf. Data Mining*, 2005, pp. 506–510.

[61] D. Lemire, "Faster retrieval with a two-pass dynamic-time-warping lower bound," *Pattern Recognit.*, vol. 42, no. 9, pp. 2169–2180, 2009.

[62] Z. Bar-Joseph, A. Gitter, and I. Simon, "Studying and modelling dynamic biological processes using time-series gene expression data," *Nature Rev. Genetics*, vol. 13, no. 8, pp. 552–564, 2012.

[63] M. V. Moreno, F. Terroso-Saenz, A. Gonzalez, M. Valdes-Vela, A. F. Skarmeta, M. A. Zamora-Izquierdo, and V. Chang, "Applicability of big data techniques to smart cities deployments," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 800–809, Apr. 2017.

[64] C. Costa and M. Y. Santos, "Improving cities sustainability through the use of data mining in a context of big city data," in *Proc. Int. Conf. Data Mining Knowl. Eng.*, 2015, vol. 1, pp. 320–325.

[65] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proc. 8th ACM SIGMOD Workshop Res. Issues Data Mining Knowledge Discovery*, 2003, pp. 2–11.

[66] M. Hirzel, H. Andrade, B. Gedik, G. Jacques-Silva, R. Khandekar, V. Kumar, M. Mendell, H. Nasgaard, S. Schneider, R. Soulé, et al., "Ibm streams processing language: Analyzing big data in motion," *IBM J. Res. Develop.*, vol. 57, no. 3/4, pp. 7–1, 2013.

[67] L. Li, F. Noorian, D. J. Moss, and P. H. Leong, "Rolling window time series prediction using MapReduce," in *Proc. IEEE 15th Int. Conf. Inf. Reuse Integr.*, 2014, pp. 757–764.

[68] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2016. [Online]. Available: https://www.R-project.org/

Aurora González Vidal received the graduated degree in mathematics from the University of Murcia, in 2014. In 2015, she received a fellowship to work in the Statistical Division of the Research Support Services, where she specialized in statistics and data analyisis. Since 2015, she has been working toward the PhD degree in computer science, focusing her research on data analytics for energy efficiency and studied for a master's degree in Big Data. She was a visiting PhD student at the University of Surrey, where she worked on the study of segmentation of time series.

Payam Barnaghi is a reader in machine intelligence in the Institute for Communication Systems Research with the University of Surrey. He was the coordinator of the EU FP7 CityPulse project on smart cities. His research interests include machine learning, the Internet of Things, the Semantic Web, adaptive algorithms, and information search and retrieval. He is a senior member of the IEEE and a fellow of the Higher Education Academy.

Antonio F. Skarmeta received the BS (Hons.) degree in computer science from the University of Murcia, the MS degree in computer science from the University of Granada, Spain, and the PhD degree in computer science from the University of Murcia. He is a full professor with the Department of Information and Communications Engineering, University of Murcia. He is involved in numerous projects, both European and National. Research interests include mobile communications, artificial intelligence, and home automation. He is a member of the IEEE.