

---

# Goal-directed graph generation for anomaly detection on time-series

**An application to arrhythmia detection on ECG signal**

Master's Thesis submitted to the  
Faculty of Informatics of the *Università della Svizzera Italiana*  
Master of Science in Artificial Intelligence

presented by

**Amirehsan Davoodi**

under the supervision of

Prof. Cesare Alippi

co-supervised by

Daniele Zambon

September 2020



---

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

---

Amirehsan Davoodi  
Lugano, 5th September 2020



All models are wrong, but some are  
useful.

George Box



# Abstract

Detecting outliers in skewed data or identifying anomalies that have not been observed before in a given set is a challenging task with simple statistical methods and requires specialized algorithms. An automatic anomaly detection method that can be used as an oracle to label anomalous data with high accuracy is very useful across different industries. In the context of the time-series, anomaly detection can lead to forecasting new patterns in the recorded data or detecting a change in the baseline. Here in this work, we are proposing a novel methodology to detect anomalies for univariate time-series data. The intuition behind my methodology is to take advantage of the temporal dependence in the time-series data.

Our model for capturing the temporal dependencies across the signal is based on generating a graph from the periodic patterns in the data. Then we analyzing the change in the topology of the graph instead of considering single data-points. The advantage of this method is to provide robustness in dilated or contracted time-series data. Furthermore, this method can be useful in cases with non-stationary input data. This graph-based approach can be the key to distinguish anomalies which are very close to normal data by using the local dependency of the data stream.

The developed algorithm works in three main steps: (1) create a vector stream, (2) generate a natural visibility graph, and (3) apply a local outlier factor (LOF) method to detect anomalous graphs. The LOF method requires using a distance metric that can be chosen based on the graph size and computational resources. As a case study, we applied this algorithm to electrocardiograph signals to detect arrhythmia in 48 patients to validate the effectiveness of our proposed methodology. We tried graph-edit distance, Hausdorff-edit distance, and NetSimile as the most feasible graph similarity metrics. The latter metric provided the best combination of computational and accuracy performance. Benchmarking our method against the vector-based anomaly detection by Oehmcke et al. [1] shows similar average performance for 21 patients in our dataset.





# Acknowledgements

This dissertation was only possible thanks to the help and support of many people. I apologize that I am not able to list all the people to whom I feel indebted.

Firstly, I would like to express my sincere gratitude to my advisor Prof. Alippi for the continuous support and useful advice. His guidance and encouragement helped me immensely during the research and writing of this thesis.

Besides my advisor, I would like to thank Daniele Zambon for stimulating discussions, useful advice, and multiple revisions of this thesis.

A special thanks to my best friend and brother Amirhossein for his supports, love, and affection.

I would like to thank my parents for their never-ending support through the years in every aspect of my life and providing the best role models for me. Last but not least, I would like to thank my wife, Mina, who shared the burden of the last year of my graduate studies.



# Contents

<b>Contents</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.1.1 Graph-based approach . . . . .	1
1.1.2 Anomaly detection . . . . .	2
1.2 Our proposed methodology . . . . .	3
1.3 Evaluation on MIT-BIH Arrhythmia Database . . . . .	4
1.4 Contributions . . . . .	5
1.5 Thesis structure . . . . .	5
<b>2 Graph-based anomaly detection</b>	<b>7</b>
2.1 Anomaly detection . . . . .	7
2.1.1 Outlier vs. Novelty . . . . .	8
2.1.2 Supervised vs. Unsupervised learning . . . . .	8
2.1.3 Outlier scores vs. Binary labels . . . . .	8
2.1.4 Outlier score threshold . . . . .	9
2.1.5 Anomaly detection vs Classification . . . . .	10
2.1.6 Quick review of the existing approaches . . . . .	12
2.2 Time-series data . . . . .	13
2.2.1 Time-series properties . . . . .	13
2.2.2 Stationary vs. Non-stationary Time-series . . . . .	14
2.2.3 Time-variant vs Time-invariant . . . . .	15
2.2.4 Types of Anomalies . . . . .	16
2.2.5 Time-series models . . . . .	18
2.3 Graph-based approaches . . . . .	21
2.3.1 Why graph representation? . . . . .	21

2.3.2	Static vs. Dynamic Graphs . . . . .	22
2.3.3	Graph-based models . . . . .	23
2.4	Model selection . . . . .	25
<b>3</b>	<b>Proposed methodology (GrAnD UniTs)</b>	<b>27</b>
3.1	Vector stream from univariate time-series . . . . .	28
3.1.1	Find periodic events . . . . .	29
3.1.2	Find periodic patterns . . . . .	29
3.1.3	Create vector for each periodic pattern . . . . .	30
3.2	Graph stream from vector stream . . . . .	30
3.2.1	Visibility Graph . . . . .	30
3.3	Anomaly detection on graph stream . . . . .	32
3.3.1	Local Outlier Factor (LOF) . . . . .	33
3.3.2	Graph Edit Distance (GED) . . . . .	36
3.3.3	NetSimile (a size-independent network similarity) . . . . .	37
3.3.4	Quantitative Evaluation . . . . .	37
3.4	Challenges . . . . .	38
3.4.1	Data-specific challenges . . . . .	38
3.4.2	Problem-specific challenges . . . . .	39
3.4.3	Graph-specific challenges . . . . .	40
<b>4</b>	<b>Experiments on ECG</b>	<b>41</b>
4.1	Electrocardiogram . . . . .	41
4.1.1	MIT-BIH Arrhythmia Database . . . . .	43
4.2	Vector stream from time-series . . . . .	47
4.2.1	Find periodic events (R-peaks) . . . . .	47
4.2.2	Find periodic patterns (Heartbeats) . . . . .	51
4.2.3	Anomaly labels from annotations: . . . . .	54
4.2.4	Performance of different R-peak detectors: . . . . .	55
4.3	Graph stream from vector stream . . . . .	56
4.4	Anomaly detection on graph stream . . . . .	57
4.4.1	Tools . . . . .	58
4.4.2	Computational challenges . . . . .	59
4.4.3	Thresholding and Hyperparameter tuning . . . . .	61
4.4.4	Performance evaluation: . . . . .	63
4.5	Graph-based anomaly detection vs. Vector-based anomaly detection . . .	64
4.5.1	Comparison on a single patient . . . . .	66

4.5.2 Comparison on the whole data set . . . . .	70
<b>5 Conclusion</b>	<b>75</b>
<b>Bibliography</b>	<b>77</b>



# Chapter 1

## Introduction

In this Masters thesis, we examine the potential advantage of representing a time-series as a sequence of graphs, with the end goal being to be able to develop a methodology for anomaly detection on univariate time-series data. In particular, we generate a visibility graph stream of a one-dimensional time-series such that we can learn a mapping between the topology of the graph and the presence of the abnormal event. As a case study, we considered the problem of arrhythmia detection on an electrocardiogram (ECG) signal.

### 1.1 Background

In this part, we give an overview of the preliminary information for understanding the ideas discussed in the thesis.

#### 1.1.1 Graph-based approach

In our interconnected world, almost anything is connected inside and outside. The relation among different objects can be represented as *dependency networks*. Graphs are powerful mathematical structures to capture the relation between objects.

Many objects can be naturally represented as graphs due to their interconnected properties. For instance, airport transportation networks, genetic networks, and social networks which each present pairwise dependency (edge) between their entities (nodes) [2] and there are applications in which this representation exhibited great performance.

In the case that data samples have temporal order such as time-series data the local dependency network between neighboring data-samples can be represented by a visibility graph [3]. The benefits of using graphs to encode information in time series are their ability to capture the presence of nonlinear correlations in time-series and distinguish noise from chaotic series. [4, 5].

### 1.1.2 Anomaly detection

An anomaly is the data point that is significantly different from other observations. [6] The deviation in the observational data can reflect the change in the data generation process or the unusual behavior of the system. Therefore, recognition of abnormal characteristics of the system using an anomaly detection mechanism can provide us useful insight in various applications such as Credit-card fraud detection, Intrusion detection in a computer system, and diagnosis of an *Arrhythmia in an electrocardiogram (ECG)*.

There are several family of models for identifying anomalous data in which each focus on one *specific compelling aspect* of the data that shows the *significantly interesting deviations*. For example, *probabilistic and statistic* models concentrated on the *probability density function (pdf)* of the observations and *proximity-based* models are designed to recognize the isolated data-points base on the *similarity or distance functions*. [6, 7]

In practice, the peculiarity of the anomaly detection resides in its difficulties to characterize the anomalous data from noise. Based on the application and its specific data type, different models have to be tuned to specify the threshold of deviation for a data-point to be labeled as abnormal. This model threshold tuning decision is not always straightforward and usually performed based on the feedback of the model from previous choices of the criteria.

For choosing the right anomaly detection model, we have to consider the data type and its nature. One specific type of data is when the data arrive in a stream such that the order of data and the pattern of change in their values can play an important role in identifying anomalies. In the context of time-series data, anomalies can be of three different kinds of *Point anomaly*, *Contextual anomaly* and *Collective anomaly*. The last of these types are of special interest and also more difficult to detect due to our assumption of temporal continuity.

It is worth noting that time-series data can be multivariate or univariate which each requires its anomaly detection models. Moreover, identifying abnormalities on the time-series data bring us some extra challenges such as the case when streaming



data is non-stationary over time or in case that the data generation process is time-variant.

Graph-based anomaly detection models pursuing a new approach for having insight into the dependency network of the data arrive in the temporal order. Most of the previous graph-based models are designed for the situation when we are dealing with multivariate time-series and they work by generating a dynamic graph for a multi-dimensional data stream [8, 9, 10]. These models can find the suspicious data-points from the remaining normal ones by processing the generated graph called *Graph Signal Processing* [11]. In other words, they only capture the dependencies between multiple data streams not the local inter-dependency network among data-points of a univariate time-series.

A single-lead ECG signal could be a real-world example of a univariate time-series which contains a local inter-dependency network inside a window for each heartbeat. In an electrocardiogram the heart of the person can be considered as our data generating process. To following the nature of our data, we can use a graph-based model for identifying arrhythmia on an electrocardiogram signal.

## 1.2 Our proposed methodology

Our approach here consists of the following three-steps as shown in figure 1.1.

- **Vector stream from time-series:** we split the original time-series into a vector stream using a sliding window with the end goal of examining whether the sequence of samples inside each vector is anomalous or not. The splitting process is to satisfy that temporal data points in each vector are matched with the cycles or seasonality of our data generation process. Thus, even though our time-series may be non-stationary, our vector variables are independent and identically distributed (i.i.d).
- **Graph stream from vector stream:** To capture the local inter-dependency network between the data samples in each vector, we generate a stream of *Visibility Graph (VG)* [12] corresponding to our vector stream.
- **Anomaly detection on graph stream:** In the final step of our methodology, we use *Local Outlier Factor (LOF)* [13] algorithm to detect the abnormal graphs in our graph stream by only considering the topology of the graph. LOF algorithm as a proximity-based anomaly detection method relies on a well-defined distance function between pairwise objects. At this work, we considered two measures of

similarity that works with graphs data types, Hausdorff Edit Distance (HED) [14] and NetSimile [15].

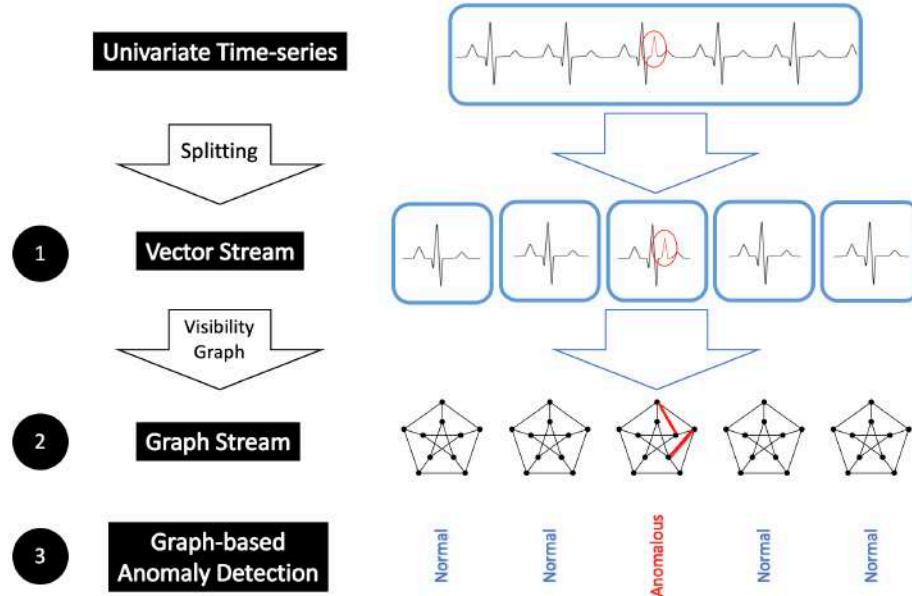


Figure 1.1. Three main steps of our proposed methodology.

### 1.3 Evaluation on MIT-BIH Arrhythmia Database

As a case study, we performed arrhythmia detection on a data-set of one-dimensional ECG signals tracing to validate the effectiveness of our proposed methodology. Experiments on ECG show that what proposed is able to identify heartbeat arrhythmia and the performance of our simple method encourages us to further explore this direction in order to see if it can be even better than the vector alternatives by an ablation study of graph representation of the data or vector representation. This comparison helped us to have a reference point for the ability of our model and the results of this comparison shows a clear improvement on our database.

By our analysis, we conclude that graph representation is able to successfully capture the local inter-dependency on a univariate time-series. Notice, this local dependency network can still help us identify anomalies without having the quantitative values of the series.

## 1.4 Contributions

In summary, our contributions are

- Proposing a new methodology for anomaly detection on time-series that employ graph representations of the time-series
- Performed arrhythmia detection on the electrocardiogram tracing to validate the effectiveness of the proposed method.
- Conduct an ablation study to contrast against the very same method except for the graph representation

## 1.5 Thesis structure

The thesis is structured as follows. In chapter 2, we describe anomaly detection as a machine learning task of identifying anomalous data that deviate from normal ones. Particular focus is devoted to anomaly detection on time-series data and graph-based approaches for doing anomaly detection on a graph database.

Chapter 3 defines the main phases of our proposed methodology "*a Graph-based Anomaly Detection for Univariate Time-series*" for short (*GrAnD UniTs*). In chapter 4, we performed arrhythmia detection on the electrocardiogram tracing to validate the effectiveness of the proposed model. Finally, we summarize our results and mention the future direction in chapter 5.



## Chapter 2

# Graph-based anomaly detection

In this chapter, we explain some background material and notions for understanding the discussions of the chapters to follow. Section 2.1 provides you an overview of the problem of anomaly detection and its related properties. In section 2.2, we explicitly consider the time-series data type and the characteristics of dealing with data in the temporal domain. Finally, in the last section, we elaborate on the different types of graph-based approaches and their features that distinguish them from other anomaly detection methods.

### 2.1 Anomaly detection

*Anomaly*, *Outlier*, *Novelty*, *Abnormality* and *Deviant* are different names for data that exhibits some kind of out of the way behavior. In literature, there are multiple definitions for Anomalous data. In a recent book, from a data mining perspective N. N. R. Ranga Suri [16] defines an object is anomalous in a data set if

- It deviates from the normal/known behavior of the data.
- It assumes values that are far away from the expected/average values.
- It is not connected/similar to any other object in terms of its characteristics.

A more intuitive definition for the anomaly is presented by Aggarwal [6]

"An outlier is the data point that is significantly different from the remaining data."

The problem of anomaly detection can be the same as to the problem of finding a pattern in the normal data and point out the abnormal data using the discovered pattern. Normal data points are also called *inliers* in the data mining and statistic literature as they are values that can fit in a discovered pattern.

### 2.1.1 Outlier vs. Novelty

The two terms of *Outlier* and *Novelty* are usually used as follows:

- **Outlier** is a point which does not conform to the others.
- **Novelty** is a point which has not be seen before.

For instance, *Outlier* detection algorithm is usually used when we are considering only the train data and looking for deviant observations in the training data. However, in *novelty* detection, we are examining the data on the test set by letting the model look only at the assumed normal data which is provided as a train set. In other words, in novelty detection, we are looking for the data which have novelty in their pattern compared to the extracted pattern from data in the train set.

### 2.1.2 Supervised vs. Unsupervised learning

An anomaly detection as a learning problem can be applied to the data in three different possible ways considering the availability of the label.

- **Supervised** is when all the true labels for normal and abnormal data is known to the model, we call it a supervised anomaly detection.
- **Semi-supervised** is the case if the model only has partial information about the labels of the data. For instance, the model can only be learned by some normal data and does not have any information on anomalous data.
- **Unsupervised** anomaly detection is when data without a label is provided.

Thus, *novelty* detection is a type of *semi-supervised* and *outlier* detection can be viewed as a *unsupervised* method.

### 2.1.3 Outlier scores vs. Binary labels

An anomaly detection algorithm can have two different types of output [6].

- **Outlier score:** an outlier score simply indicating the level of *outlierness*. The score can represent the tendency of being an outlier. It could also be normalized between 0 or 1 to shows the likelihood. An outlier score near 0 shows the data is normal and values close to 1 express the high probability that the data is an anomaly.
- **Binary label:** a binary label classifying data into the normal or abnormal group. 0 or 1 represents a normal data point and 1 or  $-1$  can indicate the anomalies.

In practice, binary labels output of an algorithm is the final result for decision making and different anomaly detection techniques labeling a data point into normal or anomalous using a *threshold* on the outlier score. In the next part, we discuss more the importance and problems of setting the right threshold.

#### 2.1.4 Outlier score threshold

One of the main difficulties of anomaly detection is its inability to distinguish the anomalies data from noises as both categories show some unusual behavior in the data the model simply labels them as abnormalities. In an ideal model, abnormalities should have a higher level of deviation or *outlier scores* than noises. As a practical solution, usually data analysts tuning the threshold on outlier scores in order to be able to correctly separate anomalies from noises in the binary labeling process. Even the two terms of *Weak outlier* and *strong outlier* in the literature [17] are emphasizing this goal of the model that a noisy data-point has a lower outlier score than an anomaly.

Based on the application and its specific *data type*, *data size*, and *availability of the true labels*, a data analyst have to select the right model and tune the outlier score threshold with respect to the feedback of the model from previous choices with a goal to spot anomalies from the unwanted normal or noisy data. For instance, in an unsupervised anomaly detection model, as we do not have any prior knowledge about the true label of the data, usually some of the outputs of the model will be provided to an expert for further examinations and get feedback. The expert can be a cardiologist for an online heartbeat monitoring system which have a closer look at the result of an alert from an arrhythmia detection model.

Different data representation can affect the effectiveness of an anomaly detection algorithm as different data types can amplify one aspect of the original data. Finding the most effective data representation is not always easy when the whole spectrum of possible anomalies is unknown at design time. In chapter 4, we try to see the potential advantages of the graph representation of the ECG signal on our model.

### 2.1.5 Anomaly detection vs Classification

The two problems of *anomaly detection* and *binary classification* have the following three significant differences that make them suitable for different situations [18].

- **Imbalance data set:** in a binary classification problem there is a large number of samples for both normal and anomaly class. However, we only have a very small number of anomalous data and a very high number of normal examples in an anomaly detection problem. In statistics, we call this type of data set an *imbalance* or *skewed* data.
- **Types of anomalies:** considering a large number of anomalous samples in a binary classification problem, the algorithm has enough number of examples to learn their pattern. On the contrary, having a few numbers of anomalies makes it hard for the anomaly detection algorithm to generalize its learned pattern and being able to identify all different kinds of anomalies.
- **Future anomalies:** as the binary classification problem is able to learn different types of anomalies, a future anomaly is likely to be very similar to one of the anomalies in the training set. However, the future anomalous data point for an anomaly detection problem may not look like any of the previously observed samples.

In figure 2.1, you will see a visualization of the differences between anomaly detection and classification.

There are three main approaches to perform *anomaly detection* in presence of imbalanced data.

- Training on the *majority* class
- Training and well-tune on the *minority* class
- Training on *both* majority and minority class and combining the outputs.

The most common approach is to cast the anomaly detection problem as a *one-class classification* problem and let the model train on a set containing only *normal* data-points then use the model to identify the class of data-points in the test set. Notice the difference between *binary* and *one-class classifier*, one-class classifier is only learning the pattern of the data on the normal class to identify examples which are not normal and don't need to have labels for the two classes as binary classifier needs. Thus, a one-class classifier can be used when data from one of the two classes is difficult to collect and it is hard to characterize with few samples.



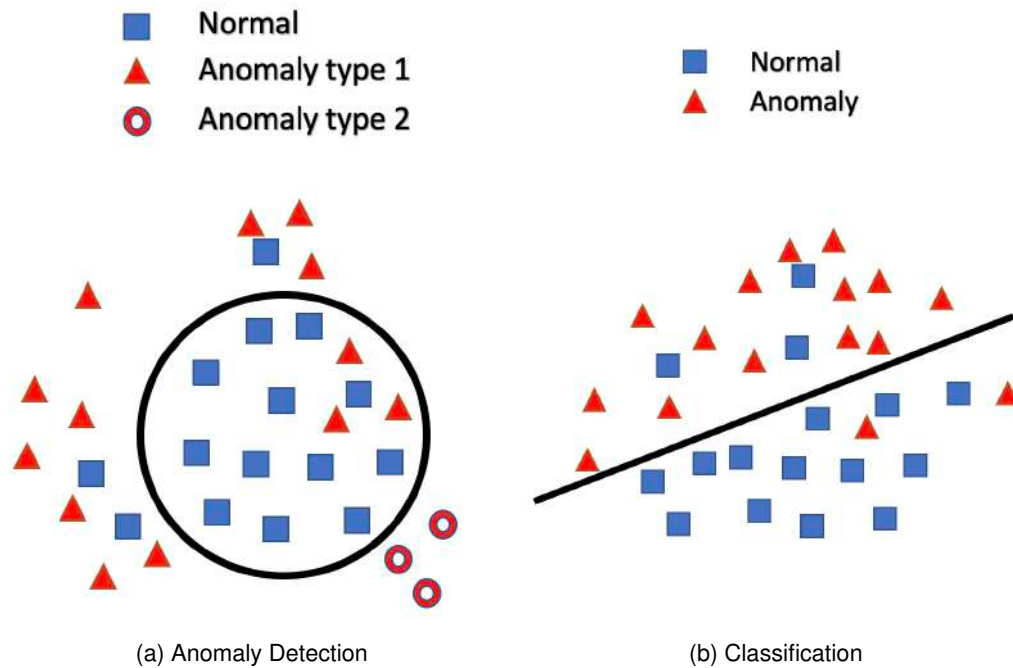


Figure 2.1. The main differences between anomaly detection and classification problem.

In table 2.1, you will see the comparison between anomaly detection and classification problem considering factors such as the learning category, data set, and the number of classes.

Problem	Category	Data set	Class
Anomaly Detection	Supervised Semi-Supervised Unsupervised	Imbalance	Binary
Classification	Supervised	Balanced	Binary Multi-class

Table 2.1. Comparison of Anomaly detection and Classification

Even though one class classifier can be robust to the nature of anomaly when we don't have consistent pattern or structure in the feature space, this method suffers from the fact that any examples of outliers (positive cases) we have during training are not used by the one-class classifier and are discarded. The most popular algorithms for doing one-class classifiers are *One-Class SVM*, *Isolation Forest*, *Elliptic Envelope*, and *Local Outlier Factor (LOF)*. In chapter 3, we explain the algorithm of *Local Outlier factor*

(*LOF*) and in chapter 4, we run our experiments using this method.

### 2.1.6 Quick review of the existing approaches

In [7] survey, the Pimentel et al. divides the novelty detection approaches into the following five major family of models.

- Probabilistic approaches
- Distance-based approaches
- Reconstruction-based approaches
- Domain-based approaches
- Information-theoretic approaches

Here we quickly review these different approaches to understand their unique perspective and mechanism to solve the problem of identifying novelties when we have a very large number of normal data and insufficient data to learn the pattern of abnormalities.

#### Probabilistic approaches

Probabilistic or statistical models are focused on the probability density function (pdf) of the data. These methods are able to find anomalies by a threshold on the pdf. They are labeling anomalous data point as points which have lower probability density compared to the normal population. This big family of models have two subcategories of *parametric* and *non-parametric* models. Parametric methods including algorithms such as *mixture models* and *state-space* models. Non-parametric methods includes *kernel density estimators* and *negative selection*.

#### Distance-based approaches

Distance-based models are depending on precise distance metrics to compute the distance or similarity among data-points. *Nearest neighbors-based* methods and *cluster-based* methods are examples of this group. In chapter 4, we are doing our experiments on a data set of ECG signals using *Local Outlier Factor (LOF)* [13]. The algorithms of this group like Probabilistic approaches are specifying a threshold to labeling the data into normal and abnormal classes. Usually, an anomaly labeled with  $-1$  and normal data as  $+1$

### Reconstruction-based approaches

Reconstruction-based algorithms are working by automating the construction of a model from a normal population and latter on can take on test data as input and produce the construction error as a distance between normal and the test data-point. Two famous subset of this approaches are *neural networks-based* models and *subspace-based* models.

### Domain-based approaches

Domain-based approaches are building a boundary that can separate the normal and anomalous data-points. Using this boundary they are robust to a particular sampling and population of the target class. *Support vector data description* (SVDD) and *One-class support vector machines* (SVM) algorithms are two well-known members of this family.

### Information-theoretic approaches

The final group is Information-theoretic in which we are using the information content of a data set by computing entropy relative or entropy of the data-set. The assumption in this family of models is that the deletion of anomalies from the data-set can create the biggest entropy different.

## 2.2 Time-series data

The data type and the process of generating the data are extremely important factors in selecting the suitable anomaly detection model. Different data types require a different model that will be able to identify their specific type of anomalies from the remaining normal ones. In some data types, we have a kind of relationship between data instances.

For instance, the distance between two data-points is the relation between spatial data, and the order of data is the relation in sequential data. If data arrives in a stream as time-series the time or temporal order is the natural relation between data and the pattern of change in the value of data points can play an important role in identifying anomalies.

### 2.2.1 Time-series properties

In the context of statistical univariate time-series analysis, we have the following significant properties for our series.

- **Trend:** a trend is happening when the mean of time-series ( $\mu$ ) increasing or decreasing over time.
- **Seasonality:** seasonality is the periodic recurrence of fluctuations in the time-series. Notice, a seasonality always has a fixed period.
- **Cycle:** a cycle is a type of seasonality if the period of fluctuation is not fixed and the duration is longer.
- **Level:** level is another name for  $\mu$ , the mean of the series. If the level is changing the time-series has a trend.
- **Stationary:** a time-series that have the same statistical properties over every time interval or window is called a stationary time-series. Due to the importance of this property, I will elaborate more on this property in the next part 2.2.2.

Notice, a time-series with *seasonal* or *cyclic* patterns are sometimes called *periodic* time series.

### Univariate vs. Multivariate Time-series

In the context of time-series data, we have to distinguish between univariate and multivariate time-series. *Univariate* anomalies can be found when we are looking for significantly interesting deviations such as an abrupt change in the distribution of values in a *single feature space* (e.g., a single-lead ECG signal).

However, *multivariate* anomalies can be found in the n-dimensional space of n-features (e.g., 8 lead ECG signals). Looking at distributions in n-dimensional spaces can be very difficult for the human brain, that is why most of the previous works focused on training a model to learn from multiple related signals [8, 9, 19]. At this master thesis, we are more focused on the univariate time-series such as a one-dimensional signal, and in chapter 4, we do our experimental analysis on a single lead ECG signal.

### 2.2.2 Stationary vs. Non-stationary Time-series

Intuitively, a data generating process is stationary when generated data are independent and identically distributed (*i.i.d*) variables which means that their distribution does not change over time. In simple words, stationarity in a time-series means that the statistical properties of the series such as *mean*, *variance*, and *covariance* does not change over time.

In figure 2.2, you will see a non-stationary ECG signal along with its computed moving average (The black line is the ECG signal and the red line is the computed

simple moving average). The moving average is one of the simple methods to analyze the stationarity of a time-series or even prediction on them. Here a simple moving average of the ECG signal is computed by considering a window size of 0.75 seconds and it is very clear from figure 2.2 that the moving average is not constant over time. The general rule is that if the moving average is changing over time the signal is non-stationary.

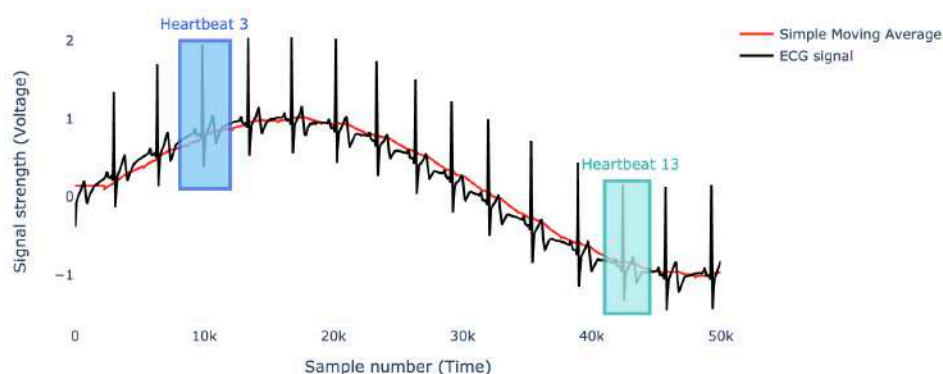


Figure 2.2. A non-stationary ECG signal. The signal is presented with a black line and its computed simple moving average is shown in red. Even though the signal is non-stationary the two indicated heartbeats can be considered independent and identically distributed with respect to each other.

However, if you have a closer look at heartbeat number 3 and heartbeat number 13 of the ECG signal in figure 2.3 these individual heartbeats can be considered independent and identically distributed with respect to each other. Later on in chapter 4, we make use of this simple trick and by splitting the long recorded signal into its heartbeat, we satisfy the stationarity condition for our prediction model to learn on i.i.d variable of heartbeat vectors.

Having a stationary time-series can be good news for anomaly detection or prediction model. Generally, a stationary process is easier to analyze and predict compare to other types of stochastic processes that can generate our data. Intuitively, being stationary meaning that the process that generates the data should be possible to predict as the way they change over time is predictable.

### 2.2.3 Time-variant vs Time-invariant

Another important characteristic of the data generating process for time-series is being *time variant* or *time invariant*. This feature usually is usually discussed in the context of

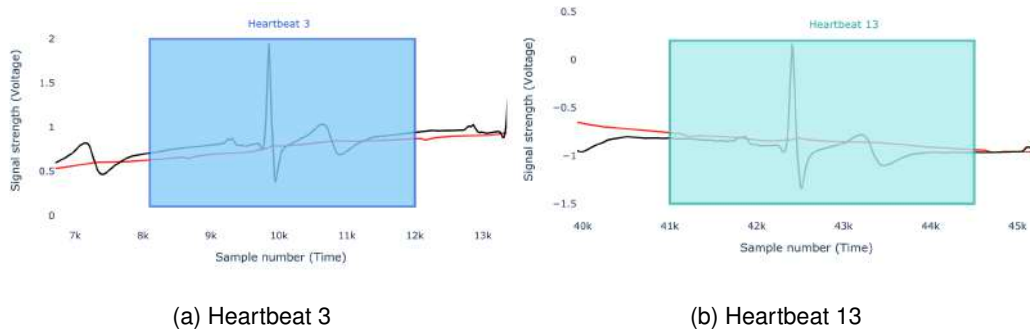


Figure 2.3. Stationarity of each time-interval or window with respect to each other. Notice. the time-series is not stationary but the window variables are i.i.d variables and stationary.

the systems. A system is said to be time-invariant if its input and output characteristics change with time. In other words, the process is time-variant when its output does not explicitly depend on time. In chapter 4, by considering an ECG signal as a univariate time-series, we assume that the person's heart is not changing significantly over the recording time and if so happen it would be an anomaly which our model will be able to detect. Thus, our system is assumed to be time-invariant.

### 2.2.4 Types of Anomalies

For time-series data, Chandola et al. [20] categorized anomalies into three different types.

#### Point Anomalies

point anomalies are the simplest type of anomalies and it defines as an individual data point that deviates from the rest of the data. For instance, a data point in a time that has a very high value or low value in a signal. In figure 2.4, an example of point anomaly in an ECG signal is presented.

#### Contextual Anomalies

A contextual anomaly happens when the data instance is anomalous in a specific context (or neighborhood) such as a sudden increase or decrease in a value of a data-point which may be in range of normal changes but its value is to different than its neighboring data-points. Figure 2.5 shows an example of contextual anomaly in an electrocardiogram signal.

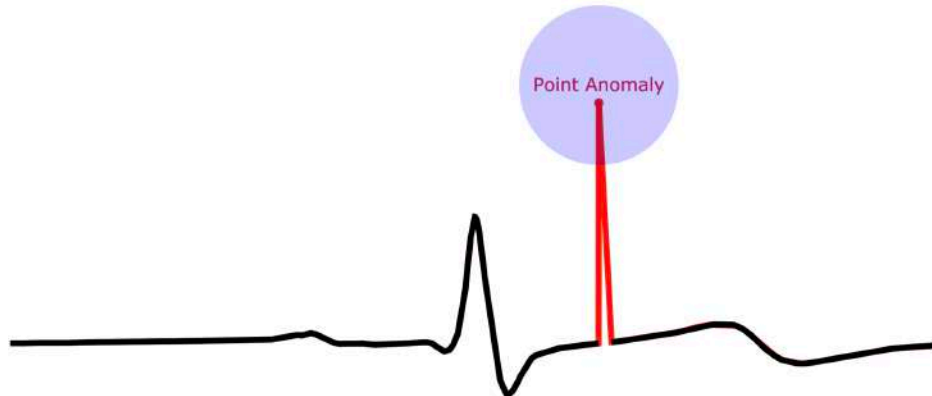


Figure 2.4. An example of a point anomaly in an ECG signal. The value for the individual point is unusually higher than the normal range between peaks.

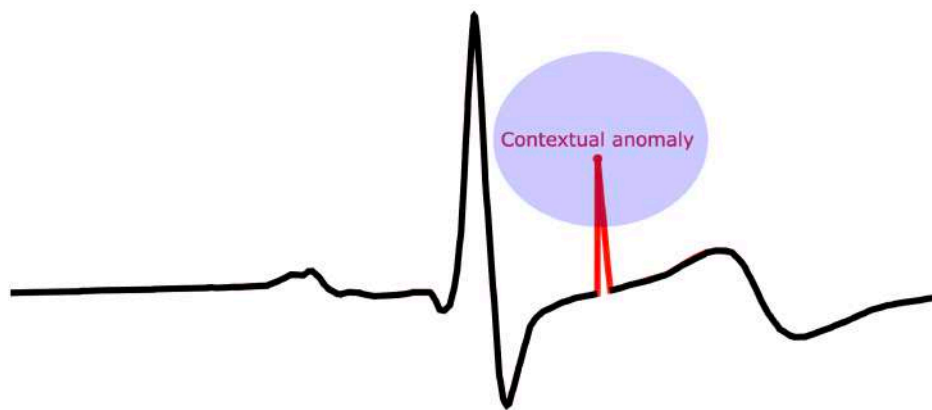


Figure 2.5. An example of a contextual anomaly in an ECG signal. The value for the individual point is in range or normal values, however, the value is abruptly change compare to its context (neighborhood).

### Collective Anomalies

Collective anomaly is the hardest type of anomaly to detect and it happens when a collection of single data-points are happening together in a specific order. Each of these single data-points is not anomalies but together in the special order, they create abnormality with respect to the entire data-set. Chandola et al. [20] called this sequence of individual data-point that cause anomaly an *anomalous collection*. In figure 2.6, you will see a collective anomaly in an electrocardiogram signal. Collective anomalies can be a subset of novelties in data that may indicate the discovery of new phenomena such as a heartbeat arrhythmia or replacement of the recording patch on the skin in an online ECG signal.

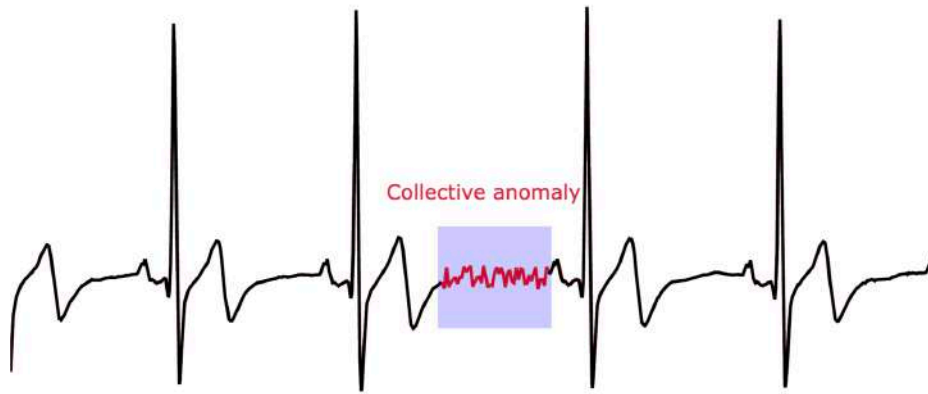


Figure 2.6. An example of a *collective anomaly* in an ECG signal. Each point in the collection is not anomalous itself but together they are anomalies.

#### 2.2.5 Time-series models

According to [21], anomaly detection on a univariate time-series is strongly related to time-series forecasting in which we try to predict future values based on previously observed values. Most of the statistical approaches for univariate time-series analysis are considering a sliding window on the time-series with the size ( $\omega$ ) that the model is learning based on the data-points inside this window and making a prediction for the next data-point ( $\hat{x}_i$ ). This problem can be formulated as equation 2.1 in which  $\psi$  is the prediction model.



$$\begin{aligned}\psi : \mathbb{R}^\omega &\rightarrow \mathbb{R} \\ \hat{x}_i &= \psi(x_{i-\omega}, \dots, x_{i-1})\end{aligned}\tag{2.1}$$

The anomaly score for this approach is the distance between the prediction of the model ( $\hat{x}_i$ ) and the true value ( $x_i$ ). 2.2

$$s_i = d(\hat{x}_i - x_i)\tag{2.2}$$

The distance function  $d$  can be any distance metrics such as *Euclidean distance*. Finally, the model is employing a threshold of  $\delta$  to label the data as anomalous or normal.

However, there are also other models that do not follow this approach and make use of the unusual shapes of the anomalies to identify them in a time-series such as *one-class Support Vector Machine (SVM)* [22]. It is important that in both of these two models we are searching for a significant deviation between the next data-point and the previously observed data in our sliding window. Through this thesis, the terms *time intervals*, *windows* and *vectors* of a time-series all refers to the observed data inside a sliding window of our time-series.

The three most popular statistical process models which can be used for both anomaly detection or forecasting on a time-series data are the followings:

- *AutoRegressive (AR) model*
- *Moving Average (MA) model*
- *AutoRegressive Moving Average (ARMA) model*

### **Autoregressive (AR) model**

In an *Autoregressive (AR)* model, we assume that the output is generated as a linear function of its past values plus a noise or error. 2.3

$$X_t = c + \phi_1 X_{t-1} + \dots + \phi_p X_{t-p} + \varepsilon_t\tag{2.3}$$

AR model is an instance-based (memory-based) learning model, in the sense that each instance is in correlation with the  $p$  preceding instances. In equation 2.3, co-

efficients  $\phi_i$  are weights that measuring the influence of each preceding instance,  $c$  is a constant intercept and  $\varepsilon_i$  is a univariate white noise process which is commonly assumed to be Gaussian noise.

Another improved version of AR model is *VAR Vector AutoRegressive* that simply can be applied for multivariate time-series by changing the  $\phi$  weight from a scalar value to a vector of size  $K \times K$  which  $K$  is the number of variables in our multivariate time-series.

### Moving Average (MA) model

Another famous stochastic process model is *moving average (MA)* that denoted by  $MA(q)$  which  $q$  specifying the order of the moving average. Equation 2.4 specifying that the output value at each time will be generated as a linear function of the last  $q + 1$  random shocks generated by a univariate white noise generator  $\varepsilon_i$  plus the mean of the series  $\mu$ . Same as AR models, MA models can also be applied to multivariate time-series by using a multivariate noise process instead of  $\varepsilon_i$ .

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} \quad (2.4)$$

In figure 2.2, for computing the red line that indicates the simple moving average, we assumed all the errors as zero and simply just compute the  $\mu$  as output. However, usually,  $\mu$  will be assumed to be zero.

### AutoRegressive Moving Average (ARMA) model

A more complex stochastic process model that we have a quick look at it is *AutoRegressive Moving Average (ARMA)*. An  $ARMA(p, q)$  time-series model generating the next instance as a linear function of the last  $p$  instances and the last  $q + 1$  random shocks generated by a univariate while noise generator  $\varepsilon_i$ . Equation 2.5 presenting this linear function.

$$X_t = c + \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} \quad (2.5)$$

Even though the ARMA model can be effective in many applications, it has its own challenges such as specifying the correct  $p$  autoregressive terms and  $q$  moving-average terms and tuning the coefficients. This model can be modified to handle multi-variate time-series in *VARMA* or deal with a specific type of non-stationary data (*ARIMA*). The

main downside of this model is that every time we work with new time-series data we have to build a new model.

## 2.3 Graph-based approaches

A *Network* represents the pairwise relationships between different objects. We call these objects as vertices or nodes and the relations as edges or links. Networks and the connection between objects are almost everywhere in any field such a physics, social science, and biology.

A powerful mathematical abstraction of the network is *Graph*. A graph  $G(V, E)$  is formally defined by its set of vertices  $V$  and its set of edges  $E$  connecting those vertices. An edge  $e \in E$  between two nodes  $u, v \in V$  will be represented by  $e = (u, v)$ . Graphs can be attributed by having different attributes for their nodes and edges such as weight and sign. If the edges of a graph have direction, we call the graph a directed graph and otherwise it will be an undirected graph.

Two nodes which are having an edge between them are called neighboring nodes. The degree of a node  $d_v$  is the total number of edges or neighbors that the node has. The *degree-distribution*  $P(k)$  is the probability distribution of the degrees for all the nodes in a graph. In other words,  $P(k)$  specifies the fraction of nodes with the degree of  $k$  as  $n_k$  from the total number of nodes as  $n$ .

$$P(k) = \frac{n_k}{n} \quad (2.6)$$

### 2.3.1 Why graph representation?

The main reasons why we believe that a graph representation could be beneficial to spot anomalies in our data are listed as follows.

- **Different natures of the data:** Graphs are powerful abstract structures to store data and can be used to represent different measured values regardless of the nature of the data
- **Generalize approaches:** Many approaches neatly generalize when thought them in graph terms. If you consider an attributed graph and its ability to encode different types of values inside the nodes and edges, graphs will be able to apply to almost any problem to show their dependency or relation.

- **Effective for time-series machine learning:** As machine learning can be considered as learning the existing patterns in data. Using graphs machine learning, we can create functions to spot recurring patterns for time-series data. Remember that in a time-series, the relationship between data points is the time order and the pattern of change in the value of data points over time can be recurrent.
- **Capture dependency structure:** Attributed graphs can be considered as suitable mathematical tools to capture the dependency or local dependency in the data. In the context of time-series, each data-point is dependent on its previous and next neighbors and for capturing this relation, attributed graphs with weight attributes for each edge and nodes are suitable structures. Thus, the *local dependency network* between samples of a time-series can be easily encoded by a structure or topology or a graph.

Considering the aforementioned advantages of the graph representation of a time-series data, doing machine learning on a graph domain can be defined as identifying the patterns in our graph database. In the problem of anomaly detection, we want to learn the pattern from a normal graph and using this recognized behavior of the data spot anomalous graphs in our graph data set. [23]

### 2.3.2 Static vs. Dynamic Graphs

The graph databases can be of two different kinds according to their generation mechanism from a time-series.

- **Static graphs:** This type of graph  $G = (V, E)$  are not changing over time and their structures and attributes are fixed. They can have different attributes for each node and each edge or just be plain graph to capture the structure (topology) of the dependency network between node entities.
- **Dynamic graphs:** The second type of graph databases are changing over time. A *dynamic graph* denoted as  $G_t = (V_t, E_t)$  is usually built based on a multivariate time-series data to capture the inter-dependencies between multiple recorded signals. [8, 10, 24]

Another name for *dynamic* graphs is *time-evolving* graphs which can be considered as a sequence of static graphs. In practice, there is two possible way to build a dynamic graph from time-series data based on the type of time-series.

- *Multivariate time-series:* A dynamic graph can be generated from the *inter-dependency network* between different dimensions of a multivariate time-series

- *Univariate time-series*: first, the *local dependency network* inside each window of the univariate time-series will be encoded in a static graph. Then, the dynamic graph can be built from the stream of static graphs generated for windows of a univariate time-series.

### 2.3.3 Graph-based models

In [23], the authors categorized different anomaly detection approaches for both *static* and *dynamic* graphs. At this master thesis, we are more focused on the *dynamic* or *time-evolving* graph databases. The graph-based anomaly detection on static graph databases is more concentrated on the node or edge anomalies inside a static graph. However, the anomaly identification methods designed for dynamic graphs can be used to labeling a graph into two classes of normal or anomaly.

Four family of models for doing anomaly detection on *dynamic* graphs from the prospective of Akoglu et al. [23] are the following

- *Feature-based*
- *Decomposition-based*
- *Community or clustering-based*
- *Window-based*

#### Feature-based models

The underlying assumes in feature-based models is that if two graphs are similar using a measure of similarity they should be similar in their other properties. For instance, in the problem of anomaly detection, if the two graphs are very similar considering their topology, they should belong to the same normal or anomaly class. The measure of similarity or distance metric between two graphs has to be selected with respect to each problem. In a nutshell, there are three main steps to follow in a feature-based model.

- **Extract features**: extract suitable features or an appropriate summary for each graph.
- **Compare features**: compare the two features using the relevant similarity measure.
- **Define threshold**: select a manual or automatic well-tuned threshold to identify graphs based on the computed distance of the graph from other graphs.

Some of the most popular distance metrics for having a measure of similarity between two graphs are:

- *Maximum Common Subgraph (MCS)*
- *Graph Edit Distance (GED)*
- *Hamming distance for the adjacency matrices*
- *NetSimile* [15]

In this master thesis, we do our experiments using the *GED* and *NetSimile* similarity metrics for computing the distance between our graphs. *NetSimile* distance measurement between graphs consist of three following phases:

- *Feature Extraction*: they extract some local and egonet-based features (egonet-based feature are like the number of neighbors, clustering coefficient, average of neighbors degrees)
- *Feature Aggregation (Create Signature vector)*: the node $\times$ features matrix of the first phase is converted to a single "signature" vector that consists of the median, mean, standard deviation, skewness, and kurtosis of each extracted feature over all the nodes in the graph (In our implementation the size of the signature matrix is always 35 for a variable size graph.)
- *Signature vector Comparison*: the signature vectors are compared using the Canberra Distance and a similarity score is produced for adjacency graphs in the stream.

In chapter 3, we discuss *GED* similarity measurement in detail.

### **Decomposition-based models**

Decomposition-based models are exploiting the matrix of the graphs to compute the *eigenvectors*, *eigenvalues* or *singular values* of graphs. Using the computed decomposition, these algorithms compare graphs and set a threshold to label them as normal or anomalous. One can claim that these models are a type of feature-based model with the only difference in extracting the feature of a graph using specific tools such as Singular Value Decomposition (SVD), eigenvalue decomposition, or Compact Matrix Decomposition (CMD).

### Community or Clustering-based models

The idea behind the community-based models is that they are only focusing on the changes in the community or clusters of the graphs instead of comparing the whole graphs. Notice, this family of models is more concentrated on the change detection problem which is a slightly different problem and can be also be considered as novelty detection.

### Window-based models

The last type of model that we discuss is the window-based models that are based on the idea of only considering the last graphs that are inside our moving window. In other words, the model always looking at a fixed number of previous instances to identify the pattern of normal graphs and the incoming graph is compared only against those graphs in the range of our sliding window.

## 2.4 Model selection

In this chapter, we discussed different families of models that each categorized the existing approaches from slightly different angles. In 2.1.6, we divided all the approaches from the novelty detection perspective. In 2.2.5, we discussed three popular statistical models designed for time-series data. Finally, in 2.3.3, we categorized the anomaly detection models which are specifically intended to be applied on a graph database.

Despite the effectiveness of different models, to the best of our knowledge not every one of the above methods is feasible to work with a huge size graph database and be effective for both graph and normal vector shape data type. *Distance-based* or *feature-based* models are among the most suitable approaches if we are able to provide these algorithms with a satisfactory similarity or distance function. Hausdorff Edit Distance (HED) [14] and NetSimile [15] could be the right choices of distance function on our data set of graphs. In chapter 3 and 4, we discuss them in detail and justify on our choices.





## Chapter 3

# Proposed methodology (GrAnD UniTs)

In this chapter, we are proposing our new methodology "a *Graph-based Anomaly Detection for Univariate Time-series*" for short (*GrAnD UniTs*) to studying the effectiveness of employing graph representations in the problem of finding anomalous time intervals (window) on a univariate time-series.

Our methodology, "GrAnD UniTs", contains the following three main phases as shown previously in figure 1.1 in the first chapter.

- **Vector stream from time-series:** creating a sequence of vectors by splitting our one-dimensional time-series.
- **Graph stream from vector stream** generating a graph for each of the vectors containing timely order samples.
- **Anomaly detection on graph stream** applying a suitable anomaly detection algorithm on our graph stream and finally produce the anomaly binary label for each graph.

Before explaining each phase of the *GrAnD UniTs* in detail, let me elaborate on the intuition behind our methodology. We want to construct an anomaly detector that looks only at local qualitative relations between the data points, like whether one is above or below its neighbors. Differently from standard methods, this would show that the specific numeric values are not always necessary, but relational property can suffice.

According to Braei and Wagner [21], an original univariate time-series as it is clear from its name is in the *Temporal domain*. There are multiple possible mechanisms to

transfer the data from the time domain to *frequency domain* such as Discrete Fourier Transform (DFT) to analyze the same data looking at the changes in its frequency. Here our idea is why not transforming the series into the *graph domain* so that we can analyze the changes in the dependency network. In table 3.1, you will see the differences between each domain and their pursued analysis for each representation of the data.

Data representation	Domain	Analysis
Time-series	Temporal Domain	Time-series statistical analysis
DFT	Frequency Domain	Frequency analysis
Graph stream	Graph Domain	Dependency analysis

Table 3.1. Comparison between Temporal domain, Frequency domain and Graph domain and their respective pursued analysis

This chapter is organized as follows. In section 3.1, we discuss stationarity reason behind splitting an univariate time-series into a vector stream. In section 3.2, we discuss our mechanism to generation graph stream from a sequence of vectors. In particular, we explain the *Visibility Graph* algorithm and its power to capture the local dependency network of a time-series. Finally, in section 3.3, we will elaborate on the feasible anomaly detection methods to learn on our generated graph stream. Specifically, we explain the *Local Outlier Factor (LOF)* algorithm and the distance metrics that we use for finding the similarity between graphs.

### 3.1 Vector stream from univariate time-series

As mentioned in chapter 2, a time-series is stationary if its statistical properties do not change over time. In practice, most of the time, our univariate time-series data are non-stationary. In "*GrAnD UniTs*", by creating a sequence of vectors each corresponds to a specific *seasonality* or *cycle* in our non-stationary time-series, we reach a nearly stationary vector stream. The process to split the non-stationary time-series into its vector stream is consists of the following three steps as shown in figure 3.1.

- Find periodic events
- Find periodic patterns
- Create vector for each periodic pattern

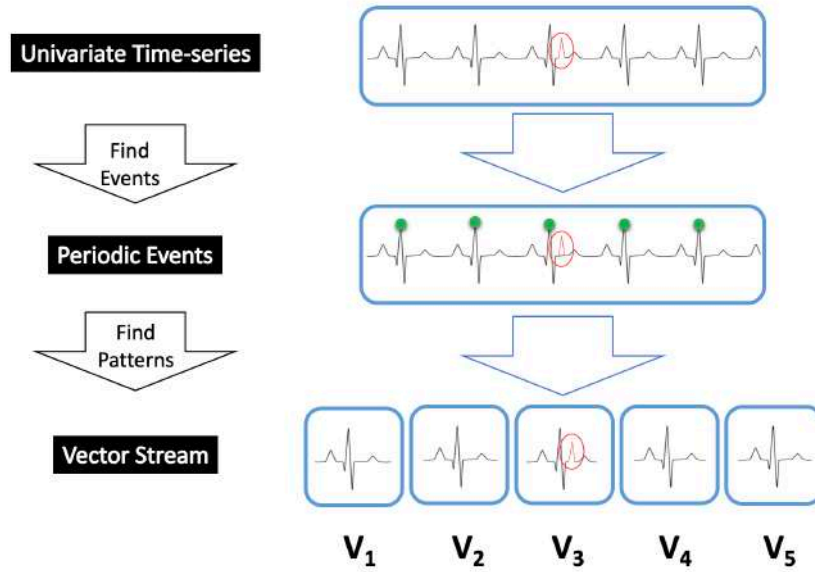


Figure 3.1. Steps to generate a vector stream from a periodic univariate time-series. Periodic events are denoted by Green dots and periodic patterns are the area around events data points.

### 3.1.1 Find periodic events

According to [25, 26], "An event is a real-world occurrence that takes place in a specific location and time that relates to a particular topic." Events in a time-series can be of different kinds and tightly related to the nature of the data. Thus, considering the application and its specific data generation process, first, we find the data points in our time-series that a *periodic event* happened. A *periodic event* intuitively means an event that happened periodically over time in our series. The duration of a *periodic pattern* can be fix or variable regarding its type.

### 3.1.2 Find periodic patterns

After finding the *periodic events* in our periodic non-stationary time-series, we search for *seasonal* or *cyclic* patterns using a sliding window which its center fixed at the data points of the time-series that each event happens. In practice, our *events* could be the list of data points in our series that particular action happened in the real world, and our *periodic patterns* are the sequences of data points around each event.

For instance, in figure 3.1, we will consider each *R-peak* of an ECG signal as an event and the heartbeat is our periodic pattern in the signal. Notice, for an ECG signal the

heart of the person is our data generator and we assumed that the heart of the person that generates data is stationary over time.

### 3.1.3 Create vector for each periodic pattern

After having all the *periodic patterns* in our non-stationary time-series, we simply create a vector from all the sequential data points in each periodic pattern. As the data points inside each periodic patterns are ordered in time the generated vectors from them can be considered as small size time-series.

Notice, after converting a non-stationary time-series to its vector stream, our vector variables are nearly independent and identically distributed (i.i.d) with respect to each other. For example, in figure 2.2 that shown in chapter 2, the signal is non-stationary as its *moving average* or *level* is changing over the time. However, in figure 2.3, the two vector, *Heartbeat 3* and *Heartbeat 13* are nearly stationary with respect to each other. The vector or periodic pattern in an ECG signal considered as a heartbeat.

## 3.2 Graph stream from vector stream

As graphs are powerful mathematical structures to capture the relativity or connections between objects, we can use them to encode the dependency network between data-points inside each vector of our stream. In the literature, graph generation algorithms are often used to capturing the inter-dependency network between components of multivariate time-series. [8, 9, 10, 19]

However, in this thesis, we are interested to build a graph stream from a univariate time-series. Figure 3.2 highlighting our main focus at this phase of "*GrAnD UniTs*" methodology which is capturing the shape of the signal inside each vector, the relative position of the data points, using the *Natural Visibility Graph* [27].

### 3.2.1 Visibility Graph

In order to map form a vector stream to a graph stream, we have to define a set of nodes  $V$  and and set of edges  $E$  connecting them for each graph. Formally, the visibility graph of a time-series  $y_1, y_2, \dots, y_N$  with  $N$  data-point is defined as a graph with  $N$  nodes corresponding to each data-points. Two nodes  $i$  and  $j$  in our time-series  $i < j$

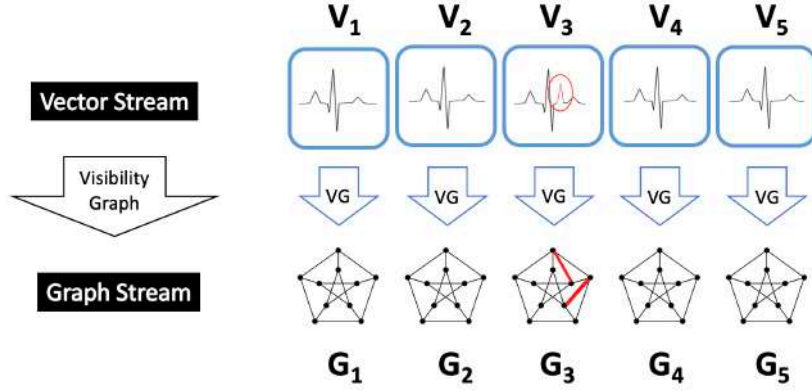


Figure 3.2. graph stream from vector stream using the *Visibility Graph* algorithm to capture the shape of the signal inside each vector.

are connected if and only if for every data-points  $k$  between  $i$  and  $j$  the inequality

$$y_k < y_i + (y_j - y_i) \frac{k-i}{j-i} \quad (3.1)$$

In simple words, two nodes  $i$  and  $j$  are connected if the line between the two points  $(i, y_i)$  and  $(j, y_j)$  is above  $(k, y_k)$ , for every  $i < k < j$ . In figure 3.3, you will see an example univariate time-series and its corresponding visibility graph.



Figure 3.3. A univariate time-series and its corresponding visibility graph.

The visibility graph that represented here is known as *Natural Visibility Graph (NVG)* and there are multiple variations of this algorithm such as *Horizontal Visibility Graph (HVG)* [28] and *Sign Visibility Graph (SVG)* [29]. A horizontal visibility graph is a limited version of Natural visibility that only considering the horizontal lines which

are connecting the two nodes. Sign visibility graph as a recently developed variation of natural visibility graph is simply having the idea of building and combining two natural visibility graphs as *Positive Visibility graph* and *Negative visibility graph*. *Positive Visibility graph* is a simple NVG and *Negative visibility graph* is resulting by multiplying every value of the time-series with  $-1$  and then applying the NVG algorithm on the negative version of the series.

Our initial three main motivations to build the *Natural Visibility Graph (NVG)* of the time-series vectors are as follows.

- **Direct implementation:** the direct implementation of the NVG and its proven effectiveness to capture the change in the data generation process using the *degree-distribution*  $P(k)$  [4].
- **Reversible process:** an extension of the algorithm can make it possible to build a vector stream by having a graph stream. In short, we propose by computing the attributed graph from each vector, making the process reversible. The attributed graph containing the following information.
  - *Visibility Line Slope:* each edge can also contains the *slop of the visibility line* between two nodes as the edge extra attribute
  - *Quantitative values:* each node can contain the *quantitative value* of the data points.
- **Identify nonlinear correlation in the time-series:** according to Ji et al. [4], using a visibility graph can help us to capture the presence of nonlinear correlations in time-series and distinguish white noise from chaotic series.

However, the *reversibility feature* of the resulting NVG has not been experimented at this work due to time constraints.

### 3.3 Anomaly detection on graph stream

In the last phase of "*GrAnD UniTs*", after generating a stream of graphs from our vector stream, we have to face the problem of anomaly detection on our graph stream. However, processing graphs which are very complex data structures have their own difficulties such as the huge size of the graph data and very high computational complexity to process them. For instance, the computational complexity of the *graph isomorphic*

problem, intuitively, two graphs  $G_1$  and  $G_2$  having the same number of nodes and edges that can be mapped to each other, belong to *NP (complexity)* class.

From all the aforementioned family of models in chapter 2, for doing anomaly detection on our graph streams, we selected *Local Outlier Factor (LOF)* [13] for the following grounds:

- **Distance-based model:** as the local outlier factor method is a proximity-based model, it works with distances among entities. This simple property of the model makes it possible to be applicable to any type of data as far as we are able to compute a distance between pairs data points.
- **Robust to the availability of label:** LOF model can be applied in both *supervised* and *unsupervised* format; the model also can be run in semi-supervised mode when we have partial labels.

Therefore, the local outlier factor algorithm makes it possible for us to apply anomaly detection on a graph stream. Moreover, the same method can also be run on multivariate streams, which allows us to make a fair analysis comparing our graph-based strategy against more standard ones.

### 3.3.1 Local Outlier Factor (LOF)

Formally, the LOF model can be defined as follows. Let's consider a set  $\mathcal{D}$  of data-points  $x_i$ ,  $i = 1, \dots, N$  in a domain  $\mathcal{X}$ , and consider a distance measure  $\psi : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  defined over it. For the data-point  $x \in \mathcal{D}$ , we define k-distance =  $\delta_k$  of  $x$  as

$$\delta_k = \psi(x, y_k) \quad (3.2)$$

where  $y_k$  denotes k-th nearest neighbor of  $x$  in  $\mathcal{D}$  and  $\psi$  is a distance function. We also define the k-distance neighborhood =  $\mathcal{N}_{k\text{-distance}}$  of  $x$  as follows

$$\mathcal{N}_{k\text{-distance}}(x) = \{y | y \in \mathcal{D}, \psi(x, y) \leq \delta_k\}. \quad (3.3)$$

In other words,  $\mathcal{N}_k(x)$  is the set containing all of the k-th nearest neighbors of  $x$ .

Then, we define the *reachability distance* as

$$RD_k(x, y) = \max\{\delta_k(y), \psi(x, y)\}. \quad (3.4)$$

From the reachability distance, the *local reachability distance* is computed as

$$LRD_k(x) = \left( \frac{\sum_{y \in \mathcal{N}_k(x)} RD_k(x, y)}{|\mathcal{N}_k(x)|} \right)^{-1} \quad (3.5)$$

Finally, we are ready to define the *local outlier factor* for each  $x$  as follows:

$$LOF_k(x) = \frac{\sum_{y \in \mathcal{N}_k(x)} \frac{LRD_k(y)}{LRD_k(x)}}{|\mathcal{N}_k(x)|} \quad (3.6)$$

The LOF algorithm has the complexity of  $O(n^2)$  which can be considered as one of the challenges when dealing with the huge size graph data.

Notice, *LOF* algorithm can be applied to any data type if we specify the suitable distance function  $\psi$ . A popular distance function or similarity function for data represented in the graph domain is the *graph edit distance (GED)* measure of similarity. In 3.3.2, we discuss these similarity metrics that can be applied to graphs. However, in chapter 4, we explain the difficulty of working with *GED*, and its ineffectiveness for our data set. Also, we tried using a modified version of a different distance metrics for graphs called *NetSimile* [15]. In 3.3.3, we explain this algorithm in short and in 4, we make use of its fast computational method.

### LOF Output

The output of the *local outlier factor (LOF)* algorithm is an *outlier score* that can be transform into a *binary label* of *Normal* or *Anomaly* using a threshold. In the original algorithm, the default threshold is considered as 1.5 the computed local outlier factor  $LOF_k(x)$  can be in any of the following ranges:

- $LOF_k(x) \simeq 1$ : the local density of the data-point is same as the neighbors.
- $LOF_k(x) < 1$ : it is an indicator that the data-point is *inlier*.
- $LOF_k(x) > 1$ : it means that the local density is lower than neighbors and it is a possible outlier.
- $LOF_k(x) > 1.5$ : the local density is lower than a neighbors and we have to label the data point as outlier.

When we use *LOF* to produce the *binary label* output, the algorithm produce either of the following two labels.



- 1: it means that the data point is normal and its outlier score is below the threshold.
- -1: it indicates that the data point is anomalous and its outlier score is above the threshold.

In fact, the LOF algorithm originally only designed for *outlier* detection as it is clear from the name then later on they extend the algorithm to be able to also applied for novelty detection. As a reminder, *outlier* is a point that does not conform to the other points in our data set and *novelty* is a point that has not been seen before in the data (For the clear definition refer to 2.1.1). In "GrAnD UniTs", we call both outlier and novelty as an anomaly.

#### **LOF *hyper-parameter* tuning:**

Like any other anomaly detection model *LOF* algorithm requires *hyper-parameter* tuning to optimize the performance of the model on any specific data with considering the availability of the true labels of the data. For the *LOF* algorithm, we have the following two hyper-parameters which required to be tuned.

- **Contamination ( $c$ ):** it specifies the amount of contamination of the data set. In other words, it defines the proportion or percentage of anomalies in the data.
- **Number of neighbors ( $k$ ):** the  $k$  number of neighbors specify the number of neighbors of the data points that the local density of the point is compared to them.

As a side note, actually *Contamination ( $c$ )* is not hyper-parameter and is the threshold that we use to binary labeling our data points. However, this threshold or hyper-parameter is very important and needs careful tuning so we consider it in our hyper-parameter tuning phase.

As mentioned in chapter 2, tuning of our *outlier score threshold* and *hyper-parameter* have to be performed with respect to the feedback of the model from previously chosen values with the goal of identifying *anomalies* from the unwanted normal or *noisy data*. Here in our methodology, "GrAnD UniTs", the feedback or the performance of the algorithm will be computed using some *quantitative evaluation* metrics such as the *false positive rate* and *false negative rate*. Moreover, *receiver operating characteristics (ROC) curve* have been used to select a suitable threshold. In chapter 4, we tune our model on a real data of ECG signal using these performance metrics.

### 3.3.2 Graph Edit Distance (GED)

The Graph edit distances (GEDs) are a family of distances between two graph  $g_1$  and  $g_2$ , denoted  $GED(g_1, g_2)$ , which assess the cost of constructing  $g_2$  by editing  $g_1$ . A GED is based on set edit operations among the following

- Adding node: adding a new node to a graph
- Delete node: delete a node from the graph
- Replace node: changing a node with a new one. This can be same as changing the name of the node or changing any of its attributes for an attributed graph
- Adding edge: adding a new edge between two existing nodes in a graph.
- Delete edge: delete an existing edge between two nodes of the graph
- Replace edge: changing an existing node or changing the attributes of the edge for an attributed graph.

The set  $\mathcal{P}(g_1, g_2)$  contains all edit paths, i.e., ordered list  $(e_1, e_2, \dots, e_n)$  of edits, that can transform graph  $g_1$  into  $g_2$ . By associating a cost  $c(e) \geq 0$  to each edit operation  $e$ , we can define the following graph edit distance

$$GED(g_1, g_2) = \min_{(e_1, \dots, e_n) \in \mathcal{P}(g_1, g_2)} \sum_{i=1}^n c(e_i). \quad (3.7)$$

The graph edit distance can be applied to a plain or attributed graph.

In simple words, *graph edit distance* is trying to find minimum cost sequence of operations needed to change the graph  $g_1$  into graph  $g_2$ . Like any other distance metrics,  $GED$  is symmetric meaning that  $GED(g_1, g_2) = GED(g_2, g_1)$ .

As a side note, variations of the graph edit distance algorithm ignore the replacement operators and, instead, they delete a node and add a new one. Moreover, some other algorithms, introduced additional operators such as *edge splitting* that introduces a new vertex into an edge and *edge contraction* that eliminates vertices of degree two between edges.

The exact algorithm to compute the GED between graphs requires to find the edit path with minimal cost. This path can be computed by the A\* search algorithm and it is an NP-complete algorithm. The high complexity of the algorithm makes it practically

inefficient for computing the GED between two large graphs. Thus, multiple approximate algorithms are proposed to find an approximate measurement of GED such as Hausdorff Edit Distance and Greedy Edit Distance [14]. In chapter 4, we explain the differences between using an approximate algorithm and the exact one.

### 3.3.3 NetSimile (a size-independent network similarity)

*NetSimile* [15] is a graph similarity method initially designed for discontinuity detection in a graph stream. In a nutshell, *NetSimile* graph distance metric contains three main steps:

- **Feature Extraction:** extract some local and neighborhood features for each node of the graph. Features are such as number of neighbors, clustering coefficient and average of neighbors' degrees.
- **Feature aggregation (Signature vector):** from the feature matrix of the first step, we aggregate all the values for all nodes and producing a single vector called *signature*. The size of the input feature matrix is  $(nodes \times features)$  and the size of the signature vector is equal to the number of features. In our implementation, in chapter 4, the size of signature vector is 35 for every size graph.
- **Comparison:** we compare the *signature vectors* of graphs using the *Canberra Distance*, and we produce a single similarity score for adjacent graphs in our graph stream.

Note, here at "*GrAnD UniTs*", we use only the signature vectors of the graphs. This is due to the fact that our anomaly detection algorithm *LOF* requires a similarity measurement that can be applied for every pair of graphs. After computing the *signature vector* for all the graphs, we simply compute the similarity between all the signature vectors using the euclidean distance.

### 3.3.4 Quantitative Evaluation

In this thesis, we compute the performance of our anomaly detector using the following standard *quantitative evaluation* metrics:

- True positives (TP): is the number correctly detected anomalous heartbeats
- False positives (FP): is the number of incorrectly detected anomalous heartbeat
- True negatives (TN): is the number of correctly detection normal heartbeat

- False negatives (FN): is the number of incorrectly detected normal heartbeat
- Area under curve (AUC): is the area under the receiver operating characteristics (ROC) curve.
- $Recall = Sensitivity = True\ positive\ rate\ (R_{TP}) = \frac{(TP)}{(TP)+(FN)}$
- $Precision = \frac{(TP)}{(TP)+(FP)}$
- $False\ positive\ rate\ (R_{FP}) = \frac{(FP)}{(FP)+(TN)}$
- $Accuracy\ (ACC) = \frac{(TP)+(TN)}{(TP)+(TN)+(FP)+(FN)}$
- $F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$

Each of these metrics can be selected for tuning the model or selecting a model over another. However, not all of them always suitable for the problem of anomaly detection because they are originally matrices that have been used for classification methods which are a slightly different problem than anomaly detection. In this master thesis *F1 score* could be the best selection for tuning a model parameter for the following reasons.

- Ideally, in a good model, we want both precision and recall to be 1 which also means *FP* and *FN* are 0.
- The *accuracy* may not be a good measure if the data set is not balanced (both negative and positive classes have a different number of data instances).
- *F1-score* is a metric which takes into account both *precision* and *recall* and it becomes 1 when *precision* and *recall* are both 1. Thus, *F1 score* is the harmonic mean of precision and recall and is a better measure than accuracy for our database.

## 3.4 Challenges

In this section, we categorized different types of challenges that one may have to face when implementing our methodology into three groups of (*GrAnD UniTs*): *Data-specific challenges*, *Problem-specific challenges*, and *Graph-specific challenges*.

### 3.4.1 Data-specific challenges

- Huge size of time-series data set: Graph representation of a time-series and having a graph stream can be very challenging if we consider the *scale of the data* that are being recorded in a very high-frequency rate.

- Periodic event detection: identifying the periodic events in time-series data is not always easy and requires extra knowledge about the nature of the data and the process of generating the data.
- Periodic pattern detection: deciding about the possible size of the sliding window and the strategy to split the time-series into fixed or variable size vectors are the questions that need a deep understanding of the nature behind the data and tightly related to the data generation process.
- Different types of anomalies: looking for each univariate time-series and their connection with the nature of the data. In particular, the periodic events and patterns are of varying in each topic.
- the threshold and hyper-parameter tuning are very related to the data type and its representation. For instance, the contamination parameter is not clear before applying the model on data, and tuning it is not a guarantee that for new data or new patients we get the same results.

### 3.4.2 Problem-specific challenges

It may not always be possible to have ground truth labels for the data to learn based on. Sometimes there is no or incomplete information about the labeling. This comes from the fact that the size of the time-series data is very huge and data arriving in a very high rate over the time and having a human to label the data is always very challenging or even the human brain can have some mistakes which can be misleading to the anomaly detection model.

In a case of lack of true labeling, the system should have a solution of being able to do unsupervised learning on the data which by the nature of unsupervised methods, they are ignoring some of the existing true labels of the data. The possible solution could be a semi-supervised approach to have the advantage of both methods.

Imbalance or skewed data can be characterized as a hard problem to deal with in any kind of learning system and detecting anomalies on them is not an exception. When anomalies are of different types and our model didn't observe any of one specific type of abnormality and doing novelty detection can be a very challenging problem for a learning system. Plus, if a misleading label happened in the data it will worsen the issue.

### 3.4.3 Graph-specific challenges

The existence of different variations of anomalies in the context of graph domain and knowing the right criteria or choice is not always automatic for the system and has to be tuned to the type of data. For example, there are multiple versions of graph embedding, and selecting the right choice because of the high complexity of graphs is more difficult compare to the one-dimensional univariate time-series.

Natural dependency between graphs of the stream can be misleading and the assumption that graphs should be independent and identically distributed (i.i.d) is not always true for any type of time-series.

As we are searching in the graph domain the size of the search space is very huge compare to the one-dimensional univariate time-series and this transformation from a lower dimension to a higher dimension can bring us many computational and algorithmic burdens.

More importantly, processing on a stream of graph data type seems to be a great barrier that we have to find a solution such as approximate algorithms or reduce the size of data to make it practically feasible to handle such a complex network.

## Chapter 4

# Experiments on ECG

In this chapter, we have applied our methodology to study its effectiveness in performing arrhythmia's detection on the electrocardiogram signal. Section 4.1 is meant to provide all the necessary information about the Electrocardiograph signals. In particular, we describe the basic required information regarding the ECG heartbeat signal. Then, an excerpt from the related documentation of the MIT-BIH arrhythmia database is reported. In addition, some essential signal preprocessing has been made to prepare our univariate time-series for our methodology pipeline.

Details regarding our implementation for the first building block of the methodology, *vector stream from time-series*, will be described in section 4.2. Specifically, we are explaining the influence of different splitting techniques on our data and our strategy on labeling the vector stream from the database annotation.

In section 4.3, we are elaborating on the process of generating *Natural Visibility Graphs (NVG)* from our vector stream. In the last phase of our model pipeline, we express our findings on using the *Local outlier factor (LOF)* method for our graph stream. The final section of the chapter is dedicated to the final results of our methodology and the comparison between graph representational and vector representational analysis.

### 4.1 Electrocardiogram

An electrocardiogram (ECG) is a medical test to detects cardiac (heart) abnormalities which will be called *arrhythmia*. It measures the electrical activity generated by the heart as it contracts. Recorded ECGs from healthy hearts have a characteristic shape and any irregularity in the heart rhythm or damage to the heart muscle can change the electrical activity of the heart so that the shape of the ECG is changing. There are five

fiducial points in the shape of a normal ECG heartbeat that usually considered in any type of ECG signal analysis. 4.1.

- **P-wave:** this wave and it's local maximum  $P$  is the first wave for each heartbeat and related to the depolarization of the atria in each heartbeat.
- **Q-wave:** Q-wave is related to the deflection immediately before ventricular depolarization.
- **R-wave:** The global maximum point in each heartbeat is R-peak which is associated with the peak of the ventricular depolarization.
- **S-wave:** S-wave is associated with the deflection proceeding the ventricular depolarization
- **T-wave:** The last wave in each heartbeat is the repolarization of the ventricles.

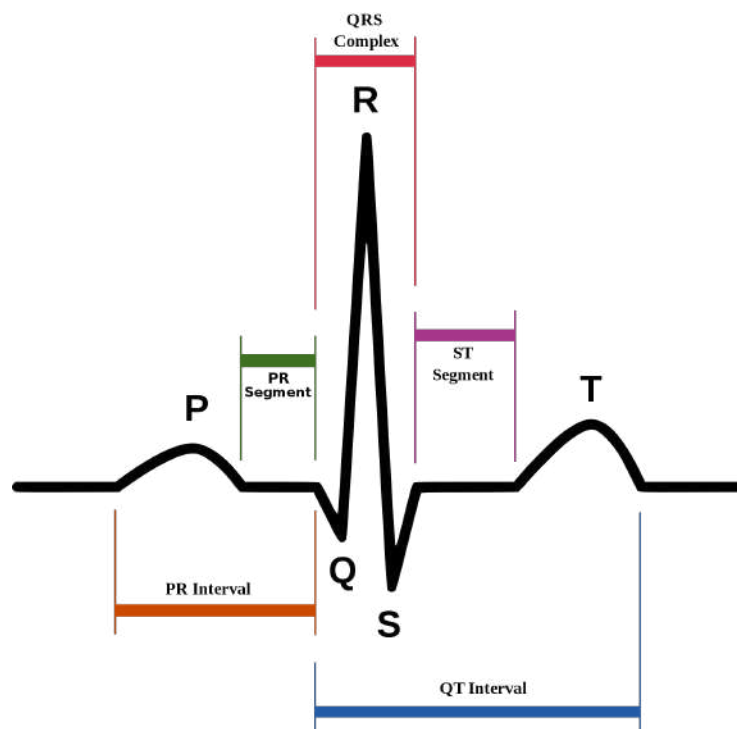


Figure 4.1. Shape of a normal ECG heartbeat (Image Created by Anthony Atkielski)

From a medical perspective, doctors analyzing an ECG signal from two main stand-points [30]:



- **Time-intervals:** They are measuring the intervals between any of the five *P*, *Q*, *R*, *S*, and *T* that provides them information on how long it takes for an electrical impulse to travel through different parts of the heart.
- **Signal strength:** A cardiologist can also assess the amount of electrical activity passing through the heart muscles by looking at the recorded voltage or the ECG signal and relate this analysis to the possible muscles of the hearts that are too large in size or are overworked.

The three major types for recording an ECG signal from a patient are:

- **Resting ECG:** The patient lies down and no movement is allowed during the recording. Each recording session usually takes 5 to 10 minutes.
- **Ambulatory ECG:** Ambulatory or Holter ECG is when you have a portable (wearable) device to monitor your heartbeat for at least 24 hours and during the monitoring, you are free to do your normal daily activities.
- **Cardiac stress test:** This type of recording is to measure your heartbeat activities while you are doing an exercise such as biking or walking on a treadmill. The required time for this recording type is about 15 to 30 minutes.

#### 4.1.1 MIT-BIH Arrhythmia Database

The MIT-BIH Arrhythmia Database [31][32] holds 48 half-hour excerpts of two-channel ambulatory ECG signals, recorded from 47 patients studied by the Beth Israel Hospital (BIH) Arrhythmia Laboratory between 1975 and 1979. The sample rate for the recordings is 360 samples per second for each channel.

Out of the set of 4000 recordings of 24-hour ambulatory ECG, 23 records were chosen at random from a mixed population of inpatients and outpatients (60% inpatients and 40% outpatients). These 23 recordings numbered from 100 to 124 (inclusive with some numbers missing) in the database. Also, 25 records of the database, were chosen from the same 4000 sets but in a way that they contain clinically significant arrhythmias that are very rare in a small random sample. These less common recordings are numbered from 200 to 234 (again inclusive with some numbers missing)

##### Inside each recorded file

Each recorded file contains a table of three columns

- **'sample #':** An identification number specifying the orders in which the samples were recorded. Notice, some samples may be missing and the sample rate is 360 samples per second per channel with 11-bit resolution over a 10 mV range.
- **Upper signal ('MLII'):** The *upper signal* is a modified limb lead II (MLII), obtained by placing the electrodes on the chest.
- **Lower signal ('V5'):** The *lower signal* is usually a modified lead V1 (occasionally V2 or V5, and in one instance V4); as for the upper signal, the electrodes are also placed on the chest.

For a better understanding of the difference between the *upper signal* and *lower signal*, you will see a 3D view to the heart activity based on the position of the electrodes on the skin in figure 4.2. Vectors reflect the angles and view of the heart activity. Notice that in our database Upper Signal (MLII) is on the Frontal Plane (Vertical Plane) and the Lower Signal (V1) is on the Horizontal Plane.

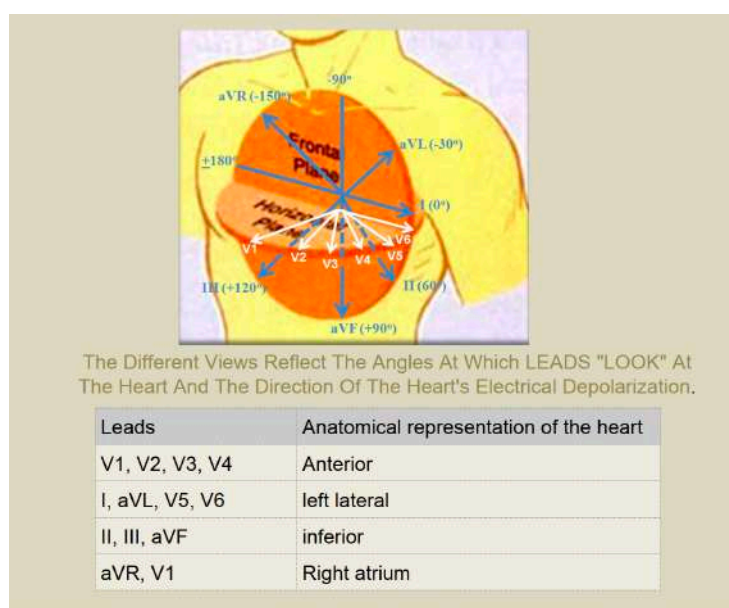


Figure 4.2. A 3D view to the heart activity monitoring based on the position of the electrodes. (from drkupe.blogspot.com)

### Decide the right channel as our univariate time-series

In the database, normal QRS complexes are usually prominent in the upper signal and the lead axis for the lower signal may be nearly orthogonal to the mean cardiac electrical

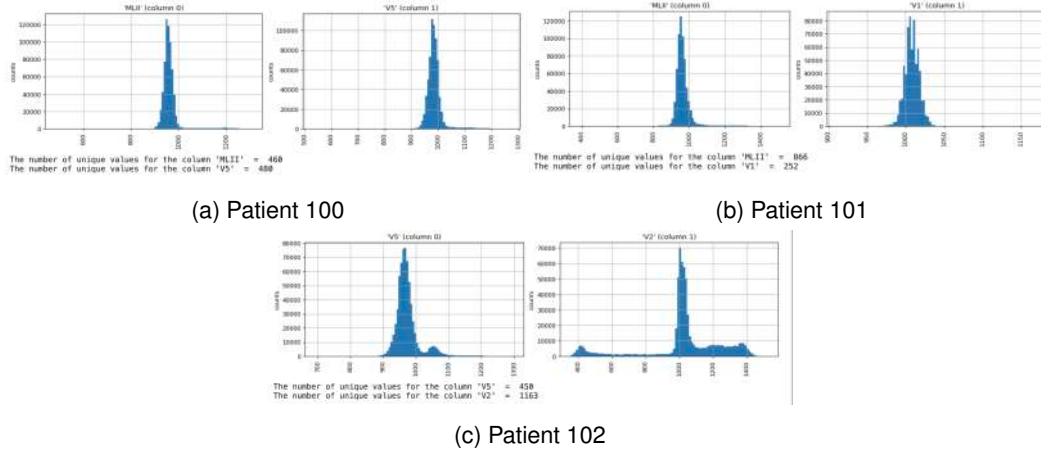


Figure 4.3. The distribution plot (histogram) of upper and lower signal for the first three patients numbered 100, 101 and 102

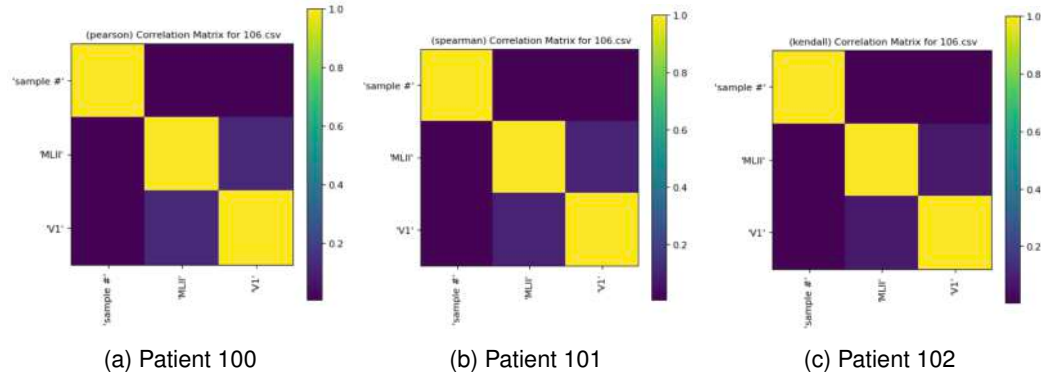


Figure 4.4. Three different correction matrix (*pearson*, *spearman* and *kendall*) for the upper and lower signal of patient number 106.

axis. Thus, we are considering my univariate time-series to be the upper signal for each recording.

To also see the distribution of the two upper and lower signals, we plot the histogram of both signal for the first three patients in figure 4.3. As expected from the database documentation the distribution for the upper signal is a better choice for our univariate time-series anomaly detector.

Also, in order to be sure about the relationship of the upper signal and lower signal for each patient, we also plot the correlation matrix between the two upper and lower signal in figure 4.4. The resulted graph is an indicator of almost zero correlation between the two channels of our recording.

The correlation between two-channel of the recordings can be presented by a clear visualization of a scatter matrix. In figure 4.5, the relationships, and patterns of change between 1000 data-points of different recorded variables for patient number 100 are presented. From this scatter matrix, we see that there might be a correlation between lower signal V5 and upper signal *MLII* according to the fact that the computed correlation coefficient is 0.706. However, the purpose of this work is not about this correlation and we just wanted to make sure that we are selecting the right signal for each patient to performing our anomaly detection model on it.

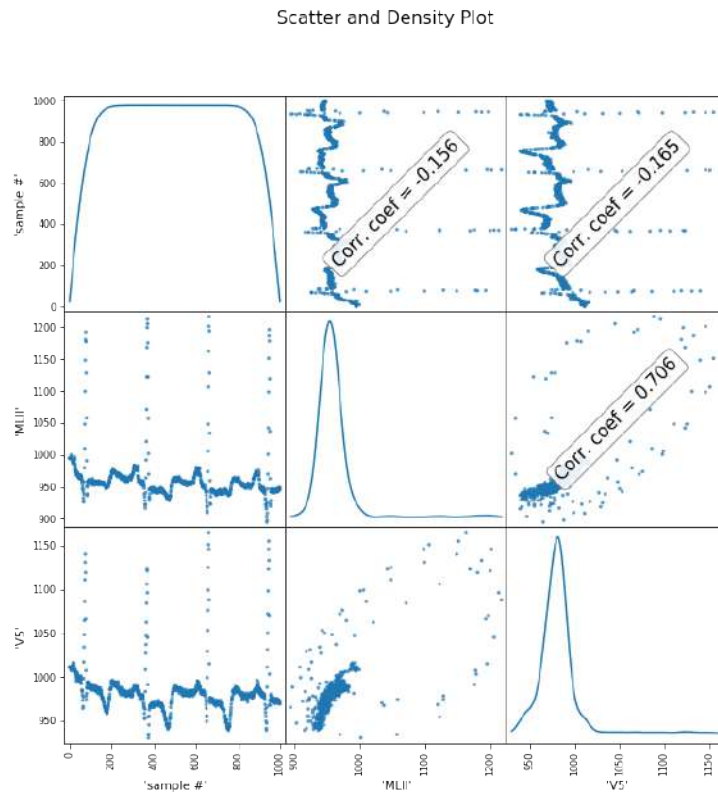


Figure 4.5. Scatter matrix for the first 1000 data-points of the patient number 100

Notice, The values for each recorded sample ranging from 0 to 2047 inclusive, with a value of 1024 corresponding to zero volts.

### Annotations

The annotations of the database initially produced by a simple *slope-sensitive QRS detector* which only marked each detected event as a normal beat. Then these initial labels were given to two cardiologists who worked on them independently. The cardiologists

added additional beat labels where the detector missed beats, deleted false detection as necessary, and changed the labels for all abnormal beats. They also added rhythm labels, signal quality labels, and comments.

The annotations were transcribed from the paper chart recordings. Once both sets of cardiologists annotations for a given record had been transcribed and verified, they were automatically compared beat-by-beat. Each discrepancy was reviewed and resolved by consensus. Once the annotations match, the results are generally appended at the R-wave peaks.

Thus, annotations for each record is any of these three kinds:

- The *true labels* for beat
- Rhythm
- Signal quality

In table 4.1, you will see the list of annotations Code and their respective description.

## 4.2 Vector stream from time-series

Based on our proposed methodology, "*GrAnd UniTs*", in order to split our non-stationary ECG signal into a stream of nearly stationary vectors, we have to perform the following three steps as previously shown in 3.1.

- **Find periodic events:** In our ECG signal we are looking for *R-peak* as the best event to specify each heartbeat.
- **Find periodic patterns:** We use a sliding window with the center of detected *R-peaks* to detect every heartbeat in the signal.
- **Create vector for each periodic pattern:** we create our vector stream from all the detected periodic patterns, heartbeats.

### 4.2.1 Find periodic events (R-peaks)

*R-peaks* is the most suitable periodic events in our ECG signals in order to identify heartbeats for two main reasons.

- *Centrality:* *R-peaks* are located approximately at the center of each heartbeat which can make it easier for us to use a sliding window to identify a heartbeat.

Code	Description
N	Normal beat
L	Left bundle branch block beat R
R	Right bundle branch block beat
A	Atrial premature beat
a	Aberrated atrial premature beat J
J	Nodal (junctional) premature beat
S	Supraventricular premature beat
V	Premature ventricular contraction
F	Fusion of ventricular and normal beat
[	Start of ventricular flutter/fibrillation
!	Ventricular flutter wave
]	End of ventricular flutter/fibrillation
e	Atrial escape beat
j	Nodal (junctional) escape beat
E	Ventricular escape beat
/	Paced beat
f	Fusion of paced and normal beat
x	Non-conducted P-wave (blocked APB)
Q	Unclassifiable beat
	Isolated QRS-like artifact

Table 4.1. List of annotations Code and their respective description that were used to label the ECG signal in MIT-BIH arrhythmia database.

- *Highest peak*: *R-peak* is the highest peak in each heartbeat. Being the global maximum point of the heartbeat is a big advantage that can help us to identify them with higher accuracy compared to other peaks of the signal.

Considering the shape of an ECG signal 4.1, there are varieties of different approaches for detecting the fiducial points of *P*, *Q*, *R*, *S* and *T* [33]. For a better understanding of the concept, we also tried to implement our simple *R-peak detector* using a simple moving average method. [34]. Our *R-peak* detector can be broken down into two steps of:

- Specify the Region Of Interest (ROI) for each R-peak.
- Determine R-peak (the maximum of the points) in the region of interest.

### Specify the Region Of Interest (ROI) for each R-peak:

Lets first define what do we mean by region of interest. ROI is the area of the signal which we want to find the *arg\_maximum* of it. In other words, R-peak is a local maximum of its ROI. To find our local maximum point, ROI, we first have to find the area in which the maximum point is falling in and then finding the point with the maximum value.

For finding this region, we first compute the simple moving average of the ECG signal then we select the areas in which the ECG signal value is greater than the simple moving average. This area is my ROI (Region Of Interest).

My simple moving average is a very simplified version of the original equation 2.4 of the MA model. we only compute the average for my sliding window and considering all the random shock generators to be zero. In our implementation, the size of the sliding window is set to be 0.75 seconds. The decision about the size of the moving window should be linked to the nature of our data generator. As a normal resting healthy heart rate for adults ranges from 60 to 100 beats per minute, having a moving window of size 0.75 for our moving average function would be a reasonable choice.

### Determine R-peak in the ROI

In total, there are three possible ways to find the R-peak in the ROI.

- Fit a curve or interpolate a curve to the points in the ROI and find the *arg\_max* of the curve.
- Find the slopes between every two adjacent points and detect the point where the slope is reversed.
- Mark points within ROI and finds the position of the highest point.

At this work, we are using the last and simplest approach which is just finding the position of the point with maximum value *arg\_max*.

In addition to the list of R-peaks, we also computed some extra properties of my ECG signal which latter we used to improve my R-peak detector.

- **R-R distances:** Compute the distance between each two adjacent R-peaks in Millisecond.
- **Beat Per minute (BPM):** Compute the Beat Per Minute (BPM) values from each heartbeat from the *R-R distances*

In figure 4.6, you will see a resulting ECG signal after detecting our R-peaks (R-peaks specified with red dots) using the simple moving average method. In addition, the Beat Per Minute line (purple line) and the simple moving average (green line) are also displayed in the same graph.

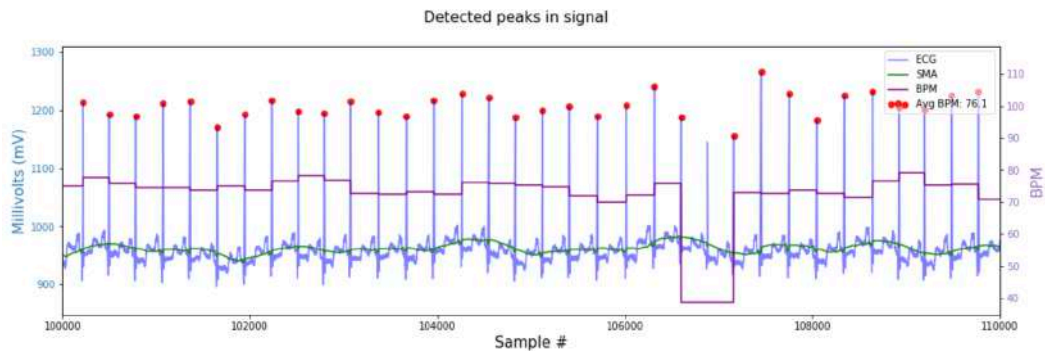


Figure 4.6. Detection of R-peaks in ECG signal using simple moving average method. Beat Per Minute (BPM) is also calculate for each heartbeat from the *R-R* distance between two adjacent R-peaks. Simple Moving Average (SMA) is also displayed on the green line.

### Improve R-peak detector

In practice, our R-peak detector is not detecting the correct R-peaks peaks as we expected because of the noise and non-stationarity of the signal. In figure 4.6, our detector missed one R-peak near the sample number 107000. To improve our detector, we simply shifting our moving average line to the top to compensate for any noise. We tried different amount of shifting of the moving average and select the one who produces more realistic changes in the BPM over the recording time. For instance, a normal heartbeat should have a beat per minute between 30 and 130. After doing this little trick and optimizing our level of shift for moving average line, we have a better detector in figure 4.7

As you can see from 4.7 plot, even though the average BPM didn't change that much, the BPM is more stable now and it varies between 60 to 110 instead of 40 to 110. However, the shift can also decrease the quality of the detector if the signal is non-stationary and only shifting of the moving average cannot be helpful.



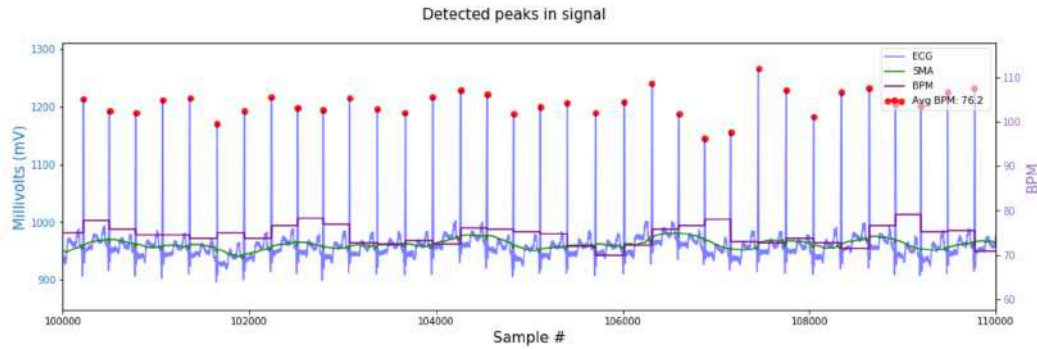


Figure 4.7. An improved detection of R-peaks in ECG signal using simple moving average method. Beat Per Minute (BPM) is also calculate for each heartbeat from the *R-R* distance between two adjacent R-peaks. Simple Moving Average (SMA) is also displayed on the green line.

#### 4.2.2 Find periodic patterns (Heartbeats)

Periodic patterns here in the context of an ECG signal are the heartbeats. Each heartbeat is a *cyclic pattern* because its duration is not fixed and may vary over time as the person's heart may beat slower or faster during sleeping hours or when we do exercise. Notice that for an ECG signal the heart of the person is our data generator and in a healthy person, the heartbeats always have the same pattern.

To detect this pattern from our signal, we simply use a sliding window having its center fix at each of the found *R-peaks* as our periodic events. From the shape of a heart-beat, we expect that an *R-peak* should be approximately at the center of the window for each heart. However, we don't know what would be a suitable size for each window. In our experiment, we had the following two options to decide about the window size.

- Fixed size window
- Variable sized window

Our choice on the variable or fixed window size is the same as deciding whether the resulting vectors of the stream can be overlapping or not. but this question can only be answered when we are considering the usage of our vectors in our methodology.

The final goal of our methodology, "*GrAnD UniTs*", is to perform anomaly detection on vector stream (Graph stream that we build from vector stream). The vector stream can be stationary if and only if our vector variables will be independent and identically distributed (i.i.d). Thus, the sliding window on our non-stationary time-series should try to capture the cycles in the signal. These cyclic patterns can be better understood

if we consider our data generator which is the person's heart. Now, it should be clear that our windows should be trying to capturing the heartbeats.

We tried both fixed and variable size window and in the next part, we comment about the advantages and disadvantages of each method.

#### Variable size window:

First, we implemented the variable size window by splitting the signal into vectors between adjacent detected R-peaks. For instance, in figure 4.8, the first four vectors for the ECG signal of the patient number 100 is plotted.

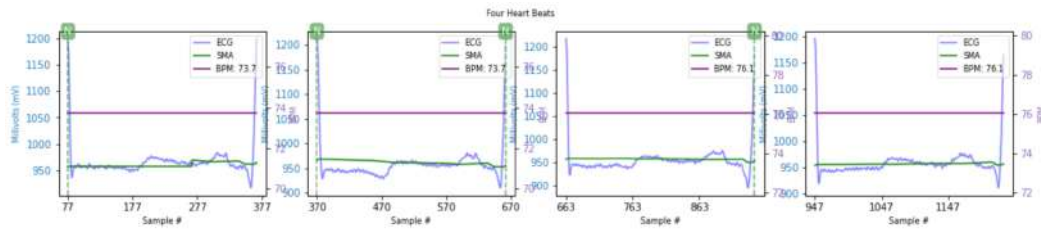


Figure 4.8. First four variable size vector (window) of ECG signal for patient number 100.

Notice, in figure 4.8, the vectors (windows) are not representing a heartbeat. In fact, each of the vectors is containing two half heartbeat of the adjacent heartbeat. This may cause a problem for two main reasons.

- **Violation of stationarity:** having vectors that are not matched with our data generator cycles can be a problem for our assumption of the stationarity of vector variable. In other words, the heart of a patient is considered as our data generator and if our vector stream is not representing a cycle or seasonality in our signal they cannot be assumed as independent and identically distributed (i.i.d) variable as the signal can have some trend in its mean over the time.
- **Misinterpretation of Annotations:** using this splitting approach, sometimes we can have multiple or even zero annotation for each vector instead of one. This is due to the fact that our R-peak detector sometimes detecting the R-peak on a single sample right or left from the real R-point point. As the annotations for each point are positioned on the R-point of each heartbeat, this single sample miscalculation of the R-peak will cause that the label for the neighboring heartbeat (the heartbeat on the right or left) can be mistakenly assigned to each other.

For the two aforementioned reasons, we decided to split the signal in a way that each vector representing a heartbeat. This can be done by considering the center of the

window to be at *R-peaks*. However, as we are going to build a graph stream from our vector stream later on in our methodology pipeline, it would be better to build the fixed size vector for each heartbeat to have a fixed size number of nodes for our graphs. To just remind us, in the last step of our pipeline, we want to compare the performance of graph-based anomaly detection versus vector-based anomaly detection, and if the vectors will not be of the same size it will not be possible for the same algorithm to handle both graph stream and vector stream. For this reason, we decided to only build a fixed size vector for each heartbeat.

#### Fixed size window:

To split our ECG signal into fixed size vectors corresponding to each heartbeat, we considered the center of the vector to be the detected R-peaks and the fixed length will be the average length of the heartbeats for each patient. Thus, our methodology will be *patient-specific*. In figure 4.9, the first four vectors (heartbeats) of our vector stream for the patient number 100 is displayed. As expected, now we have only one annotation for most of the labels. (Still, there are some vectors in our vector stream that whether we don't have any annotation for them or we have multiple annotations for them. In the next part 4.2.3, we discuss this issue and propose a practical solution for it.

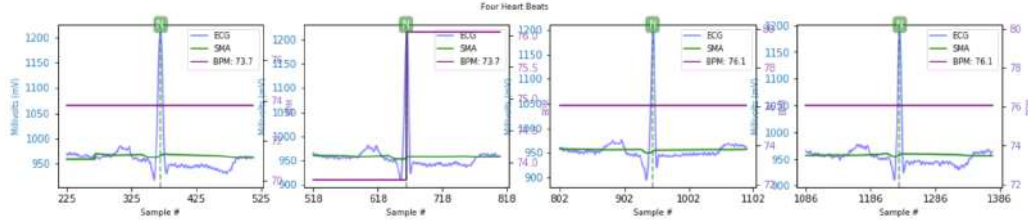


Figure 4.9. First four fixed size vector (window) of ECG signal for patient number 100.

Possible problems of using a fixed size vector can be:

- **Disconnected vectors:** When the patient is resting without any movement the actual length of our heartbeat is much higher than the average length of the heartbeat. As a result, the heartbeats will be disconnected from each other, and in the most extreme case, we even may lose some part of the *P-wave* and *T-wave*.
- **Overlapping vectors:** When the person is doing heavy exercises, his heartbeat length will be much smaller than his average heartbeat. In this case, we also consider the bordering data-points of the neighboring heartbeat as elements of

our vector. This can violate our assumption about the *stationarity* of our vector stream by including the *P-wave* and *T-wave* of the previous and next heartbeats.

Thus, both fixed size vector and variable size vectors have their own aforementioned advantages and disadvantages that we have to consider them in our methodology pipeline.

### 4.2.3 Anomaly labels from annotations:

After splitting our ECG signal into a vector stream which is addressing a sequence of heartbeats, creating the ground truth anomaly labels of +1 as normal and -1 as anomalous heartbeat from the annotations of each vector is not automatic. The issue here is that still there are some heartbeats that have multiple or zero annotations assigned to them. The problems come from two different causes.

First, for some patients in our database, our R-peak detector is not working as it is expected to work due to high noise or abrupt changes in the signal quality. In subsection 4.2.4, we try to use the most effective R-peak detector for our database of ECG to reduce this problem. However, using the best R-peak detection, it is not always the same as the ground truth and we have to consider what we have to do when multiple or zero labels are assigned to a vector.

Second, annotations are of different types, and usually for abnormal heartbeats cardiologists assigned multiple annotations of different kinds for the same heartbeat. In this case, we have to specify a strategy for labeling our vectors that have multiple annotations.

In practice, our solution for anomaly labeling of our vector stream from their respective annotation stream is presented in 4.2.3 as pseudocode. In short, we only consider each vector as a normal (anomaly label = +1) vector if all the labels assigned to it were "N" and if not we assume it is an anomalous vector (label = -1).

#### Anomaly labels from annotations

```

1  def anomaly_labels(hbs_annotations):
2      '''
3      Assign the anomaly labels based on the annotations of each HB
4      1 = Normal heartbeat ('Type' column is ['N'] or [two 'N'])
5      -1 = Anomalous heartbeat ('Type' != 'N')
6      -----
7      :param hbs_annotations: a list of dataframes which can contains
8                               ↳ annotations for Heartbeats.
9      :return true_labels: list containing the true labels of (1) or (-1)

```

```

9      '''
10     true_labels = []
11     for hb_index, hb_annotation in enumerate(hbs_annotatations):
12         if(len(hb_annotation) == 1):
13             if(hb_annotation == "N"):
14                 true_labels.append(1)
15             else:
16                 true_labels.append(-1)
17         elif(len(hb_annotation) > 1):
18             if('N' in hb_annotation):
19                 true_labels.append(1)
20             else:
21                 true_labels.append(-1)
22         elif (len(hb_annotation) == 0):
23             true_labels.append(-1)
24     return np.array(true_labels)

```

#### 4.2.4 Performance of different R-peak detectors:

As shortly mentioned in 4.2.3, our simple moving average R-peak detector is not able to perfectly identify the true R-peaks. At this work, we also tried to use some popular R-peak detection algorithms and compare their performance for our database of ECG signals.

To remind ourselves, our data-set contains 48 recordings of about 30 minutes long for different patients. We tested the effectiveness of four different R-peak detection methods by computing the three measurements of *Total Heartbeats*, *Normal heartbeat* and *Anomalous heartbeat* for each individual patient and sum all the values for the whole database.

The compared algorithms are *Pan and Tompkins*, *Two Moving Average* and *Stationary Wavelet Transform* all implemented in *py-ecg-detectors* library [33]. We also tested our developed R-peak detector to see its practicality for the whole data-set. In addition, the ground truth values for our measured values extracted from the database documentation at [1]. The resulting comparison table is presented in figure 4.10. Notice, the difference between the outcome of each method and the ground truth printed in parentheses next to the detected numbers.

From the figure 4.10, it is noticeable that the *Two Moving Average* algorithm produce closer results to our ground truth baseline. However, this may not be the case for every record in our database. We also showed this comparison between different R-peak

R-Peak Method Anomalies	Simple Moving Average	Pan and Tompkins	Two Moving Average	Stationary Wavelet Transform	Ground truth (Database)
Normal Heartbeats	73400 (1653)	67116 (7937)	74826 (227)	73612 (1441)	75053
Anomalous Heartbeats	51414 (16308)	46141 (11035)	36279 (1173)	33354 (1752)	35106
<b>Total Heartbeats</b>	124814 (14655)	113257 (3098)	111105 (946)	106966 (3193)	110159

Figure 4.10. Comparison of different R-peak detectors to find *Total Heartbeats*, *Normal heartbeat* and *Anomalous heartbeat* for the whole MIT-BIH arrhythmia database. Notice, the difference between the outcome of each method and the ground truth printed in parentheses next to the detected numbers.

methods for patient number 106 in figure 4.11. Even though for this patient *Stationary Wavelet Transform* outperforms others, the difference is negligible.

R-Peak Method Anomalies	Simple Moving Average	Pan and Tompkins	Two Moving Average	Stationary Wavelet Transform	Ground truth (Database)
Normal Heartbeats	1507	1494	1505	1505	1507
Anomalous Heartbeats	1244	754	576	517	520
<b>Total Heartbeats</b>	2751	2248	2081	2022	2027

Figure 4.11. Comparison of different R-peak detectors to find *Total Heartbeats*, *Normal heartbeat* and *Anomalous heartbeat* for patient number 106.

### 4.3 Graph stream from vector stream

In the second building block of our pipeline, by having our vector stream from the full ECG signal, we generate the *Natural Visibility Graph (NVG)* of each vector. In our visibility graph, each data point of the vector is a node, and connection between nodes specified using 3.1 inequality condition. The python implementation to build the visibility connection of our graph is presented in 4.3.

#### Natural Visibility Graph of a vector

```

1 def nvg(series, timeLine):
2     L = len(series)
3     # timeLine is the vector containing the time stamps
4     #if timeLine == None: timeLine = range(L)
5
6     # initialise output

```

```

7     all_visible = []
8
9
10    for i in range(L-1):
11        node_visible = []
12        ya = float(series[i])
13        ta = timeLine[i]
14
15        for j in range(i+1,L):
16            yb = float(series[j])
17            tb = timeLine[j]
18
19            yc = series[i+1:j]
20            tc = timeLine[i+1:j]
21
22            if all( yc[k] < (ya + (yb - ya)*(tc[k] - ta)/(tb-ta)) for k
                ↪ in range(len(yc)) ):
23                node_visible.append(tb)
24
25            if len(node_visible)>0 : all_visible.append([ta, node_visible])
26
27    return all_visible

```

Using our implementation of the visibility algorithm, we generate our visibility graph presentation of each heartbeat. In figure 4.12, we presented the third heartbeat of patient number 100 and its generated visibility graph mapped on the heartbeat for better visualization.

Notice, for better visualization of the computed *visibility graph*, we only present nodes with a degree higher than 20. Also, for better graphical presentation of the degree of each node, the circles representing the nodes are bigger if the node degree is higher. Form this *NVG*, it is clear that the nodes near the R-peaks are having a higher visibility degree compare to other nodes in the graph. In figure 4.13, a summary of information about the number of nodes and edges, the average node degree, and the list of nodes with the highest degree is presented.

## 4.4 Anomaly detection on graph stream

Finally, after having our graph stream, we use the *Local Outlier Factor (LOF)* algorithm in order to do graph-based anomaly detection on our graph representation of the ECG

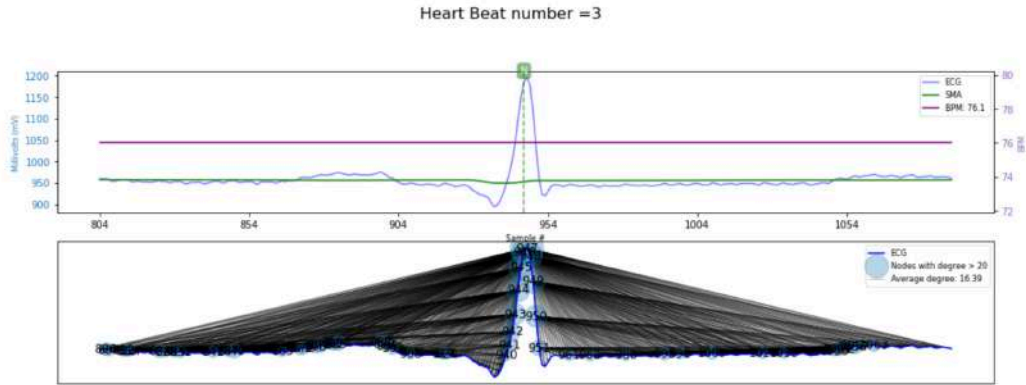


Figure 4.12. On the top a single heartbeat with its computed simple moving average (SMA) and beat per minute (BPM) is presented. Also, on the bottom the generated visibility graph mapped on top of the ECG signal.

```

-----Visibility Graph info HB = 3 -----
Name:
Type: Graph
Number of nodes: 286
Number of edges: 2344
Average degree: 16.3916

Number of connected components = 1

Top five nodes with the highest degree (In and Out degree)
Node with sample number: 947 => Degree: 225
Node with sample number: 946 => Degree: 125
Node with sample number: 948 => Degree: 125
Node with sample number: 949 => Degree: 124
Node with sample number: 945 => Degree: 123

```

Figure 4.13. General information about the generated Visibility graph that presented in 4.12.

signal. Using *LOF* method, we provide *Graph Edit Distance (GED)* and *NetSimile* metric to measure the similarity between each pair of graphs.

#### 4.4.1 Tools

At this project, we implemented our experiments using the *Python v3.8* programming language. In recent years, *Python* language has become popular and became the most widely used language for data science and machine learning purposes. A brief description of some of the software tools used for the project is as follows.

- **Scikit-learn [35]:** *Scikit-learn* is a free software machine learning library for the



Python programming language. Through this thesis, we used its direct implementation of the *Local Outlier Factor (LOF)* [13]

- **GMatch4py** [36]: GMatch4py [36] is a library dedicated to multiple graph matching methods. At this work, we used their implementation of the algorithm *Hausdorff Edit Distance* [14]. This algorithm is an approximate implementation of the original *GED* algorithm.
- **Networkx** [37]: *NetworkX* is a python library for studying graphs and networks and we used this library to building our visibility graphs and visualize them in plots. We also use the implementation of the library for computing the exact *Graph Edit Distance (GED)* between graphs.
- **Pandas** [38]: The *Pandas* is an open-source data analysis and manipulation tool, built on top of the Python programming language to better management of the data processing.
- **Numpy** [39] *Numpy* is a library for scientific computing in Python. It provides a high-performance multidimensional array object and tools for working with these arrays.

Notice, the *scikit-learn* [35] implementation of *LOF* method can be applied in two possible ways on our graph stream:

- **Outlier mode**: In this mode, the training data can contain anomalies and the model is identifying the deviant observations.
- **Novelty mode**: using this way of anomaly detection, the training data should not be polluted by anomalous samples and we are interested in detecting whether a new coming observation is a novelty (anomaly) or not.

For this thesis, we applied it in the Outlier detection format to see the performance of our methodology on our data set of ECG signals.

#### 4.4.2 Computational challenges

Here at this part, we address the computational challenges and difficulties that we faced while working with the time-series data and its constructed graph stream.

##### Graph Edit Distance (GED) and Hausdorff Edit Distance (HED)

In the first attempt, we tried to test the performance of exact *Graph Edit Distance (GED)* on our graph stream using the exact implementation of the *Networkx* library. However,

due to the huge size of our graphs (each graph has about 284 nodes and more than 2000 edges.), computing the exact GED between every two graphs took more than hours. Thus, we tried to use the approximate algorithms of the GED. The only possible algorithm that makes it feasible for us to compute the *Distance matrix (similarity matrix)* was *Hausdorff Edit Distance* [14]. We used its implementation on python by Fize [36].

### Downsample the original ECG signal

However, even an approximate graph edit distance was still computationally very slow for our purpose regarding our huge size graphs with roughly 284 nodes and 2300 edges. Our solution to address this issue was to downsample our original ECG signal. There were multiple ways to downsample a signal and we have chosen the easiest method which is sampling on out of every  $M$  observation. We first start by considering  $M = 2$  and it was still taking days to compute the *similarity distance matrix* for all the graphs of heartbeats for every single patient (we tried it for patient number 106 which had about 2086 found heartbeats.). Finally, we decided to down-sampling our original ECG signal with the step of  $M = 8$  makes it possible to compute the distance matrix between all the graphs in about 2.65 hours for every single patient.

We could have downsampled the signal using a step greater than 8 but the quality of the signal would reduce significantly. In a try, we do that and even our R-peak detector would not be able to efficiently detect the *R-peaks* in the second building block of our methodology pipeline.

### NetSimile

Although we increase the speed of our methodology both by applying HED distance metrics and downsample the signal, the method was still slow. So, we decided to use *NetSimile* graph distance similarity matrix for the following two major reasons:

- *NetSimile* is a similarity distance metric for graphs that can be suitable for huge size graphs. The focus of the model is to be size-independent.
- In the second step of the *NetSimile* algorithm, we aggregate on the local or neighboring features to a fixed size vector called *Signature vector*. The fact that this vector always has 35 elements inside it, can be very beneficial for us. It can be considered as some kind of embedding the graph to a vector.

- The fixed size signature vector can be considered as dimensionality reduction for our huge size graphs of roughly 285 nodes.

Using the new graph similarity measurement of *NetSimile* not only reduced the time but also increased the performance of the algorithm which can be also an indicator that our approximate GED metric was not as effective for huge size graphs.

Notice, In our implementation we didn't use the *NetSimile* implementation they way they implemented. In fact, we only computed the *feature matrix* and *feature aggregation* (Step 1 and 2 of the algorithm look chapter 3) with the goal of creating the *Signature vector* for each graph. Then, we calculate the pairwise *Euclidean distance* between signature vectors to build the *precomputed* distance matrix for our *LOF* algorithm.

#### **Precomputed distance matrix for LOF**

One of the advantages of selecting the *LOF* algorithm as our anomaly detection method was its ability to work with the precomputed distance matrix. We only compute the distance matrix once with any distance metric of our choice and then we can feed this similarity distance matrix to our algorithm to detect anomalies. This simple trick prevents us from calculating the same matrix over and over again in the hyper-parameter tuning phase such as k-fold cross-validation.

#### **4.4.3 Thresholding and Hyperparameter tuning**

Like any other learning model, the *LOF* model has some hyper-parameters that require to be tuned with respect to our database. The hyperparameters of this algorithm are contamination parameter  $c$  and the number of neighbors  $k$ . Contamination parameter  $c$  specifies the portion of data that we expect to be anomalies in the percentage. Actually, the contamination parameter is not a hyperparameter and is a *threshold* on our *outlier score* but it can also be considered as a parameter of a model to be tuned. Hence, we tune it to have the optimal value similar to a hyperparameter. The number of neighbors  $k$  is selecting the number of neighbors to consider in the calculation of *local density of an object* in the *LOF* algorithm which is represented in 3.3.1.

##### **(Grid search) and (k-fold Cross-Validation)**

In order to tune our parameters, we perform the well-known *k-fold Cross-Validation* and its direct implementation of the algorithm from the *scikit-learn* library [35] which is called *StratifiedKfold*.

Pay attention, due to the nature of the anomaly detection problem usually, the data set is imbalanced so the implementation that we used is designed for this case called *Stratified*. In a nutshell, we split data into folds such that to be sure that each fold has the same proportion of the two normal and abnormal classes.

Moreover, in our implementation, we used the `GridSearchCV` from the *scikit-learn* library [35] which is simply searching in a grid made of all the combinations of different parameters. Notice, as we have two parameters to tune so our grid would be of any combination of these two values.

The result of our *Stratified Cross Validation* algorithm for 100 values of number of neighbors  $2 < k < 1000$  and 100 values of contamination  $0.000001 < c < 0.5$  is the two optimal values of  $k = 923$  and  $c = 0.2165$ . The true percentage of the anomalies for this patient (patient number 106) is 0.27 which is somehow close to the found optimal contamination parameter.

#### Visualized parameter tuning using the F1 score.

We tried to visualize the effect of different hyper-parameters on the *F1 score* of our model (the reason behind our choice of *F1 score* is explained in chapter 3). Our first hyper-parameters to consider is the number of neighbors  $k$  and we try values between 2 to 1000 when we have 2086 number of heartbeat for the patient number 106. In figure 4.14, for all the values of  $k > 254$  the resulted *F1 score* is about 0.88. We know that the smaller number of neighbors can be preferable computational-wise so we consider the 254 as the optimal choice.

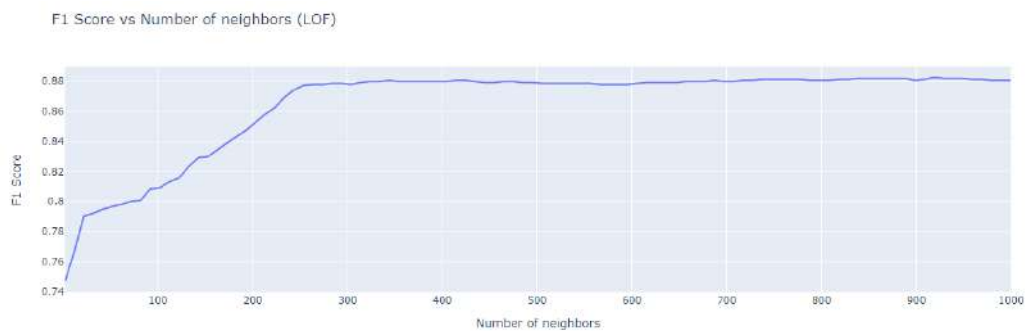


Figure 4.14. F1 score vs Number of neighbors ( $k$ )

Also, our second hyperparameter or more specifically our threshold parameter on outlier score for tuning is contamination  $c$  which again we test the effect of different

values for this parameter on the outcome of our model in figure 4.15. The best value for contamination percentage based on our database of the graphs for patient number 106 is around 0.27 with result in *F1 score* of about 0.88

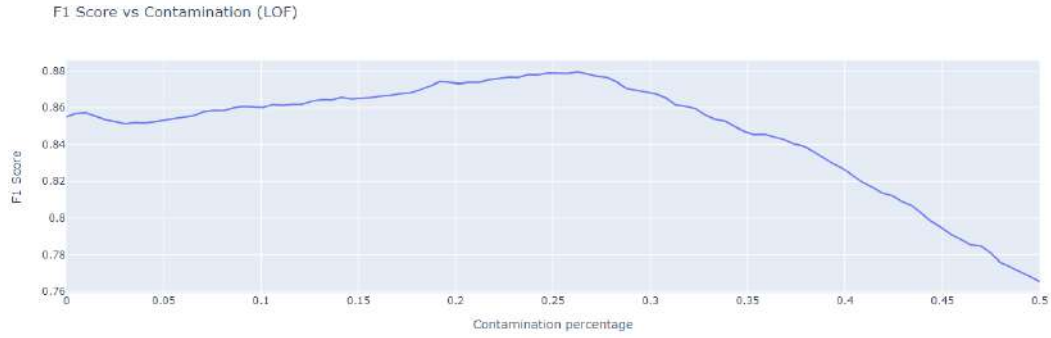


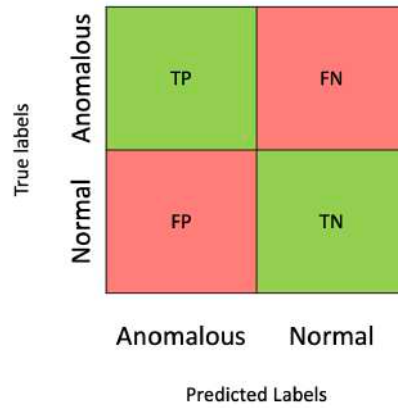
Figure 4.15. F1 score vs Contamination (c)

#### 4.4.4 Performance evaluation:

At this part, we discuss the resulted performance of our methodology, "GrAnD UniTs", applied on the ECG signal of the patient number 106. The choice of the patient number 106 out of 48 total number of patients is due to the fact that the proportions of anomalies and normal data points are somehow fair compare to the other patients in our data set. Later on, in the next part, we try our methodology for all the patients in the data set. There are multiple different evaluation metrics that we go through each of them.

**Confusion Matrix:** A *Confusion Matrix* is a table that is often used to describe the performance of a classification model on a data set. At this matrix, we will compare the true labels (true classes of the data) and the predicted label of the data (the output of the model). In the problem of anomaly detection which can be considered as a binary classifier, a confusion matrix is a  $2 \times 2$  matrix as it is shown in figure 4.16. Notice, the two green block of the matrix is when our prediction hit the true values and the red blocks are the cells that we missed the real values, and our model incorrectly predicts the other class.

As mentioned in chapter 3, the values  $TN = \text{True Negative}$ ,  $TP = \text{True Positive}$ ,  $FP = \text{False Positive}$  and  $FN = \text{False Negative}$  will be used to analyse the performance of a binary classifier algorithm. These values clearly observed at this matrix.



A confusion matrix comparing True Labels (Anomalous, Normal) against Predicted Labels (Anomalous, Normal). The matrix is a 2x2 grid. The top row (True Labels: Anomalous) has a green cell (TP) for predicted Anomalous and a red cell (FN) for predicted Normal. The bottom row (True Labels: Normal) has a red cell (FP) for predicted Anomalous and a green cell (TN) for predicted Normal. The columns are labeled 'Anomalous' and 'Normal' at the bottom.

True labels	Anomalous	TP	FN
	Normal	FP	TN
		Anomalous	Normal
		Predicted Labels	

Figure 4.16. Confusion matrix to compare the performance of anomaly detection model.

To visualize the performance of our *graph-based* anomaly detection methodology, we evaluated the prediction of our model against the ground truth binary labels (labels by Cardiologists in the database.) for the patient number 106. This evaluation metrics presented as a confusion matrix in figure 4.17. On the top of the figure 4.17, we also printed the achieved *accuracy* = 0.8188, *precision* = 0.8871, *recall* = 0.8677, *F1 score* = 0.8773 and the *ROC AUC* = 0.8570. (we will discuss on the *ROC curve* and the area under curve more at the next part when we compare our methodology with the vector-based implementation.)

## 4.5 Graph-based anomaly detection vs. Vector-based anomaly detection

Decisions on using the Local Outlier Factor (LOF) method and *NetSimile* graph similarity distance for doing anomaly detection on our graph stream are mostly based on the practicability of these methods for our database. In addition, using *LOF* and *NetSimile* make it possible for us to compare the performance of our *Graph-based* anomaly detection methodology with *Vector-based* anomaly detection approach which have been recently proposed in the literature [1].

Here at this thesis, we compare our implemented "*GrAnd UniTs*" methodology as a *Graph-based* method with the *Vector-based* format of the same pipeline which can be seen as an ablation study to see the effectiveness of the graph representation of the time-series data. In figure 4.18, the specific step of the methodology that we are studying

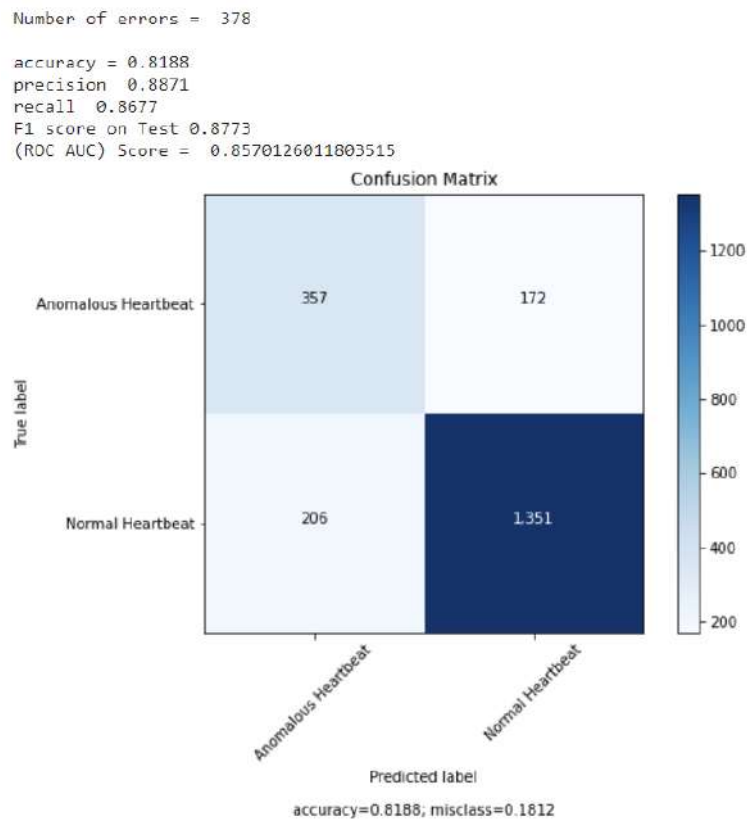


Figure 4.17. Confusion matrix to compare the performance of our *Graph-based Anomaly Detector*.

its effects on our final performance of the model is marked by the red rectangle.

Thus, *vector-based* method performs the following steps in its pipeline:

- Build a vector stream from the ECG signal
- Compute the distance matrix of *Vectors* using the *Euclidean* distance metric
- Detect anomalies using *LOF* and the computed *distance matrix* algorithm

However, our *Graph-base* method, executes the following steps:

- Build a vector stream from the ECG signal
- **Generate graph stream from the vector stream**
- **Compute the signature vectors of the graphs**
- Compute the distance matrix of **Signature Vectors** using the *Euclidean* distance metric

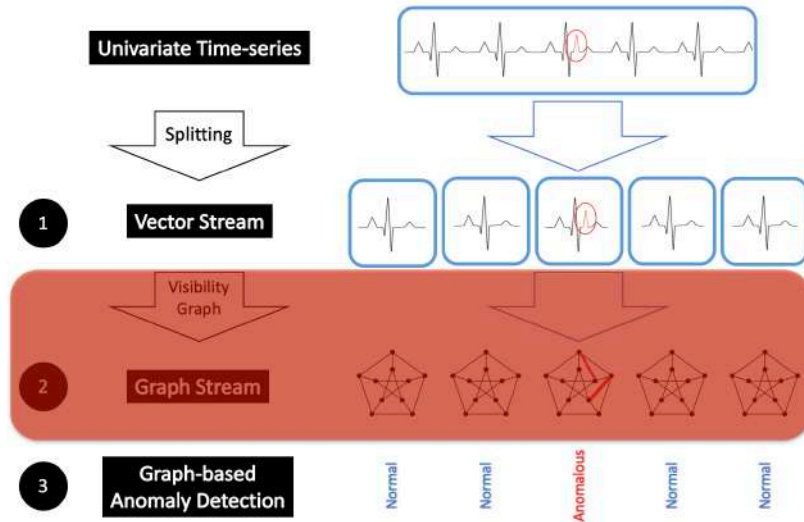


Figure 4.18. *Vector-based* anomaly detection pipeline schematic.

- Detect anomalies using *LOF* and the computed *distance matrix* algorithm

The focus of this ablation study is the influence of the intermediate graph representation of the data.

Comparison of the two *Vector-based* and *Graph-based* method would be two kinds

- Comparison on a single patient
- Comparison on the whole data set

#### 4.5.1 Comparison on a single patient

We compare the two method using three types of metrics

- *Quantitative evaluation*
- *ROC curve*
- *Precision Recall curve*

#### Quantitative evaluation

In figure 4.17, we presented the performance of the *graph-based* model for the patient number 106. After applying *vector-based* method on the same patient, the resulting confusion matrix is presented in 4.19



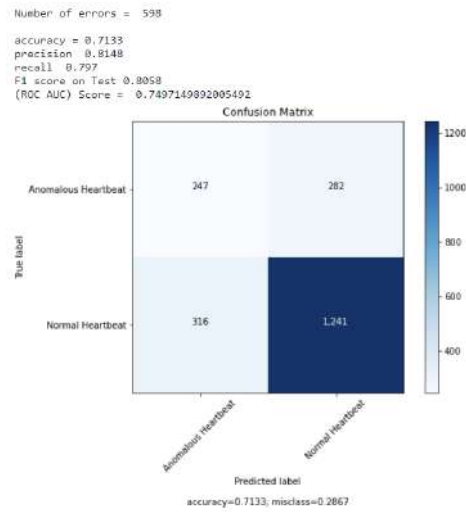


Figure 4.19. Confusion matrix to compare the performance of our *Vector-based* model.

Notice, also at the top of the figure 4.19, you will see the multiple computed evaluation metrics resulted from the output of the *vector-based* method . For better comparison of results of the two *vector-based* and *graph-based* models, we collected all the values in table 4.20.

	TP	TN	FP	FN	Accuracy	Precision	Recall	F1 Score	ROC_AUC	Errors
Graph-based	357	1351	206	172	0.8188	0.8871	0.8677	0.8773	0.8570	378
Vector-based	247	1241	316	281	0.7133	0.8148	0.797	0.8058	0.7497	598

Figure 4.20. Resulted quantitative evaluation metrics for both *vector-based* and *graph-based* models.

Figure 4.20 shows that overall for all the metrics *Graph-based* method outperforms the *Vector-based* model. For instance, *F1 score* for *graph-based* model is 0.87 and for the *vector-based* model is 0.80 which shows the slightly lower performance. Moreover, the area under the ROC curve for both *vector-based* and *graph-based* are 0.74 and 0.85 respectively which again indicates the better performance of *graph-based model*.

### ROC curve

Another important quantitative and visual metric to compare two anomaly detection methods is the *Receiver Operating Characteristic (ROC) curve*. ROC curve and the area under the curve (AUC) is based on the two computed values of *False Positive Rate* and *True Positive Rate* that both can be computed from the confusion matrix.

In figure 4.21, the resulting *ROC curve* for both *graph-based* and *vector-based* model presented.

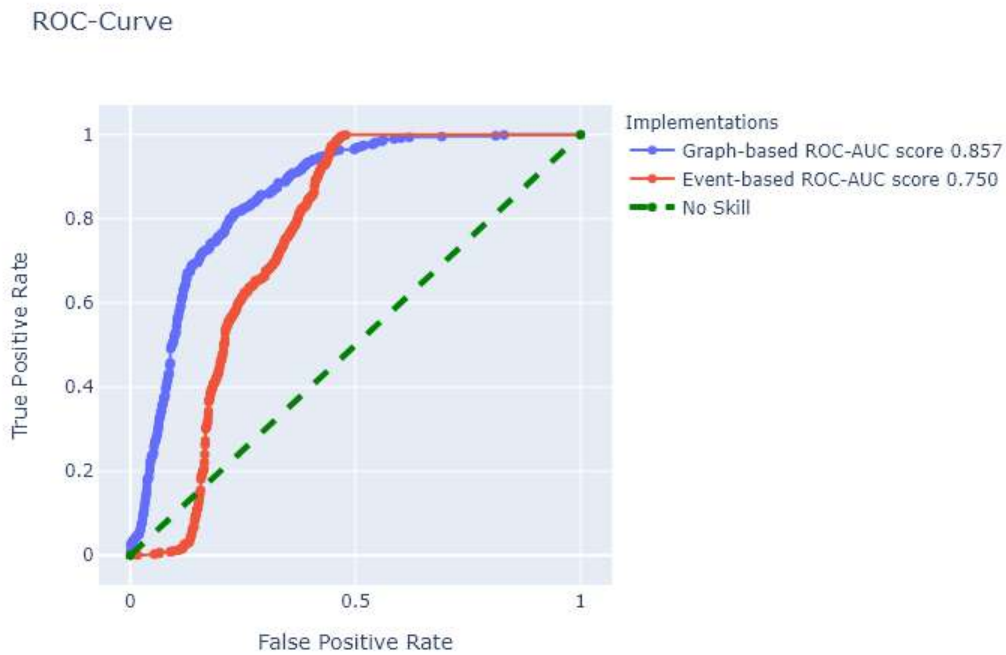


Figure 4.21. Resulted *ROC curve* for both *vector-based* and *graph-based* models.

Comparison of the two *ROC curve* shows that the *graph-based* model is outperforming *vector-based* method. This increase of performance in our fair comparison of the two models indicates the effectiveness of graph representation of the data to captures the local dependency network of heartbeats vectors. In other words, processing of the relativity or connections between data points of the heartbeat vector can be more effective than processing the quantitative in vector-based approach.

In this plot, the blue line representing the *graph-based* model is on top of the red line and the area under it is clearly showing the higher value. on the legend of the plot you will see the computed *ROC-AUC scores* of 0.85 and 0.75 for both *graph-based* and

*vector-based* respectively.

### precision-recall curve

Intuitively, precision measures how well our algorithm identifies only anomalies, and recall measures how well we identify all anomalies both of these values are normalized percentage between zero and one. You will see a visualized definition of precision and recall factors in figure 4.22.

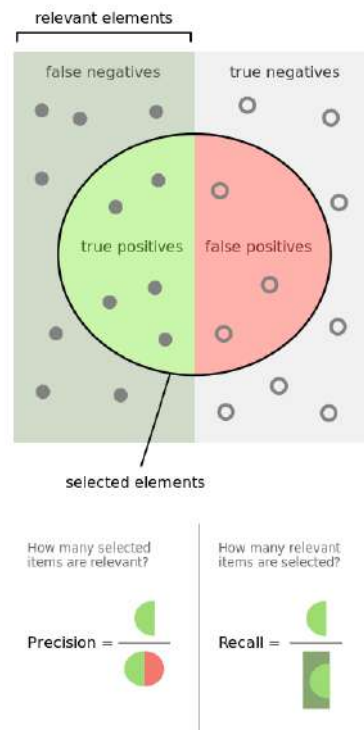


Figure 4.22. Visualization of *Precision* and *Recall* metrics (From Wikipedia)

In figure 4.23, we presented *Precision-Recall curve* for both *Graph-based anomaly detection* and *Vector-based anomaly detection*.

Even though the *graph-based* model shows a better performance compared to *vector-based* method on the figure 4.23, we see a slight decrease in the average precision of the *graph-based* model compared to *vector-based* method.

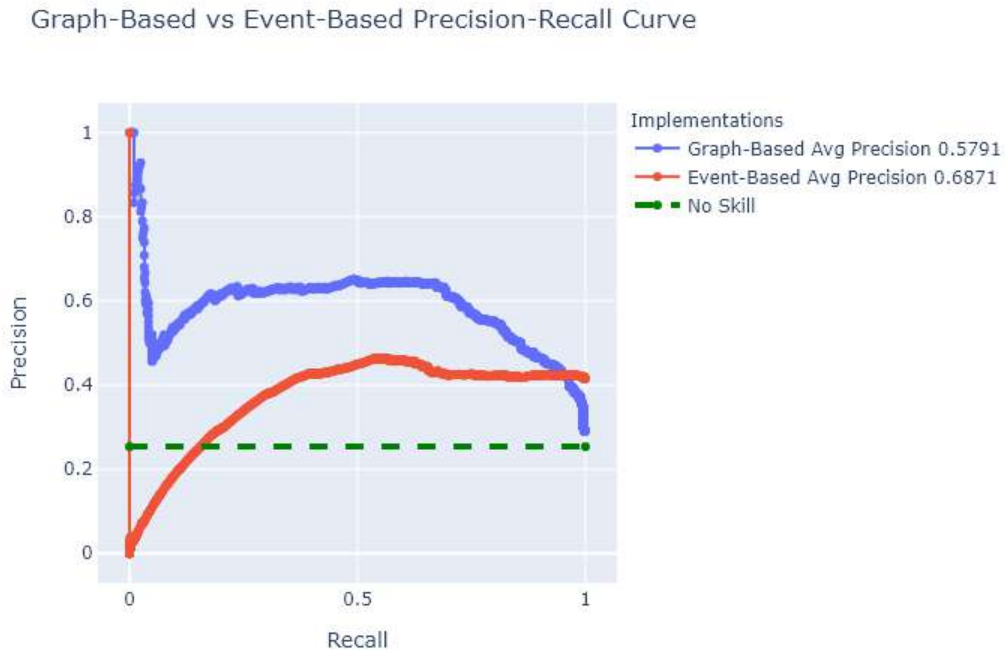


Figure 4.23. The resulting *Precision-Recall* curve for both *graph-based* and *vector-based* methods

#### 4.5.2 Comparison on the whole data set

After observing the *graph-based* model outperforming *vector-based* method on the provided ECG signal of patient number 106. We want to investigate the overall performance of our proposed methodology, "*GrAnd UniTs*", for all the patients. Remember, our methodology is a patient-specific method but we want to measure the performance of our model by detecting heartbeat arrhythmias on every single patient at a time then evaluate the results on the whole dataset.

We applying both *Graph-based* anomaly detection and *Vector-based* anomaly detection for all the 48 patients of the *MIT-BIH arrhythmia database*. The table 4.24 shows an example of the *ROC AUC*, *F1 score* results for 21 patients.

The columns *num\_normal\_hbs* and *num\_anomalous\_hbs* are representing the real number of normal and anomalous heartbeats for each patient. Also, the two columns of *gra\_fbeta\_score* and *vec\_fbeta\_score* are showing the resulted *F1 score* for each *graph-based* and *vector-based* methods respectively. The last two columns, *vec\_roc\_auc* and *gra\_roc\_auc*, are also indicating the computed *ROC AUC* values for each of the two

	p_id	num_normal_hbs	num_anomalous_hbs	vec_fbeta_score	gra_fbeta_score	vec_roc_auc	gra_roc_auc
0	100	2235	34	0.8491	0.843485	0.882406	0.712423
1	101	1859	6	0.8453	0.844720	0.958580	0.857540
3	103	2080	2	0.8444	0.844444	0.891587	0.887981
5	105	2533	52	0.8538	0.851131	0.892314	0.911871
8	108	1773	30	0.8488	0.844934	0.820549	0.706242
11	112	2534	2	0.8442	0.844242	0.935872	0.875493
12	113	1786	6	0.8455	0.845507	0.979097	0.968458
13	114	1840	66	0.8548	0.844321	0.906069	0.667531
14	115	1950	0	0.8438	0.843759	0.001000	0.001000
15	116	2298	112	0.8671	0.867143	0.979493	0.990096
16	117	1532	1	0.8442	0.844210	0.992820	0.990862
19	121	1859	2	0.8436	0.844265	0.758741	0.906939
20	122	2473	0	0.8439	0.843852	0.001000	0.001000
21	123	1518	2	0.8443	0.844309	0.973979	0.996377
25	202	2081	76	0.8564	0.855267	0.899068	0.873163
27	205	2569	74	0.8577	0.857270	0.997370	0.980700
35	215	3192	165	0.8685	0.866359	0.997839	0.983307
37	219	2103	99	0.8631	0.863073	0.943179	0.940359
38	220	1951	94	0.8670	0.865854	0.990212	0.959606
43	230	2252	1	0.8439	0.843943	0.999556	0.991119
47	234	2697	50	0.8528	0.849426	0.937360	0.868721

Figure 4.24. The resulting *ROC AUC* and *F1 score* for multiple patients using both the *graph-based* and *vector-based* method

methods.

To compare the resulting *ROC AUC* of the two models, we plot their histogram to analyze their distribution in figure 4.25. Based on this histogram both *Vector-based* and *Graph-based* methods are having almost the same distribution and their means and median are very close to each other. the mean values 0.844 and 0.812 resulted for both *vector-based* and *graph-based* methods respectively. Also the medians 0.937 and 0.906 for both methods showing the similarity of the results of the two approaches.

According to the table 4.24, the two patients with the numbers 122 and 115 shows the resulting *ROC AUC* of 0. This unusual value is due to the fact that for these patients, we only have one class of type *normal* and no *anomalies* in the recorded data. As all

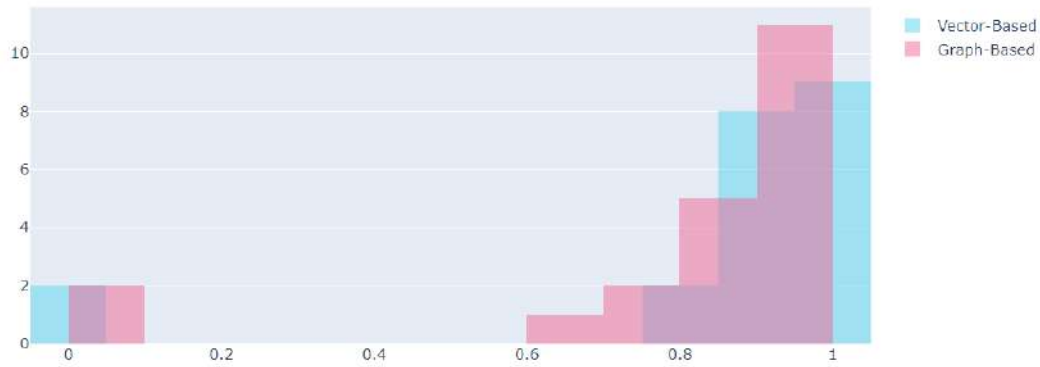


Figure 4.25. The histogram of the resulting *ROC AUC* values for 21 patients using both *graph-based* and *vector-based* method.

the true labels are from one single class, in order to avoid the calculation deadlock in our implementation, we manually set the resulting measured values of *ROC AUC* to zero. In other words, the resulting zero values in the bottom left part of the figure 4.25 should be neglected as they are manually fixed to avoid the error (exception) in the `roc_auc_score` function in the implementation of the **Scikit-learn** library.

Notice, here we only presenting a selected number of patients who are having more that 1000 normal heartbeats `num_normal_hbs > 1000` and having less that 200 anomalous heartbeats `num_anomalous_hbs < 200` in their roughly 30 minutes ECG signal.

Also in figure 4.26, we have even more similar histogram for the resulted *F1 score*. Mean values are 0.851 and 0.850 and the medians are 0.848 and 0.844 for both *vector-based* and *graph-based* models.

From the histogram plots, we cannot distinguish any significant performance improvement for any of the models. However, to make sure that the resulting values of the two models are from the same distribution or not we perform a *Wilcoxon signed-rank test* on our resulted *ROC AUC* evaluation measurement from both *Graph-based* and *Vector-based* model. In short, a *Wilcoxon* test will examine the null hypothesis that both resulted lists of values are from the same distribution. The output of this test for the resulting *ROC AUC* values of the two methods was  $p\text{-value} = 0.029$  which is less than 0.05 meaning that the two resulted data are significantly different.

Also, the resulted  $p\text{-value}$  for the *wilcoxon* test on the *F1 score* of both algorithms is 0.019 which is still very low and meaning that the distribution of the two list for both *vector-based* and *graph-based* are significantly different.

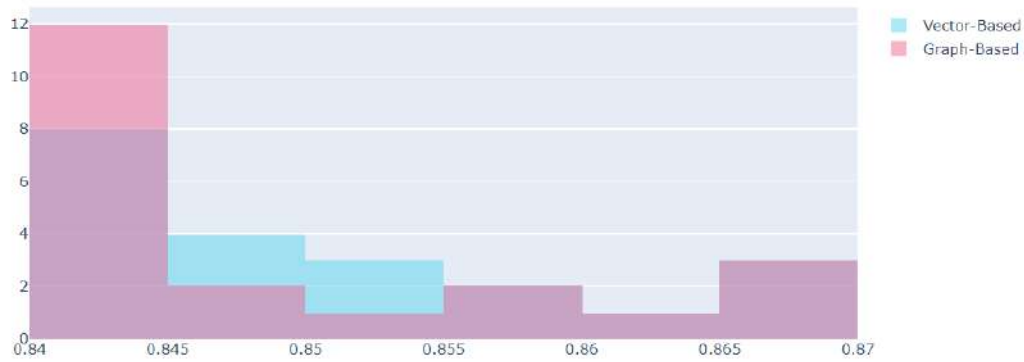


Figure 4.26. The histogram of the resulting *ROC AUC* values for 21 patients using both *graph-based* and *vector-based* method.

In figures 4.27a and 4.27b, the resulted *ROC AUC* and *F1 score* quantitative evaluation metrics for both *graph-based* and *vector-based* methods are plotted with respect to each other in a scatter plot. As indicated in the plots we can assume that there is linear relation between the two computed values.

In other words, the correlation plots below show that vector-based and graph-based algorithms have similar performance, however sometimes the *vector-based* outperforms the *graph-based* algorithm.

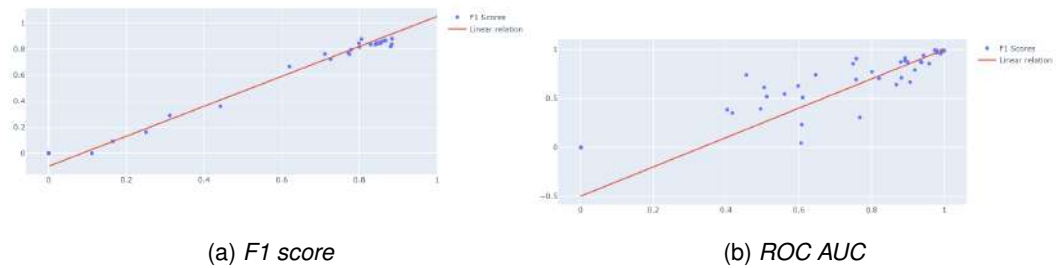


Figure 4.27. The correlation plot for the resulted *F1 score* and *ROC AUC* values computed using both *vector-based* and *graph-based* approaches





## Chapter 5

# Conclusion

In this thesis, we tried to use the potential power of graphs as an abstract structure to capture the local inter-dependency network inside a univariate time-series to identify anomalous periodic patterns. To this end, we developed our novel method, "*GrAnD UniTs*". Our method is different from other standard graph-based approaches in the sense that we try to extract interdependencies along the temporal dimension, whereas most other widely used methods extract interdependencies between the different channels of a multivariate signal.

Our model consists of the three main steps of (1) vector stream creation, (2) visibility graph generation, and (3) applying distance-based anomaly detection on the generated graph. In step one, we first detect the periodic events in the time series data, and then using a slide window technique divides the time series into a stream of vectors. In the next step, we transform the generated vector stream into the graph domain using a visibility graph. Finally, we used the local outlier factor (LOF) method to detect anomaly periods. LOF method requires using a distance metric and here we tried a suite of different methods with different computational complexities.

We implemented different phases of the model pipeline and applied our method for arrhythmia detection on an electrocardiogram (ECG) signal. The distance metric that we tried in the LOF step for arrhythmia detection was the graph edit distance which resulted in approximately 72 days of computation time per 30 minutes of ECG recording. Therefore, we tried reducing the computation time by using Hausdorff Edit Distance (HED) to calculate the graph distances. As a result of this change, the computation time was reduced to 2.6 hours per 30 minutes of ECG. However, the approximate HED algorithm resulted in poor performance for the anomaly detection algorithm. Finally, we tried the NetSimile method as a size-independent graph similarity metric which in-

creased performance and reduced the computational complexity to 20 minutes. This time improvement was achieved via feature aggregation that is happening in the Net-Simile algorithm.

In the final chapter of the thesis, we performed an ablation study for 20 patients to evaluate the effectiveness of the developed graph-based representation against a vector-based approach Oehmcke et al.. Our developed method shows average performance comparable to the vector-based approach. However, for cases where the number of anomalies in the ECG signal is high, the vector-based approach outperforms our method.

Improvements to the current algorithm can be done by using an attributed graph instead of a plain graph to capture the quantitative information as well as the dependency network in the data. Furthermore, we can employ reconstruction-based methods by using recurrent neural networks and convolutional neural networks for supervised anomaly detection on graph stream. [40, 41]

# Bibliography

- [1] Stefan Oehmcke, Oliver Zielinski, and Oliver Kramer. Event detection in marine time series data. In Steffen Hölldobler, Rafael Peñaloza, and Sebastian Rudolph, editors, *KI 2015: Advances in Artificial Intelligence*, pages 279–286, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24489-1.
- [2] C. Pozrikidis. *An Introduction to Grids, Graphs, and Networks*. OUP USA, 2014. ISBN 9780199996728. URL <https://books.google.ch/books?id=ibucAgAAQBAJ>.
- [3] Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuño. From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, 105(13):4972–4975, 2008. ISSN 0027-8424. doi: 10.1073/pnas.0709247105. URL <https://www.pnas.org/content/105/13/4972>.
- [4] H. Ji, T. Xu, W. Wu, and J. Wang. Visibility graph analysis on eeg signal. In *2016 9th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pages 1557–1561, 2016.
- [5] Minzhang Zheng, Sergii Domanskyi, Carlo Piermarocchi, and George I. Mias. Visibility graph based community detection for biological time series. *bioRxiv*, 2020. doi: 10.1101/2020.03.02.973263. URL <https://www.biorxiv.org/content/early/2020/03/03/2020.03.02.973263>.
- [6] Charu C. Aggarwal. *Outlier Analysis*. Springer International Publishing, Springer Nature Switzerland AG, 2017. ISBN 978-3-319-47578-3.
- [7] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215 – 249, 2014. ISSN 0165-1684. doi: <https://doi.org/10.1016/j.sigpro.2013.12.026>. URL <http://www.sciencedirect.com/science/article/pii/S016516841300515X>.
- [8] Vassilis Kalofolias. How to learn a graph from smooth signals. *arXiv e-prints*, art. arXiv:1601.02513, January 2016.
- [9] Xiaowen Dong, Dorina Thanou, Michael Rabbat, and Pascal Frossard. Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36(3):44–63, 2019.
- [10] Dongmian Zou and Gilad Lerman. Graph generation via scattering, 2019. URL <https://openreview.net/forum?id=HyxSBh09t7>.

- [11] Ljubiša Stanković, Miloš Daković, and Ervin Sejdić. *Introduction to Graph Signal Processing*, pages 3–108. Springer International Publishing, Cham, 2019. ISBN 978-3-030-03574-7. doi: 10.1007/978-3-030-03574-7\_1. URL [https://doi.org/10.1007/978-3-030-03574-7\\_1](https://doi.org/10.1007/978-3-030-03574-7_1).
- [12] Mutua Stephen, Changgui Gu, and Huijie Yang. Visibility graph based time series analysis. *PLOS ONE*, 10(11):1–19, 11 2015. doi: 10.1371/journal.pone.0143015. URL <https://doi.org/10.1371/journal.pone.0143015>.
- [13] Markus Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. Lof: Identifying density-based local outliers. In *PROCEEDINGS OF THE 2000 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*, pages 93–104. ACM, 2000.
- [14] Andreas Fischer, Kaspar Riesen, and Horst Bunke. Improved quadratic time approximation of graph edit distance by combining hausdorff matching and greedy assignment. *Pattern Recognition Letters*, 87:55 – 62, 2017. ISSN 0167-8655. doi: <https://doi.org/10.1016/j.patrec.2016.06.014>. URL <http://www.sciencedirect.com/science/article/pii/S0167865516301386>. Advances in Graph-based Pattern Recognition.
- [15] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Netsimile: A scalable approach to size-independent network similarity. *CoRR*, abs/1209.2684, 2012. URL <http://arxiv.org/abs/1209.2684>.
- [16] G. Athithan N. N. R. Ranga Suri, Narasimha Murty. *Outlier Detection: Techniques and Applications*. Springer International Publishing, Springer Nature Switzerland AG, 2019. ISBN 978-3-030-05127-3.
- [17] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. *SIGMOD Rec.*, 30(2):3746, May 2001. ISSN 0163-5808. doi: 10.1145/376284.375668. URL <https://doi.org/10.1145/376284.375668>.
- [18] Andrew Yan-Tak Ng. Anomaly detection vs. supervised learning, August 2020. URL <https://www.coursera.org/lecture/machine-learning/anomaly-detection-vs-supervised-learning-Rkc5x>. [Online; accessed 19-August-2020].
- [19] Byron Ellis and Wing Hung Wong. Learning causal bayesian network structures from experimental data. *Journal of the American Statistical Association*, 103(482):778–789, 2008. ISSN 01621459. URL <http://www.jstor.org/stable/27640100>.
- [20] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41, 07 2009. doi: 10.1145/1541880.1541882.
- [21] Mohammad Braei and Sebastian Wagner. Anomaly Detection in Univariate Time-series: A Survey on the State-of-the-Art. *arXiv e-prints*, art. arXiv:2004.00433, April 2020.
- [22] Rui Zhang, Shaoyan Zhang, Sethuraman Muthuraman, and Jianmin Jiang. One class support vector machine for anomaly detection in the communication network performance data. In *Proceedings of the 5th Conference on Applied Electromagnetics, Wireless and Optical Communications, ELECTRO-SCIENCE'07*, page 3137, Stevens Point, Wisconsin, USA, 2007. World Scientific and Engineering Academy and Society (WSEAS). ISBN 9789606766251.

- [23] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph-based Anomaly Detection and Description: A Survey. *arXiv e-prints*, art. arXiv:1404.4679, April 2014.
- [24] G. B. Giannakis, Y. Shen, and G. V. Karanikolas. Topology identification and learning over graphs: Accounting for nonlinearities and dynamics. *Proceedings of the IEEE*, 106(5):787–807, 2018.
- [25] Liang Zhao. Event Prediction in the Big Data Era: A Systematic Survey. *arXiv e-prints*, art. arXiv:2007.09815, July 2020.
- [26] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 37–45, 1998.
- [27] Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuño. From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Science*, 105(13):4972–4975, April 2008. doi: 10.1073/pnas.0709247105.
- [28] Bartolo Luque, Lucas Lacasa, Fernando Ballesteros, and Jordi Luque. Horizontal visibility graphs: exact results for random time series. *arXiv e-prints*, art. arXiv:1002.4526, February 2010.
- [29] Gunjan Soni. *Signed visibility graphs of time series and their application to brain networks*. PhD thesis, University of British Columbia, 2019. URL <https://open.library.ubc.ca/collections/ubctheses/24/items/1.0378503>.
- [30] American Heart Association. Electrocardiogram (ecg or ekg), August 2020. URL <https://www.heart.org/en/health-topics/heart-attack/diagnosing-a-heart-attack/electrocardiogram-ecg-or-ekg>. [Online; accessed 19-August-2020].
- [31] Victor Mondejar. Plain text format of physionet mit-bih database, August 2020. URL <https://www.kaggle.com/mondejar/mitbih-database>. [Online; accessed 1-August-2019].
- [32] G. B. Moody and R. G. Mark. The impact of the mit-bih arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.
- [33] Bernd Porr Luis Howell. py-ecg-detectors: A collection of 7 ecg heartbeat detection algorithms, August 2020. URL <https://doi.org/10.5281/zenodo.3353396>. [Online; accessed 19-August-2020].
- [34] Paul van Gent. Analyzing a discrete heart rate signal, August 2020. URL <http://www.paulvangent.com>. [Online; accessed 19-August-2020].
- [35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [36] Jacques Fize. Gmatch4py a graph matching library, August 2020. URL <https://github.com/Jacobe2169/GMatch4py>. [Online; accessed 19-August-2020].

- [37] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.
- [38] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.
- [39] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.
- [40] Pranav Rajpurkar, Awni Y. Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y. Ng. Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks. *arXiv e-prints*, art. arXiv:1707.01836, July 2017.
- [41] S. Chauhan and L. Vig. Anomaly detection in ecg time signals via deep long short-term memory networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–7, 2015.