To my grandparents

POLITECNICO DI TORINO

Department of Control and Computer Engineering

Master of Science in Computer Engineering

Master's Degree Thesis

Expert Knowledge for Unsupervised Machine Learning in Energy Time Series

Improving existing pattern recognition algorithms by enriching them with expert knowledge.

Supervisors:
Prof. Luca Cagliero
Prof. Veit HagenMayer

Author:
Andrea Misuraca

Accademic Year 2018 – 2019

Abstract

The use of machine learning techniques has spread in the last decades, allowing us to accomplish tasks like regression, classification, clustering and to apply these algorithms in various areas. At the same time we assisted to an increasing availability of time series data.

Applying data mining techniques to time series for pattern discovery is hard because we deal with a huge amount of data that affects the algorithm performances, and because we do not have a hand-labeled set to estimate our model. The only way to perform evaluation is to ask the opinion of experts. The feedback from the experts' knowledge is not useful just as a validation method but it can be reintegrated in the main algorithm in order to adjust the results.

The aim of this thesis is to implement a pattern recognition algorithm, to evaluate it using the knowledge of experts and to feed this knowledge back in the main process.

Acknowledgements

I take this opportunity to express gratitude to all the people that I met during the whole course of study and my lifelong friends for their help and unceasing support.

First, I have to thank my thesis supervisors Prof. Veit HagenMayer and Ms. Sc. Nicole Ludwig from Karlsruhe Institute of Technology. Without their assistance and dedicated involvement in every step throughout the thesis, this work would have never been accomplished. I would also like to thank my home university advisor Prof. Luca Cagliero. He was always available whenever I ran into a trouble spot or had a question about my research. I would like to thank you very much for your support and understanding over these past six months.

I would also like to acknowledge my university colleagues, with whom I shared both happy and bad moments during these five years. I am grateful to all the amazing people I met both in Turin and in Karlsruhe, during my Erasmsus experience, because they always encouraged me, making me feel like at home. And a special thanks to Martina Merlonghi for her patience and assistance during the period abroad.

Finally, I must express my very profound gratitude to my family, especially my parents, for providing me with unfailing support and continuous encouragement and attention throughout my years of study. This accomplishment would not have been possible without them. And to my grandparents, that always dreamed of seeing me with a laurel wreath, thank you.

Andrea

Contents

List of Tables

List of Figures

There is in all things a pattern that is part of our universe. It has symmetry, elegance, and grace.

[F. Herbert, Dune]

Part I

General Introduction

Chapter 1

Introduction

Recent data acquisition techniques are able to collect a huge amount of data. It is impossible to analyze them manually, for that reason we need to implement new software that helps us. Machines use both calculations and data retrieval to perform operations similar to the human brain, they learn first on a predefined data set, and finally they use the acquired knowledge to make decisions. This branch of computer science that merges both informatics and data analysis is named machine learning, and it gained a lot of attention in the literature in the last years after the availability of fast hardware able to process the big quantity of data collected and the availability of data itself.

Machine learning algorithms became very popular in recent times, both for the ability to produce a good result in a short time and for the possibility to be applied to different kinds of data. Nowadays almost every machine in the world stores data, locally or remotely, and often every single data is related to a time-stamp. Chronological sequences are the basis of time series. Time series data comes, for example, from the natural sciences (for example earthquake data produced by seismology or sensors data), from the medical field (as electrocardiograms or extracted DNA sequences), from geography, finance, engineering, statistics, speech and handwriting recognition. The great quantity of applications is what leads us to dive deep into the data analysis area.

The main task of this thesis is to develop an algorithm that is able to find patterns in time series data. This goal is not easy to reach, due to, for example, the dimension of the data. We want to handle trillions of points (e.g. think about a sensor that saves a value every five seconds, and imagine it runs for one year or more). So having an algorithm that performs well in terms of both time and accuracy is not easy. In [19] the authors describe the state of the art for representing time series, which we will us as starting point for further analysis.

The discovery process is difficult to accomplish also because we do not have any clues about the nature of the motifs we are going to find. What is their shape? What is their length? Do they all have the same length? In [27] the authors provide an approach useful to investigate motifs with such unknown properties.

Furthermore, we want to take a look at the data mainly used in this thesis. It is an industrial data set measuring the energy consumption of different machines. A method to retrieve patterns from such a data-set can also be found in [20].

Finally yet importantly, the evaluation of our work is one of the hardest parts of the whole thesis. How can we validate our result? Is it good enough or should we improve it somehow? We do not have any ground truth to compare it with. Our data is unlabeled; we are using what is defined as unsupervised machine learning. Here, the most interesting part comes. The only way to know if the found patterns are correct is to ask the opinion of an expert. The expert must be a person from the energy-engineering field and not a computer scientist, so part of our duty includes finding a way to ask the expert what the best parameters of our algorithm are. Our research brings us to wonder what knowledge itself (as it is explained in [1]) is. Or what does happen if the experts' judgments are wrong?

In our work we analyze some labeled data-sets to verify the functionality of the implemented algorithms. We start normalizing our data, and we preprocess it using a dimensionality reduction technique. Then we use a symbolic representation to discretize our continuous real-valued series, converting them into strings. After this phase, in our experiments we test our fixed-length patterns discovery algorithm, and later we extend it to discover motifs with variable length.

After estimating the performances in terms of error rate using labeled data, we run the algorithm with a real unlabeled data-set. This data comes from the energy industry and to estimate the error of our method we ask the opinion of an expert from the same domain of the data. Using knowledge elicitation techniques, we also want to feed the experts' estimations back in the main algorithm, because with his information we may be able to have a better parameters setting in order to improve the quality of our results.

In our experimental evaluation we show the good accuracy of our approach and how it has improved after the integration of experts' opinion. At the same time, we demonstrate how competitive our technique is from the computational time point of view.

Motivation comes from the usefulness of time series mining too. Patterns are everywhere, they are all around us, discovering them is just a way to understand the intrinsic properties of data and to get advantages from that.

The rest of this thesis is organized as follows. Section 2 briefly discusses related work. Section 3 introduces our methods for mining time series discussing their main characteristics. Section 4 talks over the experts knowledge elicitation. Section 5 contains an experimental evaluation of our approaches on a variety of data and Section 6 discusses the obtained results. Finally, Section 7 offers some conclusions and suggestions for future work.

Part II

State of the Art Analysis

Chapter 2

Related Work

The task of finding patterns in time series data has been an area of active research that involved academics from different fields and not only researchers from the computer science community. A great contribution has been given in the last years, in unison with the exploding of time sequences availability.

2.1 Data Mining Tasks with Time Series Data

In [9] the author summarizes the main characteristics of time series data and the most popular techniques for mining them. He explains how our data is a collection of observations made chronologically. Our input is a class of temporal data objects, characterized by huge size, high dimensionality and with a numerical and continuous nature. The goal is to apply typical data mining tasks, such as clustering, classification, rule discovery and summarization. The most popular and classical algorithms are illustrated, both for preprocessing time series and mining them. Finally, the importance of similarity measures is explained, and different measures are compared. We have many alternatives, and the choice depends on the target we want to reach. Classical distance measures are exploited to handle whole sequence matching or subsequence matching. And each may have either equal or different length.

2.2 Classifying Time Series

Our work starts with the implementation of a classification algorithm. This choice is made to have a base algorithm that performs well though, to evaluate it, we need labeled data (so we can say if our series has been classified right or not) and some performance references to compare our results. In [2] it is possible to find an analysis of the most common methods for classifying time series. 18 classical algorithms, implemented in Java, are exploited over 85 datasets and evaluated comparing them against standard benchmark classifiers. Due to the availability of their results and experiments, we have the basis for defining the performances of our code.

Talking about performance, it is necessary to remind that the main problem of dealing with big data is the computational time. [26] underline how similarity join (known also as all-pairs-similarity-search) can take months using a trivial algorithm, such as the obvious nested loop, even with indexing or pruning techniques. It is tough due to the daunting nature of the data. This kind of problems are the main causes of a slowdown in the research: very little progress has been made in the specific area of subsequences retrieval techniques, though the interest is high. The authors do not just state the problem, but they offer us also a very innovative algorithm that produces a high-quality approximate result in a reasonable time, speeding it up using a GPU framework. They conclude their thesis demonstrate their results, accomplishing data mining tasks using input coming from different domains.

2.3 Mining Motifs in Time Series

Entering in the details of our specific task, that is pattern discovery in time series, we cannot avoid citing Prof. Keogh, from University of California – Riverside, and his colleagues. Their work is useful not just for this thesis, but for the whole research community. In [6] they implement a fast algorithm for finding similar subsequences in a time series. After explaining how many data mining problems can be reduced to find repeated subsequences in a longer time series, they highlight how this can be difficult, not only from the computational time point of view, but for the presence of noise too. The result they obtain is approximate, but accurate too, especially if we consider that there is no prior knowledge integrated in the algorithm. They emphasize how time series can be misclassified when noise is present, because a single piece of it can dominate the distance function. They base their method on a patterns discovery algorithm for DNA subsequences and on the Buhler and Tompa projection algorithm proposed in [22], and they show that it correctly classifies series that were erroneously categorized previously. The efficiency of their process and its sensitivity, either to noise or to “don’t care” symbols, is shown in their experimental result.

The same team of researchers in [14] explains the utility of finding surprising patterns in time series, and the methodologies to do that. It is a topic not as common as subsequence matching, so the lack of attention leads the authors to starting from the bottom, giving a new definition of what a surprising pattern is. The interest is not in surprise data points but in special patterns, so with special structure or frequency. The originality of this publication consists of demonstrating how a data structure like a suffix tree can efficiently encode the frequency of all the observed patterns. They also show how to calculate the expected frequency of unobserved patterns with Markov models, and finally they succeeded in computing the measure of surprise in time linear in the size of the database.

Surprising patterns are a special category of patterns, but they are not the only ones. Usually, we investigate for full periodic motifs, where every point in time contributes to the periodicity. By the way, in the real life, there are many more problems related to partial periodic patterns, where just part of the data points compose the period. In [10] a list of algorithms for mining partial periodic patterns is presented, with their main characteristics. Their performances are compared, and it is shown how the max-subpattern hit set method, which needs only two scans of time series database, offers excellent results, even for mining multiple periods. They dwell on the issues that we may encounter, such perturbation, and they conclude with a list of possible investigations to take in the future.

2.4 Time Series Representation

Probably the most significant contribution for our work is given by [19], where the authors present the state of the art for representing time series. Often the preprocessing part is underestimated, but this innovative representation allows dimensionality and numerosity reduction, and a discrete sequence as result. What brings our interest in such a discretization process is the fact that most of the studies assume that our data is discrete. Computer Engineers prefer working with this kind of information, even if most of the data coming from the real world is real valued.

The question that comes next is: what if we want to use the SAX for finding different length patterns? We can see the usefulness in many real world situation, though we have to struggle because of the long execution time of many methods. By the way, in [27] a new and fast technique to retrieve different length motifs is presented. They used a hybrid approach, taking advantage from both exact and approximated methods, using Positional Inverted Index based on SAX. They also apply to the found index a procedure called Motif Refinement, to remove repeated or overlapped patterns. It is necessary to point out that the starting point of a motif and its length are fast to identify, but the verification of the pattern, from the original time series still requires a longer time due to the Dynamic Time Warping distance measure used to compare the different length patterns.

2.5 Energy Time Series

In this specific thesis, what we want to do is to deal with energy time series. Motifs discovery algorithms have already been applied many times to different real-world problems, but it is difficult to find applications to energy data. One example is reported in [20], where the authors try to identify patterns with the aim of classifying processes according to their degree of potential flexibility. The different approach from classical algorithms consists of a new two-stage algorithm: they first identify the starting point of a pattern using an event search algorithm, and then they apply a motif discovery procedure to identify the events, even of different length, triggered by the same process.

They analyzed the data without any technical knowledge about the processes' properties, in order the flexibility to drive the result, while what we want to obtain is to analyze the data including some technical knowledge furnished by an expert of the sector.

2.6 External Knowledge Integration

The integration process of some new knowledge in an algorithm might not be clear. The starting point is to describe what knowledge itself is, and it may not be easy for a computer engineer. In [1] a reviewed definition of wisdom as expert knowledge is provided. Someone could argue that it is a philosophical critic, more than a technical description, but it can still be useful to contextualize such an ancient concept for contemporary jobs, even if it is not from a scientific point of view.

Instead, a more technical example of a Knowledge Discovery process is shown in [24]. The authors, at first, state the problem they want to solve, formulating it in natural language, and then they translate it in a more formal way. The interesting part of this description consists of the inclusion of a priori knowledge in the process, in order to get a better result without wasting the information we may have from the beginning. Furthermore, two more stages are illustrated: evaluation and visualization. These phases are useful as feedback for the user, the evaluation is performed to allow the user to optimize his own algorithm, while the visualization step is the only way to let the analyst interact with a huge amount of data. It is an essential support to get perception and a better understanding of our data.

A practical example of the use of prior knowledge is proposed in [16], where they try to predict daily stock prices using neural networks. Of course, the powerfulness of neural networks is exploited in their algorithm, due to the nonlinear relationship between the inputs and the desired outputs they are able to learn. By the way, we want to concentrate more on the quality of the result, which is improved with the embedding of a priori knowledge. In this case, it consists of economic indicators, political and economic events, taken from newspapers headline. The authors dwell on how taking real-world events is interesting, because they are non-numerical factors and on how their result is good with this integration.

Even if a priori knowledge is fundamental for getting good results, the difference with our work consists of incorporating the knowledge from the experts even after the algorithm already run once. Receiving feedbacks, after an intermediate result has been produced, is worth for evaluating it, but it is more useful if we can incorporate this assessment, to increase the accuracy.

Part III

Motifs Mining in Time Series

Chapter 3

Motivation

Given an overview of previously published papers on the topic, it is natural to ask: why is experts' knowledge integration important? As introduced above, temporal objects belong today to many scientific fields, and retrieve accurate patterns with unlabeled data has applications in different domains.

We want to show how we can implement a fast and accurate motifs discovery algorithm, able to run with label and unlabeled data. After that, we utilize unlabeled data to simulate what happens with real-time series. In the literature is quite easy to find publications regarding patterns retrieval methods, while it is harder to find knowledge elicitation techniques related to scientific purposes. The combination of the two is essentially absent, that is why we want to give a contribution, implementing an innovative algorithm able to find motifs and enhanced with the experts' opinion, showing accuracy and performances.

The chronological order is the main characteristic of time series data, but they are also characterized by a huge size, that is often complicated to handle. Our work wants to find a way to retrieve motifs that performs well in terms of results and computational time. Data often have a high dimensionality, that we try to reduce, and they are numerical continuous values (real number) so what we want to do is to preprocess them applying dimensionality reduction and discretization techniques before running our algorithm. Simplifying the complexity of the problem spending little time to preprocess the data lets us gain much more time during the execution phase of the main algorithm. The methods we adopt, and illustrated in the following chapters, represent the state-of-the-art of time series mining.

Chapter 4

Methods

4.1 Symbolic Aggregate approXimation

We have many possibilities to operate on time series data, from the preprocessing phase until the knowledge discovery part. The choice is huge and difficult too, because every algorithm has its own advantages and drawbacks. After analyzing many opportunities described in [15], to explore our data we consider to start preprocessing them using the SAX (Symbolic Aggregate approximation) representation stated in [19]. The main reason is that for computer scientists operate on symbolic data (such as strings) is easier, due to the quantity of algorithms and data structures available. The problem that symbolic representations used to face was to find a way to overcome the continuous nature of time series data, and after a discretization process, to define an opportune distance measure. The absence of an adequate distance measure in many other representations, like those described in [13],[12], is what generated lack of interest in the data mining community in a symbolic representation of the data. In addition, to justify our choice we also considered that Keogh et al. in [19] succeed in finding a measure that lower bound the time series distance in the original space. Having a suitable distance measure is essential for accomplish every kind of data mining task.

The SAX representation seems the best way to analyze our data, it offers all the features we need for mining time series as described in [14],[18], included dimensionality reduction.

4.1.1 Tasks and Normalization

Before running our preprocessing algorithm with an unlabeled dataset, we test it using labeled ones taken from ”UCR Time Series Classification Archive” from University of California - Riverside, [15]. What we want to accomplish with these experiments is a classification task [12].

The first step consists of normalizing the time series. It is known that is worthless to compare time series if they have different amplitudes and offsets, which is why we want to normalize them in order to get a standard deviation equal to one and a mean value equal to zero. After that the time series will have a Gaussian distribution as explained in [17].

4.1.2 Dimensionality Reduction

Now that we have a normalized time series, we want to precede reducing the dimensionality. Keeping the original number of values could affect the performance of our algorithm, slowing it down. In order to solve this problem we transform the data into the Piecewise Aggregate Approximation (PAA, from [13]). Essentially, we decide a priori the length of the future transformed series. Depending on this parameter, we divide our series in equal segments, and for each segment, we compute the mean value inside this window.

The more rigorous way to define the computation of each frame is the following:

$$\overline{c_i} = \frac{w}{n} \sum_{j=\frac{w}{n}(i-1)+1}^{\frac{n}{w}i} c_j$$

where $C = c_1, c_2, ..., c_n$ is the original time series of length n, while $\overline{C} = \overline{c_1}, \overline{c_2}, ..., \overline{c_w}$ is the approximated time series of length w = number of segments.

As descried in the figure 4.1, we could think of the performed approximation as a linear combination of box bases function.

4.1.3 Discretization

A reduced time series is not enough, what we want to do now is to change the domain, from continuous numeric values to discrete symbols. The SAX representation consists of transform the numeric values into letters, that concatenated will form a string of characters.

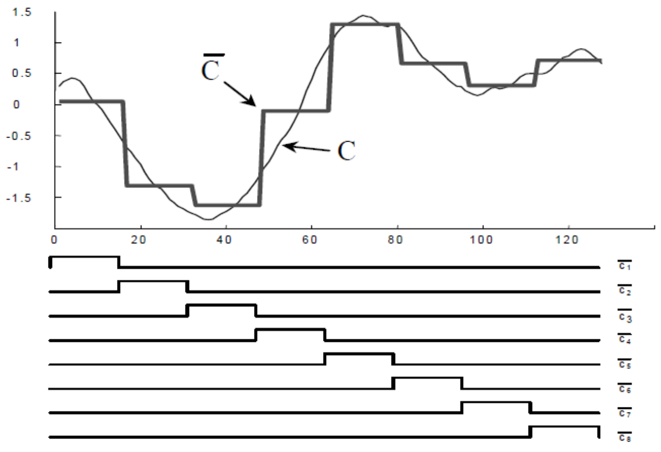


Figure 4.1. PAA representation, the original sequence C of length 128 is reduced in $\overline{C}$ of length 8. Source: [19].

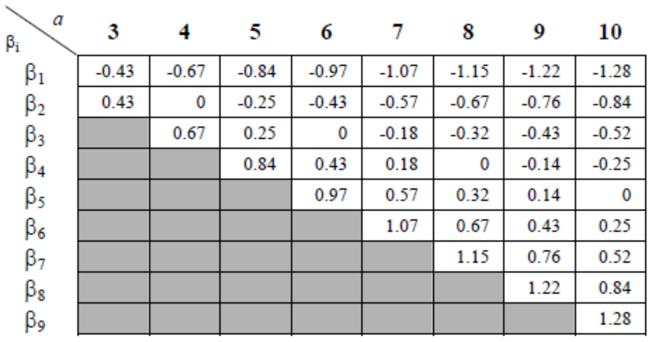
The string will have a length equal to the number of segments obtained from the PAA, and the number of different characters in the string is chosen a priori. We call this parameter ‘Alphabet Size‘, and the user will define it at the beginning of the discretization process. E.g.: if the alphabet size is equal to 3 the letters we will use to build our string will be: a, b, c; if it is equal to 10: a, b, c, d, e, f, g, h, i, j.

What we want to find now is a successions of breakpoints. Comparing each value of $\overline{C}$ with the breakpoints, we will be able to transform it in a letter. Usually they are not fixed values, but they depend on the distribution of the data. By the way, as [17] explained, normalized time series have Gaussian distribution, so we can define them fixed. The number of breakpoints depends only on the alphabet size a. Knowing that the area under our distribution curve is equal to one, we need just to divide it in a equal-sized areas. Each cutting point of the x axe is a breakpoint. It easy is to find them due to the availability of the Standard Normal Table. In figure 4.2 is reported a list of breakpoints with an alphabet size from 3 to 10.

Given an alphabet, we define as α_i the i^{th} element of the alphabet (e.g. $\alpha_2 = b$). β_j is the j^{th} breakpoint, $\overline{c_i}$ is the i^{th} element of the reduced time series and $\hat{c}_i$ is the obtained character.

$$\hat{c}_i = \alpha_i, iff \beta_{j-1} <= \overline{c_i} < \beta_j$$

where $\hat{c}_i$ is a letter and $\hat{C}$ is the full word. Figure 4.3 illustrate the discretization process.



β_i \ a	3	4	5	6	7	8	9	10
β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
β_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
β_3		0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
β_4			0.84	0.43	0.18	0	-0.14	-0.25
β_5				0.97	0.57	0.32	0.14	0
β_6					1.07	0.67	0.43	0.25
β_7						1.15	0.76	0.52
β_8							1.22	0.84
β_9								1.28

Figure 4.2. Breakpoints obtained with different alphabet sizes. Source: [19].

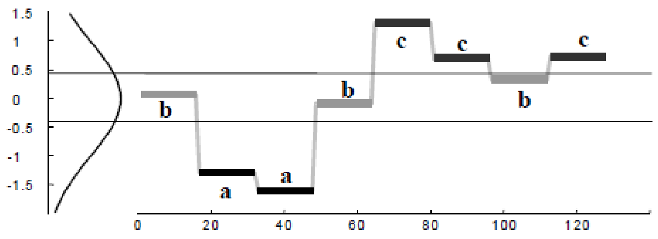


Figure 4.3. Discretization after PAA is applied. $n = 128$, $w = 8$, $a = 3$. The result is the word ”baabccbc”. Source: [19].

4.1.4 $MINDIST$ Distance Measure

Keogh in [19] gives the details of every single step. After computing a sequence of characters, we want to operate with it, in particular we would like to compare it with another similar string. The authors of [19] delineate a new distance measure called $MINDIST$ define as follow:

$$MINDIST(\hat{Q},\hat{C}) = \sqrt{\frac{n}{w}}\sqrt{\sum_{i=1}^{w}(dist(\hat{q}_i,\hat{c}_i))^2}$$

Where all the symbols have the same meaning expressed in the previous paragraph. The usefulness of this function comes from the fact that it returns the minimum distance between the original time series. The only thing we still need to explain is how to implement the function $dist()$.

The $dist()$ function returns the distance between two letters, and it is given after checking a lookup table. The distance value between every couple of characters is stored in a symmetric matrix. It is computed only once and it depends only on the alphabet size. The value of each $cell_{i,j}$

is computed as follows:

$$cell_{i,j} = \begin{cases} 0 & \text{if } |i-j| <= 1 \\ \beta_{max(i,j)-1} - \beta_{min(i,j)} & \text{else} \end{cases}$$

In the following figure, we can see the distance between two time series in different domain:

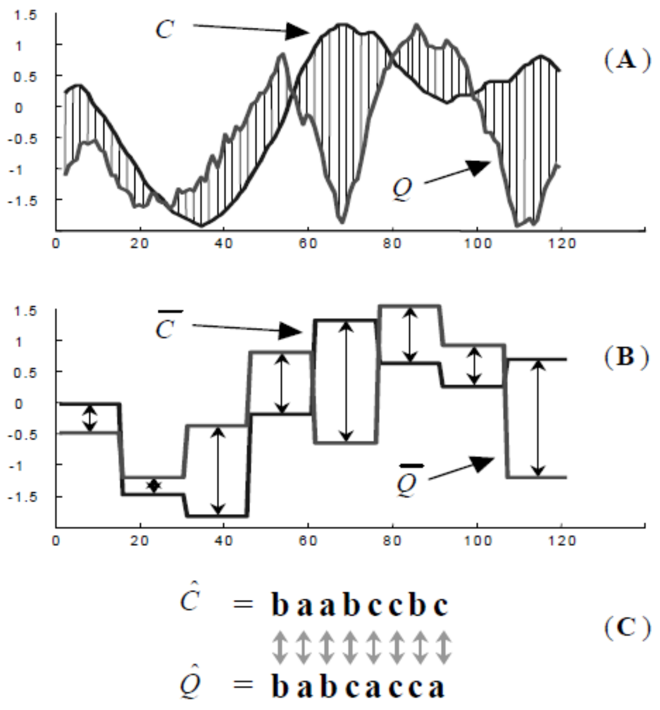


Figure 4.4. A) Original domain, B) PAA representation, C) SAX representation. Source: [19].

4.2 *k*-Nearest Neighbor

In order to perform some tests, we utilize labeled data. In this way, we can comprehend accuracy and performances. The task to accomplish is to classify the time series from our data set according to their similarity. To do that we use the popular and classic k-NN, with different parameters. Our implementations in pseudo-code is illustrated in algorithm 1.

Having some labeled data, it labels the new points based on the closest k neighbors. In our own implementation, using the $MINDIST$ distance measure we compute the distance between a new

Algorithm 1 k-NN

```
INPUT: $k$, $X$:training data, $Y$:labels, $x$:unknown sample
OUTPUT: label $Y_i$ of the nearest neighbors
  for $i = 1$ to $i = m$ do
    Compute distance $d(X_i, x)$
  end for
  Compute sei $I$ containing indices for the $k$ smallest distances $d(X_i, x)$
  return majority label for $[Y_i$ where $i \in I]$
```

time series that needs to be classified, belonging to a test set, and a bag of words, so a group of time series already classified, belonging to a train set.

4.3 Distance Measures

Struggling with parameters is not easy, but it is not the only hard choice to take. Even if for the SAX representation described in [19] is suggested to adopt the $MINDIST$ distance measure, we have to say that for testing our algorithm, without the SAX representation, is much easy to exploit the Euclidean distance:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \ldots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$$

This measure is really popular because it is easy to implement and it has is fast too.

The other choice we have is to adopt the Dynamic Time Warping (DTW) to compute the similarity between two temporal sequences. By the way, it requires much more time because of its complexity that can be approximated to $\mathcal{O}(N^2)$.

The last option is to use the Levenshtein distance (often known just as Edit distance) with the SAX representation. It gives us a way to compare two strings and returns a number proportional to how different they are. Briefly it returns the numbers of edits (that is why its name) necessarily to transform one word into the other one.

This new distance is defined as follow:

$$lev_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{a_i \neq b_i} \end{cases} & \text{else} \end{cases}$$

Where a and b are the two strings we want to measure, of length $|a|$ and $|b|$ respectively. $lev_{a,b}(i,j)$ is the distance between the first i letters of a and j letters of b, and $1_{a_i \neq b_i}$ is a function that will output 0 if $1_{a_i = b_i}$, 1 otherwise.

In algorithm 2 explains how it works.

There are different implementations, but we focus on the iterative one, because it is faster than the recursive one. The recursive algorithm is very unsuitable because it computes the distance between the same sub-strings multiple times.

Algorithm 2 Levenshtein Distance

INPUT: s:first string og length m, t:second string of length n
OUTPUT: edit distance

```
Declare matrix $d[m,n]$
**for** $i = 1$ **to** $i = m$ **do**
  $d[i,0] = i$
**end for**
**for** $j = 1$ to $n$ **do**
  $d[0,j] = j$
**end for**
**for** $j = 1$ **to** $j = n$ **do**
  **for** $i = 1$ **to** $i = m$ **do**
    **if** $s[i] = t[j]$ **then**
      SubCost = 0
    **else**
      Subcost = 1
    **end if**
    $d[i,j] = min(d[i-1,j]+1, d[i,j-1]+1, d[i-1,j-1]+subCost)$
  **end for**
**end for**
**return** $d[m,n]$
```

4.4 Variable Length Motifs

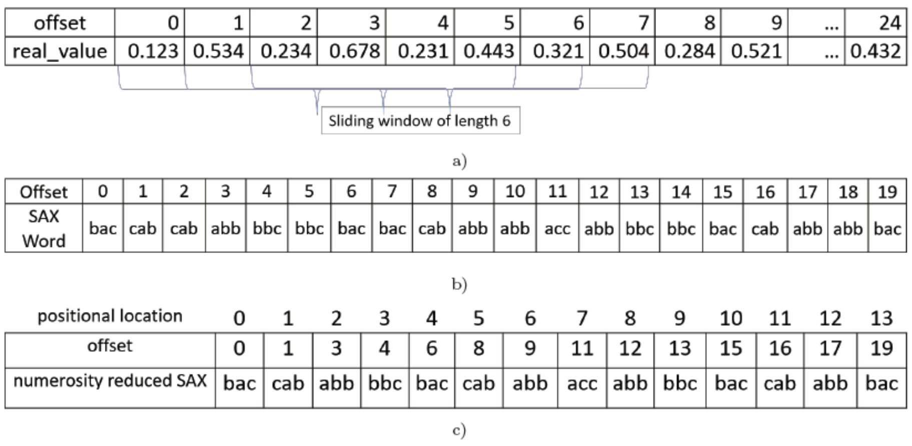
The next task is to identify patterns of different length. In many real life applications we need to deal with special (or surprising) motifs, or cyclic patterns that do not have a fixed length. Many industrial processes have some periodic, or partially periodic, motifs whose length is not fixed.

4.4.1 Numerosity Reduction

The algorithm we decide to apply is detailed described in [27] and it adopts the SAX representation described in the previous section. The use of the same method allows us to reuse part of the code we have already implemented, and at the same time to take all the advantages coming from this model.

To accomplish our new task we need to add a feature called Numerosity Reduction to the previous implementation. We extract subsequences from our raw data using a sliding window, and we convert them into SAX words. If two o more words are consecutively repeated, we record just the first occurrence with its offset.

In the example in figure 4.5, it is easy to understand the discretization process and the way we use to store our sequences applying numerosity reduction.


Sliding window of length 6

offset	0	1	2	3	4	5	6	7	8	9	...	24
real_value	0.123	0.534	0.234	0.678	0.231	0.443	0.321	0.504	0.284	0.521	...	0.432

a)

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
SAX Word	bac	cab	cab	abb	bbc	bbc	bac	bac	cab	abb	abb	acc	abb	bbc	bbc	bac	cab	abb	abb	bac

b)

positional location	0	1	2	3	4	5	6	7	8	9	10	11	12	13
offset	0	1	3	4	6	8	9	11	12	13	15	16	17	19
numerosity reduced SAX	bac	cab	abb	bbc	bac	cab	abb	acc	abb	bbc	bac	cab	abb	bac

c)

Figure 4.5. a) Real-valued time series of length 25 with sliding window of length 6, b) SAX representation for each subsequence with fragment length of 3 and alphabet size of 3, c) Numerosity reduction of b). Source: [27].

To easily store the SAX words and to associate them with their respectively offsets, we utilize the dictionary data structure available in python.

4.4.2 Positional Inverted Index

Our discovery process is both exact and approximate. The first step consists of finding motifs in the discrete space, in order to speed up the procedure, but the discovered motifs are then verified in the original space, with the raw data, using a distance measure on real-values to guarantee the quality of the result.

Concerning the discovery part, we use a term-inverted-index. This kind of index is used for information retrieval, and it usually applied to discrete data, that is why we need to preprocess our data with SAX. It is a data structure that maps the content to its position and allow fast full text searches. The algorithm described in [27] adopts a positional inverted index and so do we. At the beginning we build a 1-word inverted index, then, using a more general algorithm for n-word inverted index, we build a 2-word and a 3-word index.

In algorithm 3 we describe, in pseudo-code, the procedure adopted for building a n-word index.

Algorithm 3 $n - word$ Inverted Index

```
INPUT: $1 - word$ Inverted Index, $(n - 1) - word$ Inverted Index, $minFreq$
OUTPUT: $n - word$ Inverted Index
  //Every index is a python dictionary
  $p1 = 0$
  $pn = 0$
  $result = NULL$
  $key = key_{(n-1)} + key_1$ //n-word key
  $values = NULL$ //values associated to $key$
  while $p1! = NULL$ and $p2! = NULL$ do
    if $values_1[p1] - values_n[pn] == n$ then
      $add(values, values_n[pn])$
      $p1{+}{+}$
      $pn{+}{+}$
    else if $values_n[pn] > values_1[p1]$ then
      $p1{+}{+}$
      $pn{+}{+}$
    end if
  end while
  if $|values| >= minFreq$ then
    //add values to result dictionary
    $result[key] = values$
  end if
  return $result$
```

Each index is made up of a list of keywords with their respectively starting point. Each keyword represents a pattern and the number of starting points represents its frequency. We keep only the words that occur more than a user-defined parameter, e.g.: in figure 4.6 we delete the word *acc* from 1-word Index because it does not exceed the threshold that is equal to 2.

4.4.3 Motifs Refinement

Every keyword corresponds to a candidate motif in the original series. However, we need to consider that the retrieved patterns may be overlapped.

We say that two or more motifs are horizontal overlapped if they refers to the same SAX word and one of them starts before the end of another one. Instead we say that they are vertical overlapped if they refers to different keywords, and one starts before the other ends. To solve this problem, we use a motif refinement procedure that prune the overlapped patterns both within n-word motifs (algorithm 4) and among different words motifs (algorithm 5).

Algorithm 4 Horizontal Refinement

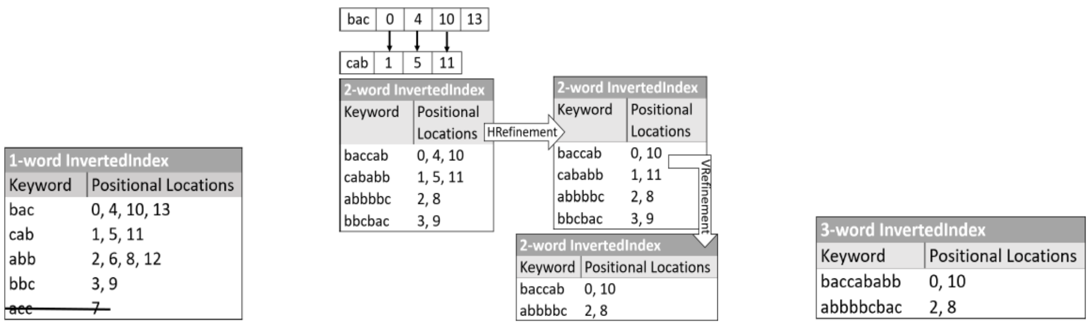
```
INPUT: $dictionary, list, n, windowSize$
// $dicrionary$ is where all the SAX words are stored
// $list$ is the temporary array containing the positional locations we want to prune
// $n$ is the paramenter of $n - word$ index
OUTPUT: $result$ //pruned positional locations
  $result = NULL$
  $add(result, list[0])$
  for $i = 1$ to $length(list) - 1$ do
    $end\_idx = list[i] - 1$ // end position+1 of current motif
    $st\_idx = list[i + 1]$ // start position of next motif
    $midx1 = dictionary.get(end_i dx) - 1$ // last offset value of current motif
    $midx2 = dictionary.get(st_i dx)$ // offset value of next motif
    if |midx2 - midx1| >= windowSize then
      $add(result, list[i + 1])$
    end if
  end for
  return $result$
```

A summary of the whole procedure is shown in figure 4.6 using the same data of the example in figure 4.5.

Algorithm 5 Vertical Refinement

INPUT: $SAXword1$, $list1$, $SAXword2$, $list2$, $minFreq$, n
// $SAXword$ and $list$ corresponds to a candidate motifs and the related positional locations
// n is the paramenter of $n - word$ index
OUTPUT: $result$ //temporary dictionary after pruning
// $result = \{SAXword$: [list of locations]$\}$

```
  $p1 = 0$
  $p2 = 0$
  if $postfix(SAXword1) == prefix(SAXword2)$ then
    $add(result, (SAXword1, list1))$
    while $p1 < |list1| and p2 < |list2|$ do
      if $list2[p2] == list1[p1] + (n - 1)$ then
        $delete(list2[p2])$
        $p1{+}{+}$
        $p2{+}{+}$
      else if $list2[p2] > list1[p1] + (n - 1)$ then
        $p1{+}{+}$
      else
        $p2{+}{+}$
      end if
      if $|list2| >= minFreq$ then
        $add(result, (SAXword2, list2))$
      end if
    end while
  end if
  return $result$
```



1-word InvertedIndex

Keyword	Positional Locations
bac	0, 4, 10, 13
cab	1, 5, 11
abb	2, 6, 8, 12
bbc	3, 9
~~acc~~	~~7~~

2-word InvertedIndex

Keyword	Positional Locations
baccab	0, 4, 10
cababb	1, 5, 11
abbbbc	2, 8
bbcbac	3, 9

2-word InvertedIndex

Keyword	Positional Locations
baccab	0, 10
cababb	1, 11
abbbbc	2, 8
bbcbac	3, 9

2-word InvertedIndex

Keyword	Positional Locations
baccab	0, 10
abbbbc	2, 8

3-word InvertedIndex

Keyword	Positional Locations
baccababb	0, 10
abbbbcbac	2, 8

a) b) c)

Figure 4.6. Motif discovery using positional inverted index: a) 1-word inverted index, b) 2-word inverted index with refinement, c) 3-word inverted index. Source: [27].

4.4.4 Motif Verification

After finding the candidates motifs, using the described approximate method that operates in the symbolic space, we want to verify them with the original time series. The proposed algorithm consists of taking the SAX motifs and their offsets, and restore them in the original space. Then we need to calculate the distances (using DTW distance measure) between every couple of found patterns. The next step will be to sort them by ascending distance and to keep the top-k similar motifs, where k is a user-defined parameter. The authors, in the last section of their research, test their algorithm on some test data-sets, highlighting the powerfulness of their study, in terms of performance. The motifs discovery process is very fast. By the way, what we obtain after that is just the SAX word of the pattern correlated with the starting point in the original series and its offset. Verifying the motif requires much more time due to the use of DTW. Even using the so-called fastDTW the result is still too slow.

To overcome the problem, one possibility may be to use different measures, like the Euclidean distance or the MINDIST described in the previous section. The problem is that we need to process our data once again, because these fast techniques work only on sequences with the same length. For example, we can either truncate the longer sequence, or apply zero padding to the shorter one. This leads to an increase of time performances, but we might lose in term of accuracy.

After a series of tests, our choice is to perform a harder selection of the candidate motifs playing with the frequency parameter of the inverted index, doing that we will have fewer motifs to compare with the DTW. And if we are still not satisfied because of the long computational time, we can also retrieve even fewer motifs from the index, and to keep them all without any comparison.

4.5 Knowledge Elicitation

We want to realize a man-machine collaborative approach, in which human and machine co-operate toward a common goal. Nowadays, many data mining techniques are based on background knowledge that drives data exploration, which means that the estimation of the machine relies on acquired experts' knowledge. The difficulty is to find a balanced interaction between the two entities, which must be complementary. For example, at the beginning the machine does not know anything about the data, so a-priori knowledge may be fundamental and, at the same time, humans cannot process huge quantity of data with the same speed of a machine. The mixed system approach is complex and difficult to realize, and we will show that it presents many drawbacks too, but the synergy between the two agents will be globally more efficient if the experts' estimations are perfectly integrated. The two systems, man and machine, share a common environment (time series data, methods for preprocessing and processing, scenarios, etc.) and a common goal, but not single tasks: the human should contribute to the machine's work, he acts asynchronously inserting his point of view, while the machine learns a new way to interpret the data every time the user puts some new knowledge.

4.5.1 Expert and Expertise

When we adopt such a methodology we need to choose the right expert. The difference among the experts is their level of expertise. In some domain is trivial to choose the experts, but other times, depending on the task and on the typology of our data, is not so obvious. The distinction among the experts is based on the position the occupy (role), on the kind of training they did, where and how they learned their job, the responsibilities they have, their status and the status of their knowledge. Based on those characteristics, as explained in [25], we can distinguish three subgroups of experts: academic, practitioner and samurai (even if an expert often embodies attributes of more groups). The academic is a person who has a theoretical understanding of the subject, and he is available to clarify and teach other people, by the way he is often far from everyday problem solving. The practitioner instead is related to everyday problem solving in his field, he has a general theoretical background, but not studied in deep. Finally, the samurai is a person who wants just to obtain optimal performances and often his only training is practice. Moreover, every experts of his subgroup has developed a different level of knowledge. The authors of [7] suggest that the knowledge evolves through the following stages: novice, advance beginner, competent, proficient and expert. Recognizing the level of knowledge is fundamental because people with a higher level are more targeted for knowledge elicitation tasks.

4.5.2 Knowledge Elicitation Techniques

The next step is one of the main challenges of the whole process: find a way to communicate the knowledge to the system. The acquisition bottleneck is one of the drawback of the procedure:

efficiently acquire the knowledge minimizing the effort and the time spent with scarce and expensive experts, and at the same time maximizing the acquired knowledge is not straightforward. There exist several techniques and the choice should be made depending on our specific purpose.

We can distinguish between two big categories of methodologies, natural and contrived. A natural method is used when an expert informally communicate its expertise, in a familiar way, while contrived methods are usually more uncomfortable for the experts but often more efficient. One of the natural methods that stands out is the interview, the most common technique. We can distinguish between unstructured, semi-structured and structured interviews. Unstructured interviews are similar to an informal conversation, where there is no agenda and there are not well prepared questions to ask the expert. It is often useful to establish a relation between the expert and the elicitor, but the lack of a structure may lead to a lack of efficiency: the expert may be verbose, he may highlight or exaggerates unimportant topics, or spend too less time on important ones. There are also semi-structured interviews, in which there is both a structure and a sort of familiarity too. A typical example is the teachback technique, where the expert explains something to the elicitor and the elicitor has to describe what he has just learnt back to the expert, in order to verify if he understood the necessary knowledge. The structured interview is the most formal method. It consists of a specific transcript prepared to avoid any waste of time, and to concentrate the experts on important topics only. Usually the procedure starts with the expert that briefly (ca. 10 minutes) describes the task, then the elicitor asks a series of specific questions. These questions are repeated and asked in depth until the elicitor does not get the full understanding of the subject.

The interviews have some weaknesses we need to mention. We can only integrate knowledge that are verbalized, what has not been explicated is lost. Furthermore, there is some kind of knowledge, like pattern recognition expertise, that is not easy to explain using the natural languages. To overcome this problem, we need to get from an expert is a series of parameters (see table 4.1).

Due to we are dealing with time series, getting information regarding the timestamps is fundamental. The frequency used for collecting data and the whole process length may help us to define some parameters, like the window size used for the SAX representation, and the variable n from n-word index for variable length patterns discovery. Another milestone for our experiments is to understand how important the values equal to zero are. As the expert said, we are dealing with the voltage of a sensor, so when it is zero, according to what he explained us, it means it is off, and it is not detecting anything. This is essential because it allows us to perform tests without considering

Name	Description
Time-stamp	Info related to period of time of sensor activity
Frequency	Frequency used by the sensor to collect the data
Process length	Length of the whole process
Zero values	Importance of values equal to zero
Max length of a motif	Maximum length of a possible pattern
Pattern frequency	How many times a pattern is repeated
Window size	Width of the window used in the SAX representation
Alphabet size	Number of characters used in the SAX representation
n-word index	Number of words to use in the index for variable length patterns
Fragment length	Length of the fragment used for PAA

Table 4.1. Parameters to ask the expert of the domain

the zero values at all, and because the expert showed us that a pattern in which most of the data are null are insignificant. Other parameters that the expert either does not know or that he learns after the data are plotted (so after a first run of the algorithm) are the maximum length of a possible pattern and its frequency. Additionally, the expert may deduce the alphabet size we want to use either because of its personal knowledge or the data or after we visualize them. The fragment size is probably the most difficult constraint to define, and the expert may not be able to explain it. By the way, as we see in the chapter IV, we can chose it empirically, performing few tests.

The method we decide to adopt to elicit the knowledge of interest is a semi-structured interview. We start establishing a relation with the expert for few minutes and we let him introduce briefly the data, from where they are, which kind of sensor we have, he can describe frequency, time stamps and what we measure (voltage). Then we run our algorithm with some default parameters and we visualize our result. Showing the plot to the expert we want him to give us an opinion and, through a more structured interview, so we can formally define the parameters in 4.1 with a specific value.

4.5.3 Handling Experts' Error

In chapter 6, we perform at the beginning some tests without the experts' opinion, and we will repeat them later, integrating external knowledge. As mentioned above, we try to give more or less the same importance both to the expert and to the machine. The first experiments, that do not have any human integration, are not used just to verify the correctness of the algorithms with random parameters, but they will be a guide for the human, and eventually they will compensate his errors too.

Handling wrong expert feedback is very difficult, especially if we are dealing with unsupervised learning. The expert that furnishes the parameters is the same person who will judge the final result, then a wrong perspective of the target may lead to wrong settings in the algorithm's values. And even if he is able to perfectly evaluate the result, he might not be able to express the parameters to adopt. Avoiding completely the possibility of receiving wrong feedback is impossible, so we decide to assume an intrinsic uncertainty, a sort of systematic error for each parameter furnished by the expert.

Each error percentage is calculated empirically, testing the labeled data with several parameters. When we will perform the experiments with the expert feedback in section 6.5 we will associate a margin of error to each given value.

Part IV

Evaluation

Chapter 5

Data

In this chapter we are going to introduce the data-sets we used to perform our tests. In order to help the reader to easily get to know our accomplishments we will adopt proper visualization methods like graphs and statistical summaries.

To perform our experiments, we run the developed algorithms with both labeled and unlabeled data-sets. The labeled ones are used for testing our methods, for verifying performances, in terms of space and time complexity, and accuracy. They consist of a series of data grouped into a train set and a test set. The first one allows us to learn how to classify our series while the other one is used for the real classification. To perform our tests with labeled data we choose the so-called CBF time series from [3] (UCR). The UCR archive is one of the most downloaded and referenced time series collection in the data mining community, since it refers to real-world problems. The aim of this gathering is to give the possibility to test new algorithms increasing accuracy. The authors of the collection furnished us additional information regarding the data and the accuracy obtained with several techniques (see table 5.1).

Name	CBF
Number of classes	3
Size of training set	30
Size of testing set	900
Time series Length	128
1-NN Euclidean Distance	0.148
1-NN Best Warping Window DTW	0.004
1-NN DTW no Warping Window	0.003

Table 5.1. Parameters provided by [3] for the test data-set

The data format consists of an integer number identifying the class and a sequence of real-valued numbers. We assume that each class of the series (so a sub-sequence) is a pattern.

CBF (Cylinder-Bell-Funnel) is built with simulated data, first defined in [23]. We cope with three classes, and each class contains standard normal noise plus an offset, as it is possible to see in fig. 5.1. We use this time series to test our equal length patterns discovery algorithm.

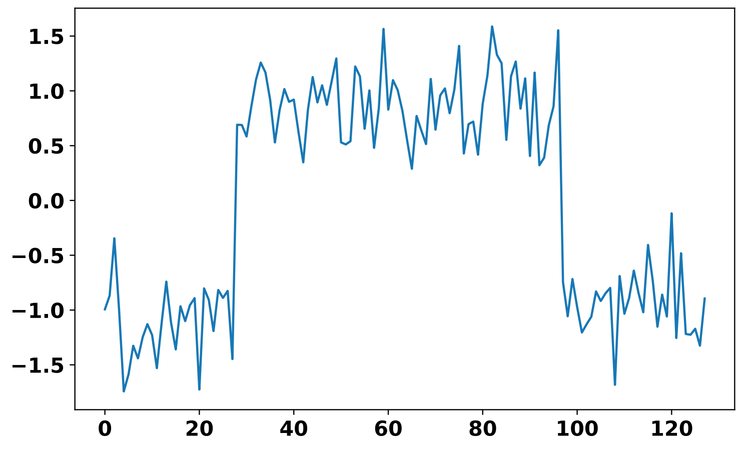


Figure 5.1. CBF time series from [3].

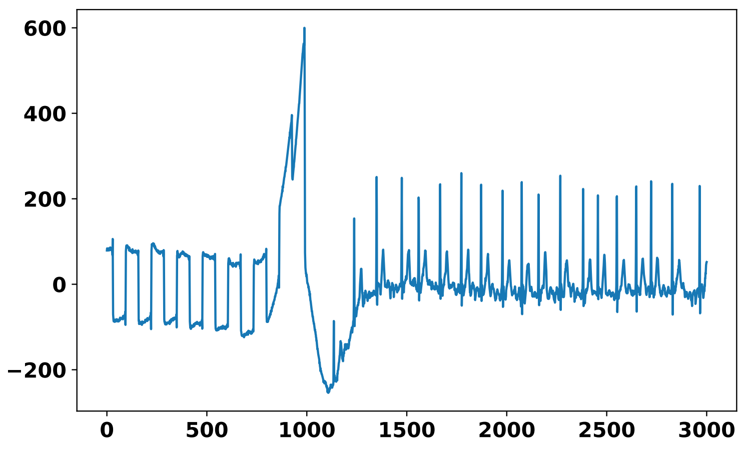


Figure 5.2. Segment from the Long-Term AF database from [21].

Instead, to check variable length motifs, we exploit an ECG database with data coming from electrocardiograms. We use the segment 71 of the Long Term Atrial Fibrillation database (fig. 5.2) described in [21] and furnished by [8]. This segment does not contain labeled data, but we can still validate our methods because we can compare our results with those obtained by [27].

Finally, after performing a series of experiments necessary to test our methods, for getting an idea of the performances and of the accuracy we may obtain, and to check the presence of eventual bugs in our code, we can run experiments with a real dataset. We use industrial data that measures the power of different machines, provided by [4] and recorded by an industrial sensor. Our data is saved in a csv file, which has the size of 256 MB and contains almost 111000 rows. Data points are collected every five seconds for one week and they are measured in Volts. We are interested in two columns: time-stamp and voltage. This series is plotted in fig. 5.3.

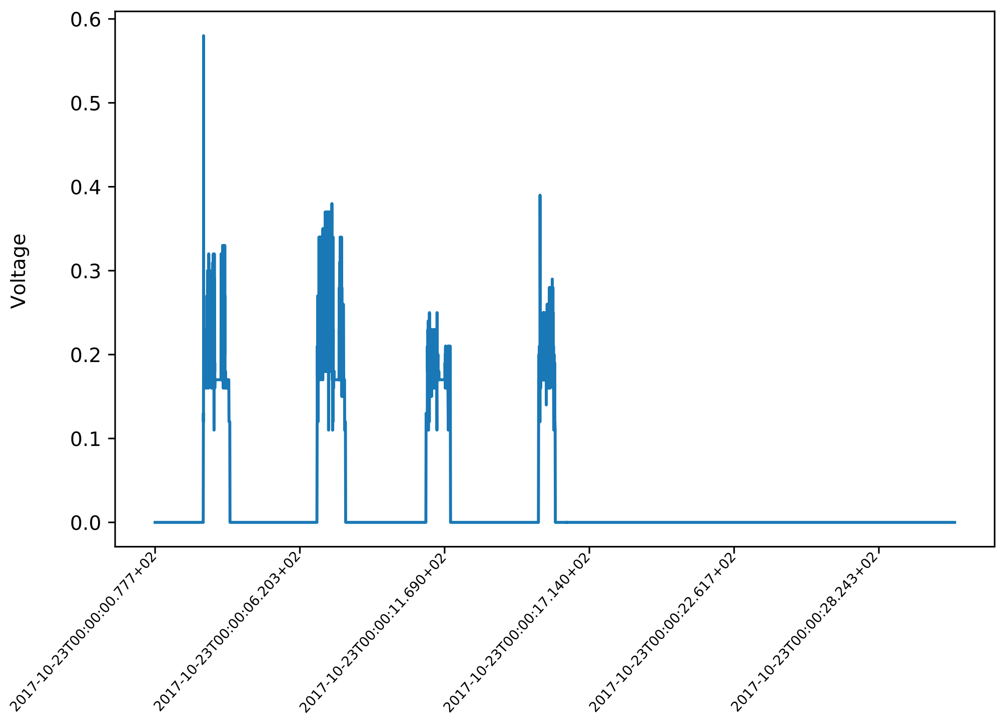


Figure 5.3. One week data from an industrial machine sensor. Source: [4].

Chapter 6

Experiments

In the following sections we will show all the relevant experiments performed with our data. Our methods are implemented in Python and they are executed on a machine with a 2.5GHz core i7-6500U processor and 8 GB RAM running the 64-bit Windows operating system.

6.1 Experiment 1: Classifying the raw data

The first test we decided to perform uses the raw data, without preprocessing them. We use the original labeled CBF time series as it is stored in the UCR archive. We know that there are three classes of motifs and each has a fixed length equal to 128. With this information we can define a non-overlapping window with a width of 128 values, and we can classify each sub-sequence with the classic k-NN from alg. 1. The aim of this experiment is to verify if the classification algorithm works well, comparing our results with what is reported in the data-set information and to make a comparison between the Euclidean distance and the DTW.

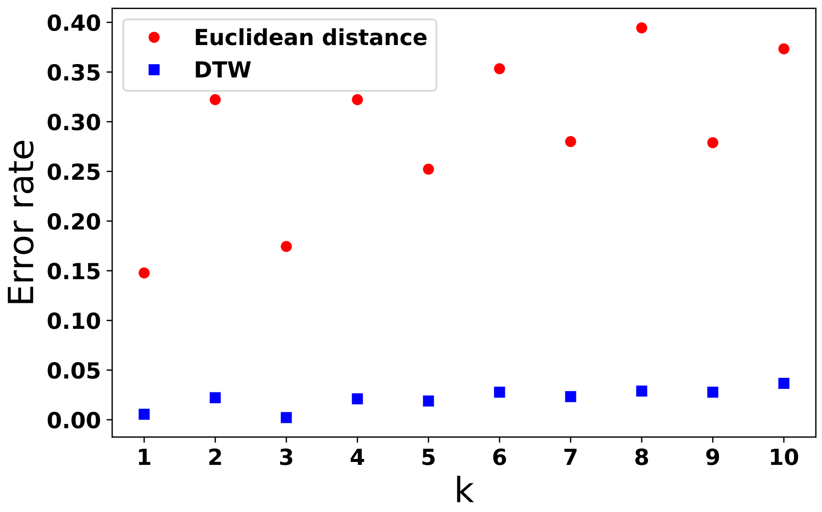


Figure 6.1. *k*-NN applied to CBF time series with DTW and Euclidean distance measure.

We run our algorithm with *k* from 0 to 10 and, as it is easy to see in figure 6.1, we obtain a lower error rate with DTW in all the cases. By the way, we have to underline how much faster the Euclidean distance is, as it is shown in table 6.1. The execution time refer to the testing part, as it is always the same for the train part.

6.2 Experiment 2: Classifying the SAX data

In the next experiments we will introduce the SAX representation. The purpose of the following tests is to show the enhanced performances offered by this new technique. We transform our continuous real-valued time series in a string (with a shorter length due to the PAA dimensionality reduction). After the discretization process we need to apply again the *k*-NN algorithm to classify our patterns. We fix the *k* parameter equal to 1 and we use the same window size of experiment 6.1 (with no overlapping). We can still apply the DTW distance between two strings, but we cannot use the Euclidean distance anymore. Instead, we will adopt the MINDIST explained in section 4.1.4.

Fixed the *k* parameter we have still to set two variables, the fragment length from PAA and the alphabet size from SAX. We could either choice them randomly, or we can use the same values proposed in [19], or we can try all the possible combinations and select those with the lowest error rate, as shown in fig. 6.2. The graph shows how the error changes when the others two parameters

k	**Distance Measure**	**Execution Time (s)**	**Error Rate**
1	Euclidean	0.311	0.148
1	DTW	1860.698	0.006
2	Euclidean	0.572	0.322
2	DTW	1670.720	0.022
3	Euclidean	0.308	0.175
3	DTW	1502.893	0.002
4	Euclidean	0.342	0.322
4	DTW	1489.233	0.021
5	Euclidean	0.368	0.252
5	DTW	1668.486	0.019
6	Euclidean	0.374	0.353
6	DTW	1660.821	0.028
7	Euclidean	0.452	0.280
7	DTW	1499.087	0.023
8	Euclidean	0.378	0.394
8	DTW	1530.152	0.029
9	Euclidean	0.367	0.279
9	DTW	1560.501	0.027
10	Euclidean	0.3481	0.373
10	DTW	1556.139	0.037

Table 6.1. Computational time of experiment 6.1.

vary, using the DTW distance. In the plot two dimensions are on the two axes, while the third one (the alphabet size) is represented by the size of the circles. It is easy to understand that shorter fragments lead to a higher error, while a low alphabet size is preferable, considering that smaller circles are at the bottom of the graph.

The next step consists of performing the same experiment but with the MINDIST distance. We can notice in fig. 6.3 how it is better to adopt different parameters than previously. This time the alphabet size affects the error rate more than before: a smaller size leads to a bigger error. The effect of the fragment length is unchanged, longer fragments are better.

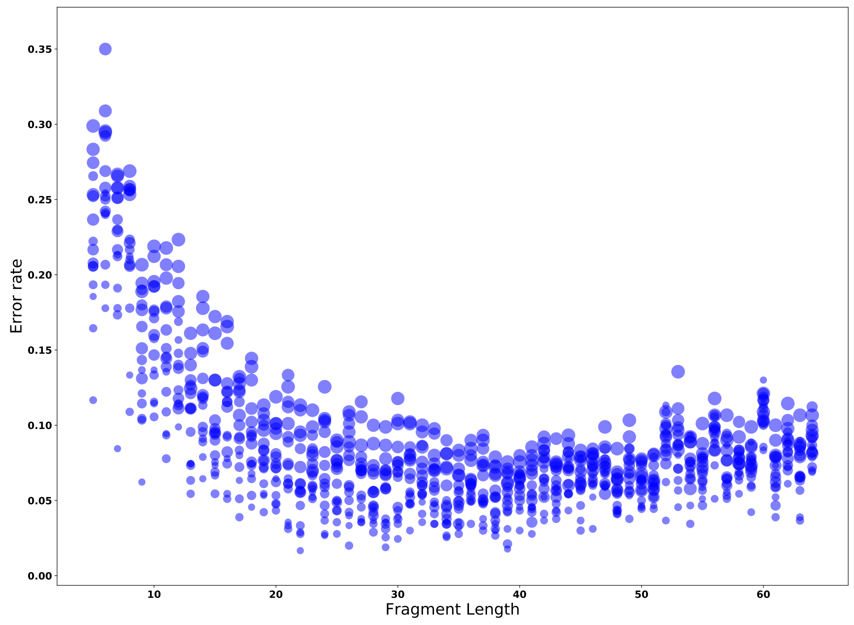


Figure 6.2. Fragment length, Error rate and Alphabet size using DTW. The alphabet size is proportional to the size of the dots.

We summarize the best results obtained with the previous techniques in table 6.2 and we highlight the execution time in fig. 6.4, so we can easily visualize the relationship between error rate and computational time for each method.

Data Representation	**Distance Measure**	**Fragment Length**	**Alphabet Size**	***k***	**Train Time (s)**	**Test Time (s)**	**Error Rate**
Raw Data	Euclidean	-	-	1	0.016	0.311	0.148
Raw Data	DTW	-	-	3	0.016	1502.893	0.002
SAX Data	DTW	22	5	1	0.016	0.010	0.016
SAX Data	MINDIST	9	8	1	0.016	0.001	0.044

Table 6.2. Best results from exp. 6.1 and exp. 6.2 with their respective parameters.

The code for the training part is always the same, and so the time is. That is why we are more interested in the testing part.

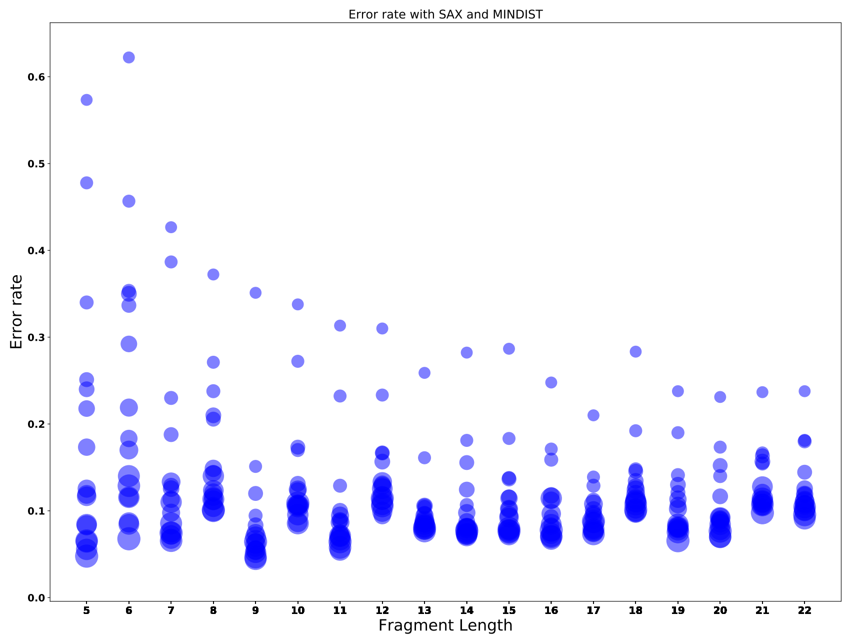


Figure 6.3. Fragment length, Error rate and Alphabet size using MINDIST. The alphabet size is proportional to the size of the dots.

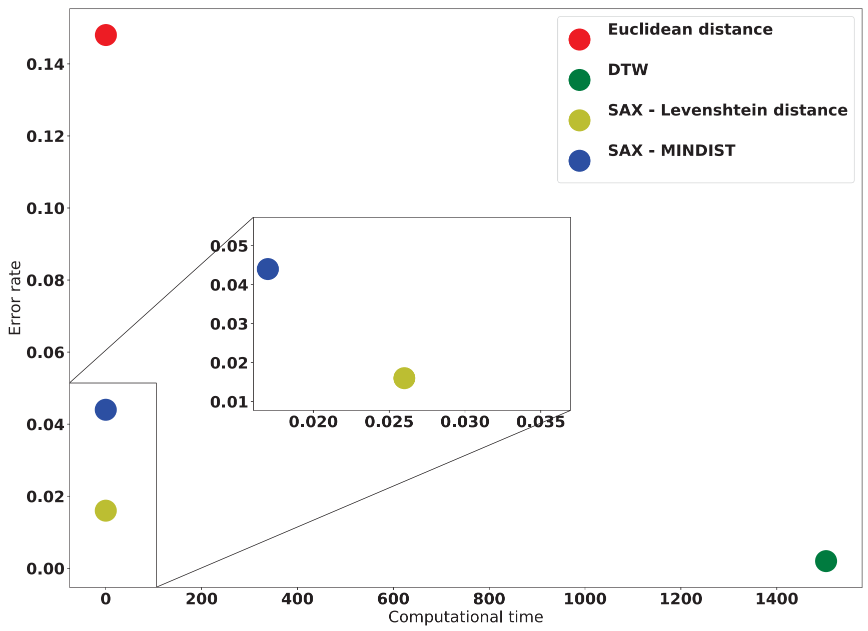


Figure 6.4. Best results from exp. 6.1 and exp. 6.2 with computational time.

6.3 Experiment 3: Variable length patterns

As mentioned in the section 5, we analyze another data-set to test our variable length patterns discovering algorithm. The choice is a segment from ECG data (see fig. 5.2) with a diagram made up of two parts: a calibration signal at the beginning, and then a true ECG signal, as shown in fig. 6.5. In the same figure the patterns retrieved by [27] are also highlighted, so we can compare them with ours own motifs. Unfortunately the authors of [27] do not report all the parameters they used,

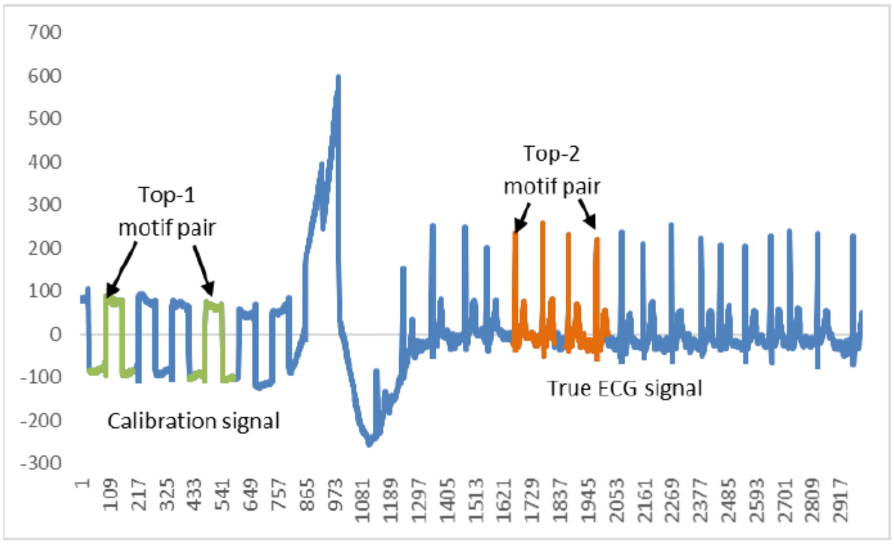


Figure 6.5. Variable length motifs retrieved with the algorithm described in section 4.4. Source: [27].

so, depending on the variables we set to test our algorithm, we may obtain different patterns.

In the next series of experiments we adopt the method of section 4.4, so the SAX representation of our data is now enhanced with the numerosity reduction feature. As we said in the previous chapter, this method comprehends several more parameters, and defining them is not a naive process. To test our code, we take the same parameters chosen by the authors of the original papers but, due to there are few missing variables, we need to define some of them empirically. We run our code dozens of time with different settings, but we decided to report only the most meaningful and relevant results. In fig. 6.6 and fig. 6.7 we show the motifs discovered using a 1-Word Inverted Index and a 2-Word Inverted Index respectively.

We have to remember that for this data-set the only way to say if our patterns are right is to compare them with what has been found in fig. 6.5.

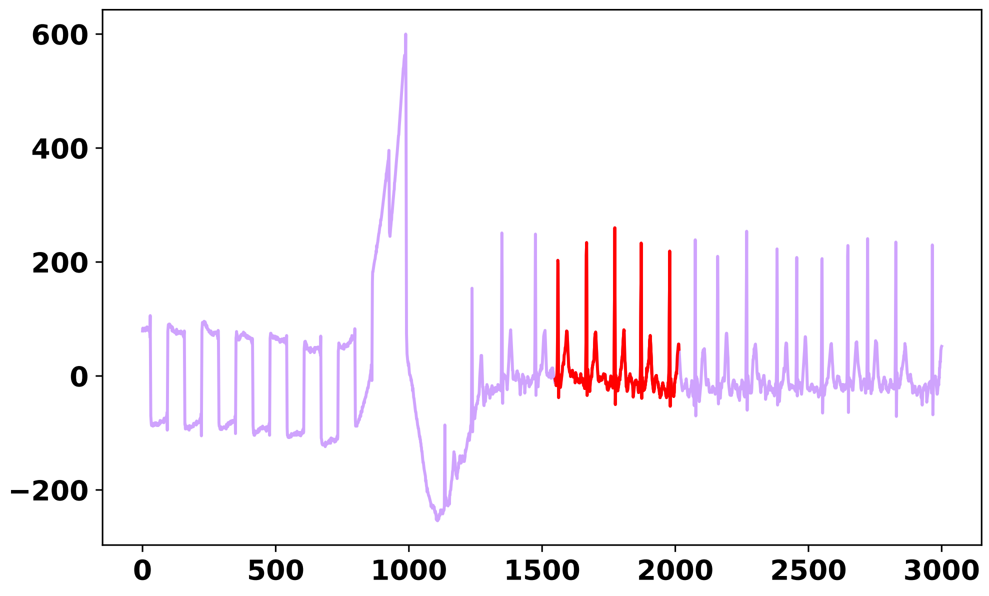


Figure 6.6. Variable length motifs retrieved with 1-Word II.

The most astonishing characteristic is the computational time necessary to retrieve our variable length motifs. The process, excluding the I/O operations, but including the conversion of our database in the SAX representation and the construction of a 1-Word, 2-Word and 3-Word Inverted Index, lasts 0.849 seconds. Retrieve the patterns is really fast, what slow down the whole process is the motif verification part, because of the DTW distance computation.

For completeness we report the adopted parameters in table 6.3. The ‘Frequency Threshold‘ has been chosen empirically.

Fragment Length	8
Alphabet Size	3
Frequency Threshold (equal for all the indexes)	2
Window Size	150

Table 6.3. Parameters used in the experiment 6.3

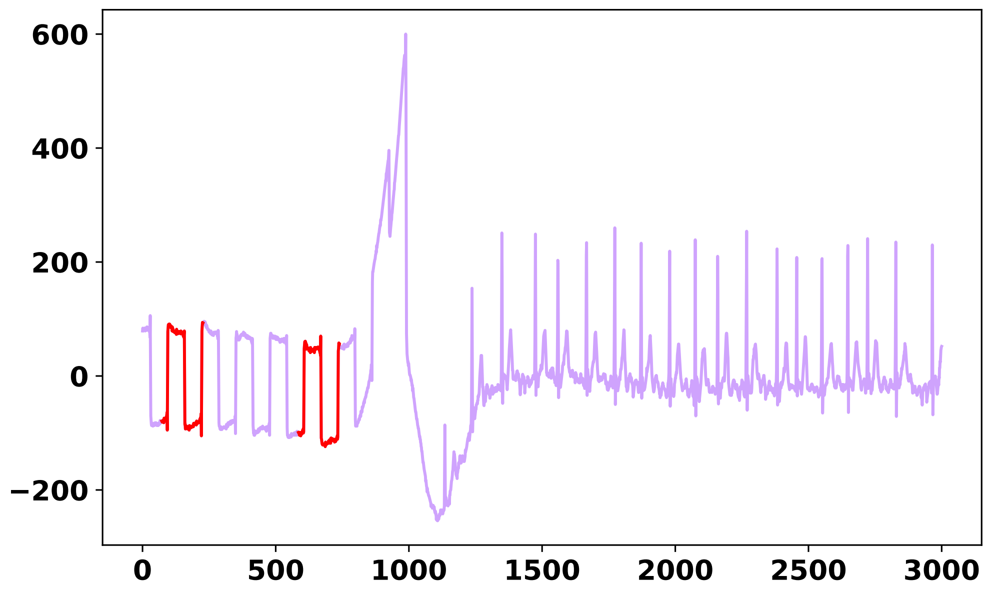


Figure 6.7. Variable length motifs retrieved with 2-Word II.

6.4 Experiment 4: Patterns in a Real Data-set

From now on, we will use our real industrial data. Performing tests with this database is the core of our work, but we have also to mention that the whole experimental procedure will be slowed down, due to the high quantity of information to process. We spend ca. 8 seconds only for reading the file. Then we need to add the time spent for visualize our data, with graphs and statistical information. They are fundamental to understand our result, but that require a significant amount of resources to be computed and printed.

Our first tests wil be performed using the same parameters reported in table 6.3 taken as default. We run it with a 2-Word and a 3-Word Inverted Index and in the figures 6.8 and 6.9 we highlight the resulted patterns.

As we already said, in the original data-set (fig. 5.3) there are a lot of zeros. To avoid these null values to be identified as eventual patterns, we allow only motifs with a maximum length equal to twice the size of the window. Then, instead of comparing every couple of candidate patterns using the DTW distance, that requires a large amount of time due to its quadratic complexity, to verify the top-k motifs, we keep them all. Doing that the total running time (excluding I/O) is 25.365 seconds. There are 18 retrieved motifs with a 2-W II and 32 with a 3-W II.

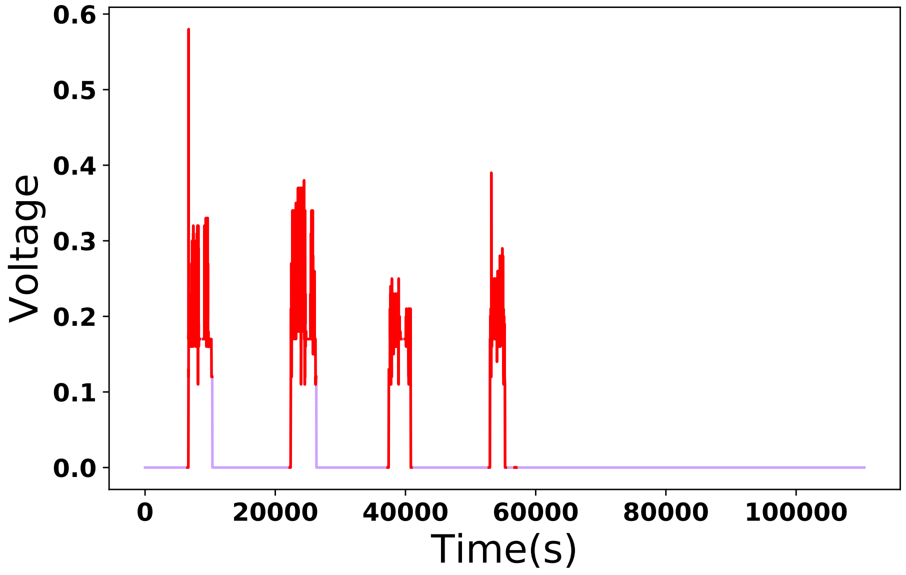


Figure 6.8. Motifs retrieved with 2-Word II from real data-set.

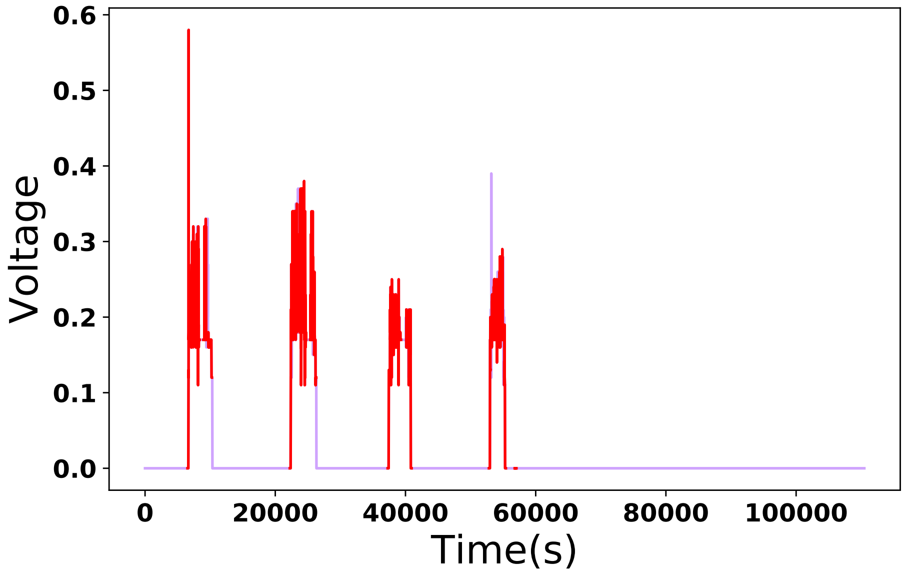


Figure 6.9. Motifs retrieved with 3-Word II from real data-set.

The default window width was valid for the previous data-set. Unfortunately there is not a predefined value valid for all the data, so in the next tests we want to show the difference of using different window sizes, choosing 4 different orders of magnitude for our parameter (see fig. 6.10 and fig. 6.11).

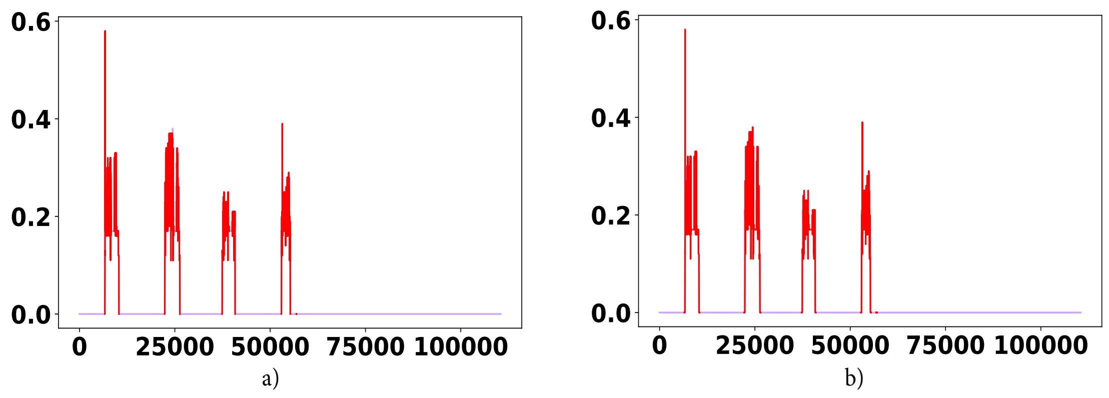


Figure 6.10. Effect of different window sizes on Chipsaw data-set. a)Window=20, b)Window=200.

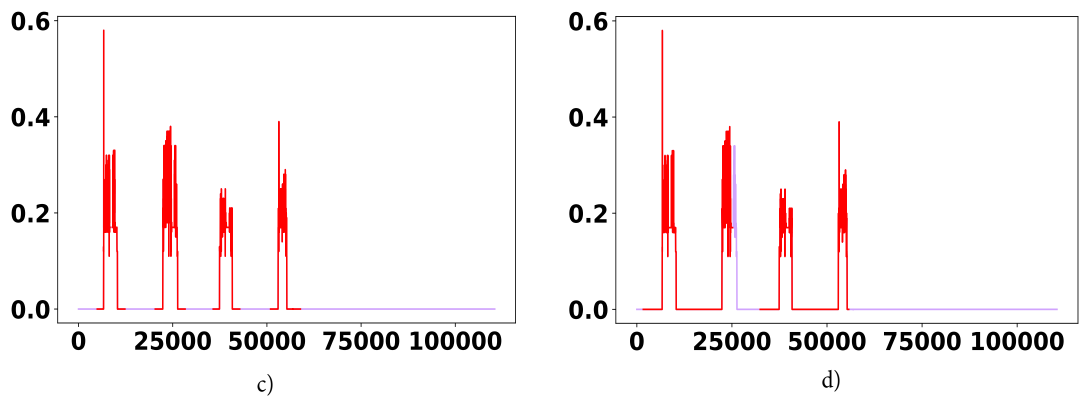


Figure 6.11. Effect of different window sizes on Chipsaw data-set. c)Window=2000, d)Window=20000.

It is obvious that the window size is related to the patterns' length: a wider window implies longer motifs, and they may include a lot of zeros too.

There is also an implication on the computational time reported in table 6.4.

Window Size	**Computational Time (s)**	**Figure**
20	31.699	Fig. 6.10 a)
200	57.715	Fig. 6.10 b)
2000	90.598	Fig. 6.11 c)
20000	97.094	Fig. 6.11 d)

Table 6.4. Computational time for different window sizes. 6.1.

To understand the zeros' problem better, we perform the same tests done previously, we edit our input data removing the values equal to zero. We keep only one zero between non-null values, so our algorithm can still understand when the sensor goes from off to on and vice-versa. The input data from an initial length of 110470 are reduced to a length of 13301 (fig. 6.12).

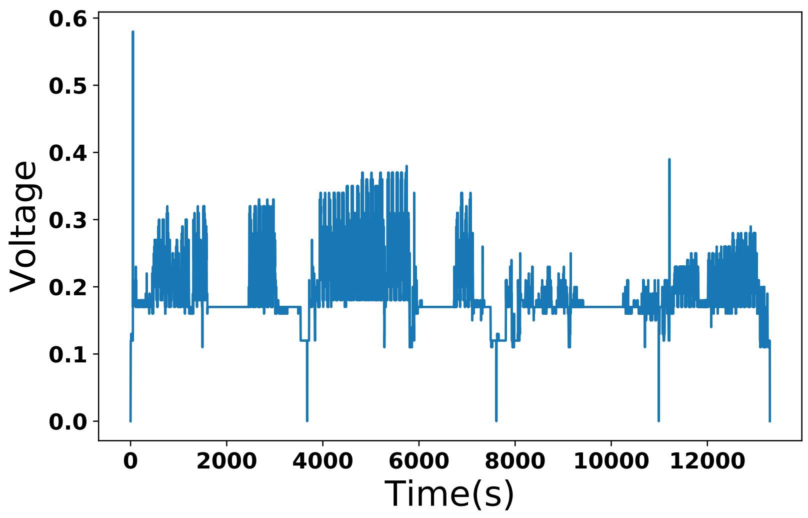


Figure 6.12. Real data-set without zero values.

In fig. 6.13 is possible to see how the candidates patterns look like in the reduced data-set, and in fig. 6.14 we show the same patterns highlighted in the original time series.

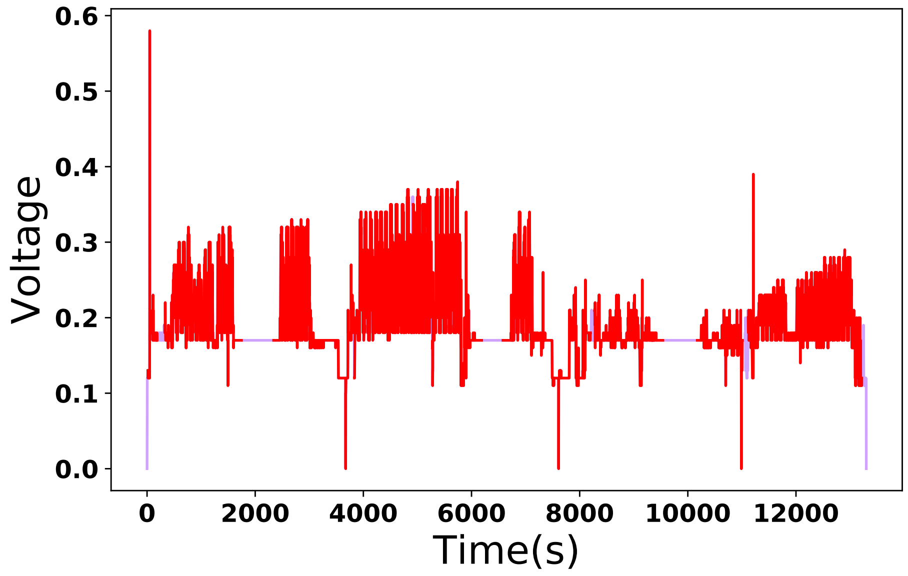


Figure 6.13. Motifs retrieved with 2-Word II, using the parameters in table 6.3, from real data-set without zeros.

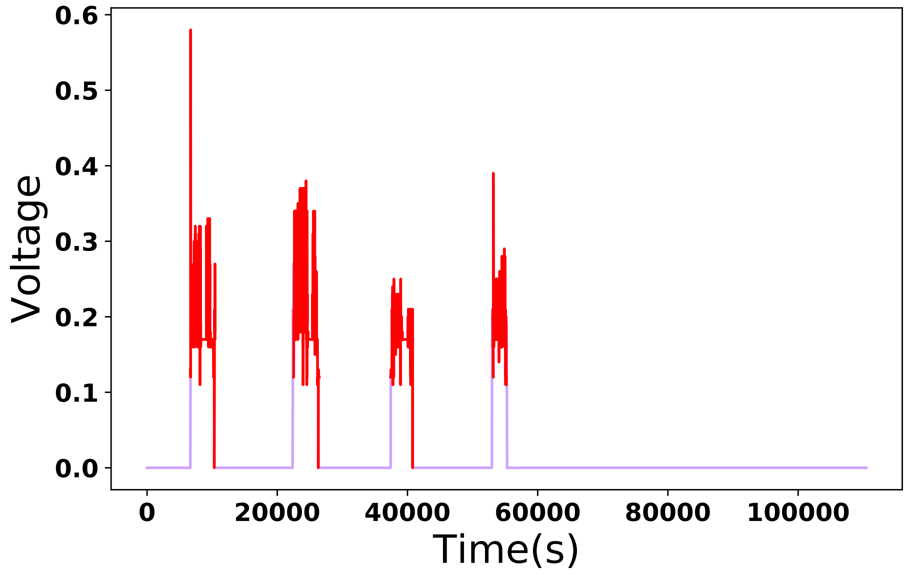


Figure 6.14. Motifs retrieved with 2-Word II, using the parameters in table 6.3, from real data-set without zeros.

6.5 Experiment 5: Incorporate Expert Knowledge

To incorporate the knowledge of an expert we need to ask him the parameters to set in our algorithm. To better understand the problem we might also ask some details about the data we are treating and, because he is probably not a data scientist, we have to prepare our questions to be focused on the data and not on the algorithm. After an accurate selection, we report in the following list the main questions we asked the the expert, with the expected type of value:

- What kind of data do we have? Could you please describe it?

  [Open question]

- Where does the data come from?

  [Open question]

- What does the sensor measure?

  [Open question]

- What kind of machines does the sensor detect?

  [Open question]

- How long does a regular process on this machine take? Is it an exact number or an approximation?

  [Numeric value]

- Do you know the maximum (or minimum) length of a possible pattern? Are you sure of it, or it is just your impression?

  [Numeric value]

- What is the time interval at which the measurements are taken?

  [Numeric value]

- Can several processes measured by a single sensor run simultaneously?

  [Open question]

- Do you have any information regarding the number of possible patterns? How many different forms can the process run? How many processes are measured by the sensor?

  [Range of numeric values]

- Are the zero values important? What do they mean? Do we have zeros when the sensor is off? Or when it is on and it does not collect any data?

  [Open question]

- And the none values? Why do we have timestamp cells that are not associated to any value? Can we just ignore them?

  [Open question]

- Is it possible to get negative values? Why?

  [Open question]

- Is there a lot of variance in your process/patterns?

  [Range of numeric values]

- We use the so-called Symbolic Aggregate approXimation (SAX) to represent our data. To do that we discretize the time series and we applied a dimensionality reduction technique. Do you have any idea of the possible number of symbols we can use to discretize our continuous real-valued series? Would you prefer to define it after we plot the data? Are you sure about that? Consider that according to our experiments a bigger number of symbols leads to a higher accuracy.

  [Range of numeric values]

- To decrease the dimensionality of our data, we divide the series in fragments using a non-overlapping window. Then we reduce the length of each fragment to a chosen value. After seeing the tests we performed, are you able to define this parameter? Alternatively, a set of possible parameters?

  [Range of numeric values]

Based on the answers of the expert we run the algorithm with the parameters reported in table 6.5, with an 2-Word II, once considering the zeros and once without considering them.

Fragment Length	7
Alphabet Size	6
Frequency Threshold (equal for all the indexes)	2
Window Size	330 (30 minutes)

Table 6.5. Parameters given by the expert

The obtained patterns are shown in fig. 6.15, and in fig. 6.16 is possible to see them side by side.

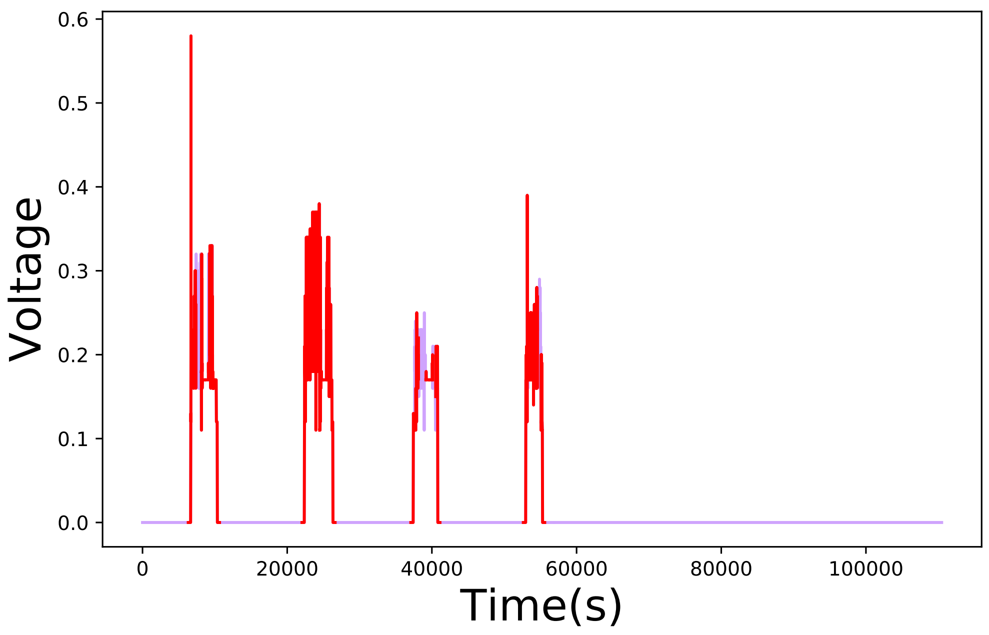


Figure 6.15. Motifs retrieved with 2-Word II, using the parameters in table 6.5, highlighted in the original time series.

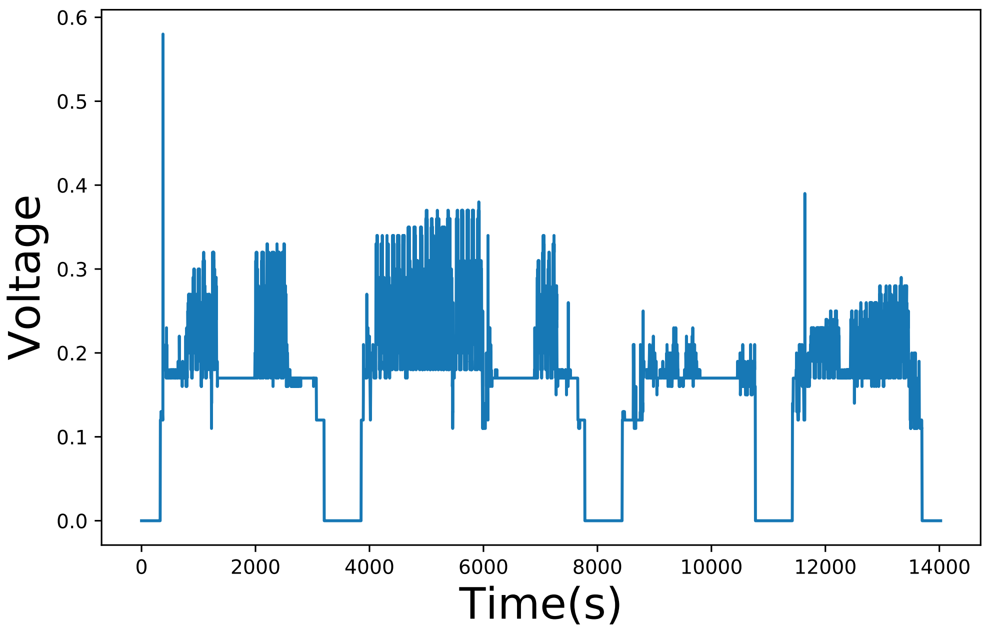


Figure 6.16. Motifs retrieved with 2-Word II, using the parameters in table 6.5, plotted side by side.

Chapter 7

Discussion

Many algorithms for patterns retrieval have been proposed in the last years. Our method, which combines several other techniques, is one of them. In the previous chapter, we demonstrate its potential and its ability to deal with a huge amount of data.

We can now better analyze the results obtained in the previous chapter. We have seen that using the raw data, without preprocessing it we obtain the same error rate reported by the creator of the data-set. Unlike with the Euclidean distance, when we use the DTW we may notice a very small difference between our value and the given one. That is caused by the use of the fastDTW function already implemented in Python libraries, instead of the original DTW distance measure.

Testing again labeled data, but this time after a preprocessing phase and with well defined windows (suggested by the author of data-set) for our SAX representation, we are able to obtain a great accuracy in less time. With SAX we adopt the MINDIST and the DTW (we need to underline that when we refer to DTW applied to strings in reality we talk about the Levensthein distance, often known also as Edit distance), and in both cases we are faster than the techniques adopted with raw data, and we reach a better accuracy than he Euclidean distance. This is an impressive result, because the only way to reach the same order of accuracy using untreated data is to exploit the DTW distance measure, that has complexity equal to $\mathcal{O}(n^2)$, and that applied to the whole series (not reduced) requires a lot of time.

Going through the analysis of unlabeled data, we show that we have a good capacity to retrieve motifs even if they have different lengths. That allows us to extend the number of possible future applications of our method. In the energy industry many processes do not have a prefixed duration.

In addition, even if we are dealing with this specific data type, we may extend our view and think about applying our techniques to others domains. Furthermore, verifying the candidates motifs, pruning the indexes instead of computing the distance between every possible couple of patterns, allows to speed up the whole retrieval process.

After all the features have been implemented and tested we need to comment the results on the real data-set.

Test our real data-set, without any a priori information, is more difficult. The integration of experts' knowledge is fundamental, for both evaluating and choosing the parameters. From the previous experiments we saw that the best combination of settings is the SAX with indexes, because in this way we have a fast algorithm able to find variable length patterns. We already discussed the main drawbacks of the integration process that is probably still slow. The acquisition bottleneck is not easy to overcome, especially if the expert of the sector has no a computer engineering background. By the way, the feedback integration gives us a better analysis with respect to others internal evaluation cluster techniques. With our implementation we try to find the best compromise between speed and accuracy. The speed gained in the retrieval process makes our approach competitive, because of the good accuracy obtained.

We try to run our algorithm with data-sets that have different characteristics and, at the end, we test it with a real data-set, but we have never evaluated the performances with a similar database that contains instead of thousands of points, trillions of points.

The algorithm's performances depend also on many other variable, ad example the characteristic of the machine where it runs.

As shown in the previous chapter, we want also to run the algorithm before the expert gives us some feedback. To do that we need to set some parameter, and this process is not naive, because they influence a lot the final result, and because to get the choice we need to run it multiple time, and it requires a lot of time. Run the algorithm just for testing the chosen variables has a very high cost.

Choosing the learning algorithm as base for our approach is difficult too, because the literature is plenty of them and often one that performs good on some data, may be not as good as with others data.

We need to say that when we state our measures referred to computational time, we always report an average of multiple running with the same data and settings.

Then we have to comment our choice to implement our algorithm in Python. It is driven from the fact that the Python programming language has a lot of libraries and build-in function related to machine learning algorithms development. The scientific community also like the fact that it is an high-level language, so it is easier to write and read code, especially for people that are not in computer engineering, like mathematicians or data analysts. In addition it is an interpreted language and his is a good point because it can be easily run on different operating systems without compiling it again. At the same time we have to mention some disadvantages, especially from the performance point of view, due to it runs within a python interpreter. Coding in some lower-level languages such C or C++ might give us the possibility to improve the performances, but implementing every single algorithm and data structure from scratch would have required much more time for developing. The absence of machine learning libraries available in C or C++ is maybe one of the lack from the data mining community.

Even if the interest in big data and data mining increased a lot in the last years, the availability of open source code and test databases is really scarce. Very popular search engines give now the possibility to search for data-sets to download, but there are still many researchers that keep their source code closed. Often even if they publish and explain their methods, they do not explicitly declare all the parameters they used for the experiments. This makes their experiments unreproducible or it takes a lot of time to do them again.

Implementing the same algorithm becomes much more difficult if we cannot even perform the same tests to verify the correctness of our own code. Code review is very important too, complete bugless is impossible to reach, but having more people available to read it and to test it with several different databases would be useful too.

The strength of our method is given by its speed. Conventional analytic and prediction models take weeks or months to analyze huge data-sets, and they are unfit for today's needs. Dynamic analytic processes like machine learning algorithms in less time furnish accurate answers. This gives the possibility to study the behavior of machines, to project data collected in months or years, and to give an accurate analysis, or even to project forecasts.

Part V

Conclusion

Chapter 8

Further Research Steps

The art of discovering motifs has a successful history that goes back to the 16th century when scientists and philosophers tried to observe regularities in the nature. Today, after centuries of experience and even though we have machines that help us, finding patterns is a difficult task to accomplish.

Nowadays big data are not just useful, but they are essential for companies and research centers. Processing them is a challenge, especially if we have no prior information about them. Our main goal was to make available a tool able to treat this huge amount of data collected everyday. In our work we put together several machine learning techniques in order to obtain a motifs discovery algorithm fast and accurate.

As already discussed in the previous chapter, we succeed in extending the potentiality of traditional methods integrating some expert feedback that allows to reduce the error and to validate the result when we deal with unsupervised learning.

We have seen that despite the scientific community keeps developing powerful solutions for machine learning algorithms, in the literature there is still a lack of attention towards man-machine collaboration systems, even if there are numerous practical applications. We have shown the improvements obtained with the help of an expert and we pointed out how traditional algorithms could perform better with knowledge elicitation techniques.

Our method is just a starting point for further research in this area. We showed that our technique works, but we felt the absence of a scientific estimation of the errors that might be caused by the experts, that is why we evaluated it empirically.

Furthermore our algorithm can be still optimized from the code and the data structures point of view. We explained that we still have the obstacle of the knowledge acquisition bottleneck, and our software is not helpful from this point of view because it is not so user friendly. A graphic user interface could allow the expert to perform his own experiments and evaluations, with an interactive section that he could use to visualize the data and set the parameters.

We hope our work arouses the interests of data analysts, and we would like to see it performing in real world applications, either for classification tasks or for forecast analysis, and maybe with a data-set with a size of several GBs.

Bibliography

[1] M. Ardelt, “Wisdom as expert knowledge system: A critical review of a contemporary operationalization of an ancient concept,” *Human Development*, vol. 47, no. 5, pp. 257–285, 2004.

[2] 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 and Knowledge Discovery*, vol. 31, no. 3, pp. 606–660, 2017.

[3] 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 and Knowledge Discovery*, vol. Online First, 2016.

[4] S. Bischof, H. Trittenbach, M. Vollmer, D. Werle, T. Blank, and K. Böhm, “Hipe – an energy-status-data set from industrial production,” in *Proceedings of ACM e-Energy (e-Energy 2018)*, (New York, NY, USA), pp. 599–603, ACM, 2018.

[5] B. Chandrasekaran, “Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design,” vol. 1, pp. 23–30, 1986.

[6] B. Chiu, E. Keogh, and S. Lonardi, “Probabilistic discovery of time series motifs,” in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’03, (New York, NY, USA), pp. 493–498, ACM, 2003.

[7] H. L. Dreyfus, S. E. Dreyfus, and T. Athanasiou, *Mind over Machine: The Power of Human Intuition and Expertise in the Era of the Computer*. New York, NY, USA: The Free Press, 1986.

[8] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, “Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals,” *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[9] T.-c. Fu, “A review on time series data mining,” *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

[10] J. Han, G. Dong, and Y. Yin, “Efficient mining of partial periodic patterns in time series database,” in *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, pp. 106–115, 1999.

[11] John R Wilson and Sarah Sharples, eds., *Evaluation of Human Work*. CRC Press, 2015.

[12] E. J. Keogh and M. J. Pazzani, “An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback,” in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD’98, pp. 239–243, AAAI Press, 1998.

[13] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Locally adaptive dimensionality reduction for indexing large time series databases," *ACM Sigmod Record*, vol. 30, no. 2, pp. 151–162, 2001.

[14] E. Keogh, S. Lonardi, and B. Y.-c. Chiu, "Finding surprising patterns in a time series database in linear time and space," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, (New York, NY, USA), pp. 550–556, ACM, 2002.

[15] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: a survey and empirical demonstration," *Data Mining and knowledge discovery*, vol. 7, no. 4, pp. 349–371, 2003.

[16] K. Kohara, T. Ishikawa, Y. Fukuhara, and Y. Nakamura, "Stock price prediction using prior knowledge and neural networks," *Intelligent Systems in Accounting, Finance and Management*, vol. 6, no. 1, pp. 11–22, 1997.

[17] R. J. Larsen, M. L. Marx, *et al.*, *An introduction to mathematical statistics and its applications*, vol. 2. Prentice-Hall Englewood Cliffs, NJ, 1986.

[18] J. Lin, E. Keogh, S. Lonardi, and P. Patel, "Finding motifs in time series," pp. 53–68, 2002.

[19] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, DMKD '03, (New York, NY, USA), pp. 2–11, ACM, 2003.

[20] N. Ludwig, S. Waczowicz, R. Mikut, and V. Hagenmeyer, *Mining Flexibility Patterns in Energy Time Series from Industrial Processes*. 2017.

[21] S. Petrutiu, A. V. Sahakian, and S. Swiryn, "Abrupt changes in fibrillatory wave characteristics at the termination of paroxysmal atrial fibrillation in humans," *EP Europace*, vol. 9, no. 7, pp. 466–470, 2007.

[22] P. A. Pevzner, S.-H. Sze, *et al.*, "Combinatorial approaches to finding subtle signals in dna sequences," in *ISMB*, vol. 8, pp. 269–278, 2000.

[23] N. Saito, "Local feature extraction and its applications using a library of bases," in *Topics in Analysis and Its Applications: Selected Theses*, pp. 269–451, World Scientific, 2000.

[24] B. Schott, J. Stegmaier, M. Takamiya, and R. Mikut, "Challenges of integrating a priori information efficiently in the discovery of spatio-temporal objects in large databases," *CoRR*, vol. abs/1602.02938, 2016.

[25] N. Shadbolt and P. R. Smart, "Knowledge elicitation: Methods, tools and techniques," in *Evaluation of Human Work* (John R Wilson and Sarah Sharples, eds.), pp. 163–200, CRC Press, 2015.

[26] C.-C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, Z. Zimmerman, D. F. Silva, A. Mueen, and E. Keogh, "Time series joins, motifs, discords and shapelets: A unifying view that exploits the matrix profile," *Data Mining and Knowledge Discovery*, vol. 32, no. 1, pp. 83–123, 2018.

[27] C. T. Zan and H. Yamana, "A variable-length motifs discovery method in time series using hybrid approach," in *Proceedings of the 19th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '17, (New York, NY, USA), pp. 49–57, ACM, 2017.