Detection of anomalies and identification of their precursors in large data series collections

Paul Boniol

Université de Paris

Université de Paris
Ecole doctorale EDITE (ED 130)
Laboratoire LIPADE

Detection of Anomalies and Identification of their Precursors in Large Data Series Collections

Par Paul Boniol

Thèse de doctorat d'informatique, science des données

Dirigée par Professeur Themis Palpanas

Présentée et soutenue publiquement le 29/11/2021

Devant un jury composé de :

Bernd Amann, Professeur des universités, Sorbonne Université, LIP6,	Examinateur
Michalis Vazirgiannis, Professeur des universités, Ecole Polytechnique, LIX,	Examinateur
Germain Forestier, Professeur des universités, Université Haute-Alsace, ENSISA,	Rapporteur
Karine Zeitouni, Professeur des universités, Université de Versailles, DAVID,	Rapporteur
Themis Palpanas, Professeur des universités, Université de Paris, LIPADE,	Directeur de thèse
Mohammed Meftah, Ingénieur de recherche, EDF R&D,	co-encadrant
Emmanuel Remy, Ingénieur de recherche, EDF R&D,	co-encadrant

Titre : Détection d'anomalies et identification de leur précurseurs dans des grandes collections de séries temporelles

Résumé : *Les larges collections de séries temporelles deviennent une réalité dans un grand nombre de domaines scientifiques et sociaux, comme la finance, les sciences de l'environnement, l'astrophysique, les neurosciences, l'ingénierie ou les métiers du numérique. Il y a donc un intérêt et un besoin de plus en plus importants de développer des techniques efficaces pour analyser et traiter ce type de données. De manière informelle, une série temporelle est une séquence ordonnée de points ou de valeurs. Une fois les séries collectées et disponibles, les utilisateurs ont souvent besoin de les étudier pour en extraire de la valeur et de la connaissance. Ces analyses peuvent être simples, comme sélectionner des fenêtres temporelles, mais aussi complexes, comme rechercher des similarités entre des séries ou détecter des anomalies, souvent synonymes d'évolutions soudaines et inhabituelles possiblement non souhaitées, voire de dysfonctionnements du système étudié. Ce dernier type d'analyse représente un enjeu crucial pour des applications dans un large éventail de domaines partageant tous le même objectif : détecter les anomalies le plus rapidement possible pour éviter la survenue de tout événement critique, comme par exemple de prévenir les dégradations et donc d'allonger la durée de vie des systèmes. Par conséquent, dans ce travail de thèse, nous traitons les trois objectifs suivants : (i) l'exploration non-supervisée de séries temporelles pour la détection rétrospective d'anomalies à partir d'une collection de séries temporelles. (ii) la détection non-supervisée d'anomalies en temps réel dans les séries temporelles. (iii) l'explication de la classification d'anomalies connues dans les séries temporelles, afin d'identifier de possibles précurseurs. Dans ce manuscrit, nous introduisons d'abord le contexte industriel qui a motivé la thèse, des définitions fondamentales, une taxonomie des séries temporelles et un état de l'art des méthodes de détection d'anomalies. Nous présentons ensuite nos contributions scientifiques en suivant les trois axes mentionnés précédemment. Ainsi, nous décrivons premièrement deux solutions originales, NormA (basée sur une méthode de clustering de sous-séquences de la séries temporelles à analyser) et Series2Graph (qui s'appuie sur une transformation de la séries temporelle en un réseau orienté), pour la tâche de détection non supervisée de sous-séquences anormales dans les séries temporelles statiques (i.e., n'évoluant pas dans le temps). Nous présentons dans un deuxième temps la méthode SAND (inspiré du fonctionnement de NormA) développée pour répondre à la tâche de détection non-supervisée de sous-séquences anormales dans les séries temporelles évoluant de manière continue dans le temps. Dans une troisième phase, nous abordons le problème lié à l'identification supervisée des précurseurs. Nous subdivisons cette tâche en deux problèmes génériques : la classification supervisée de séries temporelles d'une part, l'explication des résultats de cette classification par l'identification de sous-séquences discriminantes d'autre part. Enfin, nous illustrons l'applicabilité et l'intérêt de nos développements au travers d'une application portant sur l'identification de précurseurs de vibrations indésirables survenant sur des pompes d'alimentation en eau dans les centrales nucléaires françaises d'EDF.*

Mots clefs : série temporelle ; détection d'anomalies ; apprentissage machine ; intelligence artificielle

Title : Detection of anomalies and identification of their precursors in large data series collections

Abstract : *Extensive collections of data series are becoming a reality in a large number of scientific and social domains. There is, therefore, a growing interest and need to elaborate efficient techniques to analyze and process these data, such as in finance, environmental sciences, astrophysics, neurosciences, engineering. Informally, a data series is an ordered sequence of points or values. Once these series are collected and available, users often need to query them. These queries can be simple, such as the selection of time interval, but also complex, such as the similarities search or the detection of anomalies, often synonymous with malfunctioning of the system under study, or sudden and unusual evolution likely undesired. This last type of analysis represents a crucial problem for applications in a wide range of domains, all sharing the same objective: to detect anomalies as soon as possible to avoid critical events. Therefore, in this thesis, we address the following three objectives: (i) retrospective unsupervised subsequence anomaly detection in data series. (ii) unsupervised detection of anomalies in data streams. (iii) classification explanation of known anomalies in data series in order to identify possible precursors. This manuscript first presents the industrial context that motivated this thesis, fundamental definitions, a taxonomy of data series, and state-of-the-art anomaly detection methods. We then present our contributions along the three axes mentioned above. First, we describe two original solutions, NormA (that aims to build a weighted set of subsequences that represent the different behaviors of the data series) and Series2Graph (that transform the data series in a directed graph), for the task of unsupervised detection of anomalous subsequences in static data series. Secondly, we present the SAND (inspired from NormA) method for unsupervised detection of anomalous subsequences in data streams. Thirdly, we address the problem of the supervised identification of precursors. We subdivide this task into two generic problems: the supervised classification of time series and the explanation of this classification's results by identifying discriminative subsequences. Finally, we illustrate the applicability and interest of our developments through an application concerning the identification of undesirable vibration precursors occurring in water supply pumps in the French nuclear power plants of EDF.*

Keywords : data series ; anomaly detection ; machine learning

Université de Paris
Laboratoire d'Informatique Paris Descartes (EA 2517)
EDF R&D

Detection of Anomalies and Identification of their Precursors in Large Data Series Collections

Presented by Paul Boniol

PhD thesis in Computer Science

November 2021

Bernd Amann	Professeur, Sorbonne Université - LIP6	Examinateur
Michalis Vazirgiannis	Professeur, Ecole Polytechnique - LIX	Examinateur
Germain Forestier	Professeur, Université Haute-Alsace - ENSISA	Rapporteur
Karine Zeitouni	Professeur, Université de Versailles - DAVID	Rapporteur
Themis Palpanas	Professeur, Université de Paris, LIPADE	Directeur
Mohammed Meftah	Ingénieur de recherche, EDF R&D	Invité
Emmanuel Remy	Ingénieur de recherche, EDF R&D	Invité

Résumé

Les collections massives de séries temporelles deviennent une réalité dans un grand nombre de domaines scientifiques et sociaux. Il y a donc un besoin de plus en plus important par les acteurs des domaines concernés de développer des techniques qui peuvent les analyser efficacement. Par exemple, de tel domaines concernés qui impliquent des séries temporelles sont : la finance, les sciences de l'environnement, l'astrophysique, les neurosciences, l'ingénierie, le multimédia. De manière informelle, une série temporelles, ou séquence de données, est une séquence ordonnée de points ou de valeurs. Si la dimension ordonnant la séries est le temps, on parle alors de séries temporelles, mais les séries peuvent être ordonnées par d'autres mesures. Par exemple, de telles mesures peuvent être : l'angle dans les profils radiaux astronomiques, la fréquence dans la spectroscopie infrarouge, la masse dans la spectroscopie de masse, la position dans les séquences de génome). Lorsque ces collections de séquences sont générées, les utilisateurs peuvent avoir besoin de les analyser dès qu'elles sont disponibles. Ces analyses peuvent être simples, comme la sélection de fenêtres temporelles ou le calcul de la moyenne, mais aussi complexes, comme la recherche de similarités, le clustering, et la détection d'anomalies. La détection d'anomalies dans les séries temporelles est un problème crucial avec des applications dans un large éventail de domaines, qui partagent tous le même objectif: détecter les anomalies le plus rapidement possible pour éviter tout événement critique. Des exemples de telles applications peuvent être pertinent dans les domaines de la biologie, de l'astronomie et de l'ingénierie. Certains de ces domaines sont bien étudiés et théoriquement bien explorés. Les connaissances acquises par les experts du domaine peuvent être utilisées pour construire un algorithme qui vise à détecter efficacement tout type de comportement qui dérive d'une potentielle normalité bien défini. Cependant, de tels algorithmes peuvent être compliqués à développer ou peuvent avoir des difficultés à s'adapter à des changements inconnus ou mal définis dans le temps. D'un autre côté, si les données disponibles sont suffisamment représentatives pour illustrer correctement l'état de santé du système, une méthode s'adaptant et se basant automatiquement sur les données pourrait offrir plus de flexibilité. Par exemple, dans le cas de la détection des fraudes, un modèle basé seulement sur la connaissance des experts ne peut reconnaitre que les fraudes déjà connues. Au contraire, un modèle construit et évoluant en fonction des données collectées pourrait être utile pour trouver de nouveaux type de fraude, ce qui est crucial puisque les fraudes peuvent fréquemment changer et évoluer. En outre, dans le cas général de la détection d'anomalies, il est donc important de savoir construire des modèles apprenant et s'adaptant en fonction des données collectées. De plus, dans le cas spécifique où les anomalies sont connues et stockées dans une base de données, des méthodes se basant sur cette base de données peuvent être utilisées pour détecter les précurseurs de ces anomalies connues. Ces précurseurs (ou symptômes) peuvent indiquer à l'utilisateur les dangers et les défaillances potentiels futurs et prévenir les dommages critiques. Par conséquent, dans cette thèse, nous abordons les trois objectifs liés au problème général concernant la détection d'anomalies dans les séries temporelles.

- *Les séries de données étant stockées dans de grandes bases de données, le premier objectif est de pouvoir identifier les éventuelles anomalies survenues dans le passé. Une telle analyse peut aider les experts à explorer les séries temporelles et à identifier des comportements inhabituels. Les anomalies découvertes peuvent être utiles soit pour guider l'expert dans l'analyse d'un événement spécifique survenu dans le passé, soit pour filtrer les anomalies possibles dans les séries de données afin de créer des ensembles de données exempts d'anomalies pour les modèles d'apprentissage automatique supervisés. Voici un exemple : Une augmentation inhabituelle de la température est remarquée à l'intérieur d'un composant important d'une centrale hydroélectrique. Cette température augmente lentement sans que l'on sache exactement pourquoi. Les experts décident alors d'analyser les valeurs historiques des capteurs de la centrale, à la recherche d'explications possibles. A cause de la faible occurence dans le passé de l'anomalie (augmentation de temperature), il est impossible de créer une base de données et d'entraîner un modèle supervisé pour classer ces anomalies. Par conséquent, des méthodes non supervisées sont utilisées pour détecter les événements inhabituels qui pourraient être corrélés à la lente augmentation de la température.*

- *Le deuxième objectif est d'analyser les flux de données (provenant de capteurs retournant des valeurs en temps réel) et de déclencher des alarmes en temps réel lorsqu'un comportement inhabituel se produit. Ces alarmes peuvent aider les experts à prévenir les éventuelles pannes ou les coûts supplémentaires que le comportement inhabituel pourrait impliquer. L'analyse en temps réel est fortement liée à l'analyse rétrospective (objectif précédent). En effet, un modèle en temps réel peut être construit efficacement sur des ensembles de données sans anomalie ou pour vérifier les comportements inhabituels qui se sont déjà produits dans le passé. Voici un exemple : Les experts définissent des caractéristiques limites qui pourraient impliquer l'arrêt de la centrale électrique. Par exemple, des changements de température non désirés (atteignant une limite de sécurité) doivent être détectés en temps réel pour alerter les experts. La détection de ces évolutions inhabituelles (c'est-à-dire des sous-séquences inhabituelles) peut aider les experts à prendre la bonne décision.*

- *Le troisième objectif est d'aider le diagnostic des experts en expliquant l'occurrence d'un événement. Par exemple, cette explication peut correspondre à de sous-séquences spécifiques dans les séries temporelles qui pourraient expliquer ou aider à prédire une anomalie connue. Ces méthodes nécessitent une assistance humaine (en fournissant les événements ou anomalies connus). Voici un exemple : Grâce à un capteur booléen positionné sur une pompe dans la centrale électrique, les experts ont remarqué que, parfois, cette pompe vibre et bouge de quelques microns. Comme il n'y a pas d'explication claire de l'origine de ces vibrations, on peut utiliser des modèles qui permettent à la fois à reconnaître et à expliquer les vibrations. Par exemple, un modèle supervisé peut être entraîné sur une vaste collection de capteurs entourant la pompe. Ce dernier est entraîné à classer les événements contenant une vibration (étiquetés par les experts) des événements réguliers (sans aucune vibration). Une fois le modèle entraîné, on peut récupérer les caractéristiques discriminantes (c'est-à-dire les sous-sequences dans les séries temporelles générées par des capteurs qui pourraient expliquer pourquoi la vibration se produit) et les utiliser pour améliorer la compréhension des vibrations par les experts.*

Dans le premier chapitre, nous présentons la définition de base, la taxonomie et le contexte des séries temporelles et des méthodes de détection d'anomalies. Nous définissons ensuite les différentes tâches liées à la détection d'anomalies dans les séries temporelles. Enfin, nous passons en revue les approches les plus récentes et leurs limites, qu'elles soient non-supervisées, semi-supervisées ou supervisées. Dans le deuxième chapitre, nous abordons ensuite la tâche de détection non supervisée de sous-séquences anormales dans les séries temporelles statiques (i.e., n'évoluant pas dans le temps). Nous discutons d'abord brièvement des limites des approches actuelles de l'état de l'art. Nous présentons ensuite deux nouvelles approches adaptées à la détection d'anomalies sans contraintes liées au domaine d'application. Nous décrivons d'abord NormA, une nouvelle approche basée sur une nouvelle primitive de séries temporelles, qui permet de détecter les anomalies en fonction de leur similarité (ou dissimilarité) avec un modèle qui représente le comportement normal de la série temporelle initiale. Nous introduisons ensuite Series2Graph, une approche basée sur une modélisation des séries temporelle en réseaux, et qui vise plus précisément à intégrer les séries de données dans un graphe dirigé qui met en évidence les sous-séquences inhabituelles et potentiellement anormales. Nous présentons ensuite une analyse expérimentale comparant la précision de la détection des anomalies et le temps d'exécution de NormA et Series2Graph et des approches de l'état de l'art actuelles. Dans le troisième chapitre, nous abordons la tâche de détection non-supervisée de sous-séquences anormales sur des séries temporelles évoluant de manière continue dans le temps. Nous présentons tout d'abord les limites que les approches de l'état de l'art s'appliquant aux séries temporelles statiques peuvent avoir lorsqu'elles sont utilisées pour des flux de données continue. Nous présentons ensuite SAND, une nouvelle méthode s'utilisant en temps réel et adaptée à la détection d'anomalies sans contraintes liées au domaine d'application. Cette dernière s'appuie sur une nouvelle méthodologie de clustering pour mettre à jour de manière incrémental son modèle. Ce dernier s'adapte donc aux changement de distribution et omet les données obsolètes. Nous présentons enfin une analyse expérimentale qui compare SAND aux approches statiques et streaming (i.e., évoluant en temps réel) de l'état de l'art sur une large base de données de séries temporelles. Dans le quatrième chapitre, nous abordons le problème liée à l'identification supervisée des précurseurs. Cette tâche peut être résolue en considérant d'abord la tache précédente comme un problème de classification supervisée des séries temporelles, puis en analysant le problème de l'explication de la classification. Dans ce chapitre, nous nous concentrons sur le problème de l'explication de la classification. Nous discutons d'abord brièvement des limites des approches actuelles. Nous présentons ensuite dCNN et dCAM, une nouvelle architecture de réseau de neurones convolutif et profond et une nouvelle Class Activation Map pour chaque dimension (i.e., chaque capteurs). Contrairement à la Class Activation Map proposée comme telle, la méthode dCAM peut identifier des caractéristiques discriminantes (ou sous-séquences) dans toutes les dimensions. Nous présentons ensuite une analyse expérimentale dans laquelle nous comparons notre approche avec les approches existantes. Dans le cinquième et dernier chapitre, nous illustrons l'applicabilité et l'intérêt de nos méthodes dans une application industrielle proposée par le partenaire industriel de cette thèse : EDF R&D (département de recherche et de développement d'EDF). Nous étudions les précurseurs de la détection des vibrations indésirables survenant dans les systèmes de pompes d'alimentation en eau dans les centrales nucléaires françaises. Nous décrivons d'abord le jeu de données et le contexte industriel. Nous explorons ensuite deux scénarios : nous abordons d'abord la tâche comme si les experts d'EDF ne fournissaient aucune information sur les vibration à détectées. Dans ce premier scénario, nous appliquons NormA et Series2Graph. Nous évaluons leur précision en utilisant les infor-

mations fournis par les experts. Nous abordons ensuite la tâche comme un problème supervisé (en utilisant les informations fournie par les experts) et appliquons dCNN/dCAM. Nous montrons dans ce chapitre les bénéfices apporté par les méthodes supervisées comparé au méthodes non-supervisées. Nous démontrons ensuite l'intérêt de l'architecture dCNN et la méthode d'explication dCAM pour l'explication de l'apparition de vibration. Pour cela, nous analysons premièrement les capteurs les plus identifié par la dCAM comme contribuant le plus à la classification entant que « contenant une vibration ». Nous montrons que ces capteurs sont cohérent avec leur positionnement structurel vis-à-vis de la pompe vibrante. Finalement, nous analysons les formes des sous-séquences (de chaque capteur identifié comme important) détectées par la dCAM et les confrontons aux connaissances des experts du domaines.

Abstract

Extensive collections of data series are becoming a reality in a large number of scientific and social domains. There is, therefore, a growing interest and need to elaborate efficient techniques to analyze and process these data, such as in finance, environmental sciences, astrophysics, neurosciences, engineering. Informally, a data series is an ordered sequence of points or values. Once these series are collected and available, users often need to query them. These queries can be simple, such as the selection of time interval, but also complex, such as the similarities search or the detection of anomalies, often synonymous with malfunctioning of the system under study, or sudden and unusual evolution likely undesired. This last type of analysis represents a crucial problem for applications in a wide range of domains, all sharing the same objective: to detect anomalies as soon as possible to avoid critical events. Therefore, in this thesis, we address the following three objectives: (i) retrospective unsupervised subsequence anomaly detection in data series. (ii) unsupervised detection of anomalies in data streams. (iii) classification explanation of known anomalies in data series in order to identify possible precursors. This manuscript first presents the industrial context that motivated this thesis, fundamental definitions, a taxonomy of data series, and state-of-the-art anomaly detection methods. We then present our contributions along the three axes mentioned above. First, we describe two original solutions, NormA (that aims to build a weighted set of subsequences that represent the different behaviors of the data series) and Series2Graph (that transform the data series in a directed graph), for the task of unsupervised detection of anomalous subsequences in static data series. Secondly, we present the SAND (inspired from NormA) method for unsupervised detection of anomalous subsequences in data streams. Thirdly, we address the problem of the supervised identification of precursors. We subdivide this task into two generic problems: the supervised classification of time series and the explanation of this classification's results by identifying discriminative subsequences. Finally, we illustrate the applicability and interest of our developments through an application concerning the identification of undesirable vibration precursors occurring in water supply pumps in the French nuclear power plants of EDF.

Contents

Résumé **i**

Abstract **v**

Table of contents **ix**

List of figures **xv**

List of tables **xvii**

1 Introduction **1**
1.1 Introduction . . . 2
1.2 Motivation: Industrial Context . . . 2
1.2.1 General Objectives . . . 3
1.3 Contributions . . . 4
1.3.1 Unsupervised subsequence anomaly detection . . . 4
1.3.2 Streaming Unsupervised subsequence anomaly detection . . . 7
1.3.3 Supervised identification of anomaly precursors . . . 9
1.3.4 Pump Vibration Case . . . 10
1.4 Thesis Outline . . . 11
1.5 List of publications . . . 12

2 Background and Related Work **15**
2.1 Anomaly Detection Primer . . . 17
2.1.1 Point versus Subsequences . . . 17
2.2 Preliminaries on Data Series . . . 17
2.2.1 *Univariate* versus *Multivariate* . . . 17
2.2.2 *Static* versus *Streaming* . . . 18
2.2.3 *Discrete* versus *Continous* . . . 18
2.2.4 Real-World constraints: *Missing* points and *non-synchronized* data series . . . 19
2.2.5 Data Series Notations . . . 19
2.3 Preliminaries on Anomaly Detection Methods . . . 19
2.3.1 *Unsupervised*: No prior information . . . 20
2.3.2 *Supervised*: prior information . . . 20
2.4 Problems Formulation . . . 20
2.4.1 Unsupervised Subsequence Anomaly Detection . . . 21
2.4.2 Supervised Detection of Anomaly: Identification of their precursors 22
2.5 Unsupervised Subsequences Anomaly Detection in Data Series . . . 24
2.5.1 Data Series Distances Definitions . . . 24
2.5.2 Data-Series Clustering . . . 26
2.5.3 Density based Approaches . . . 27
2.5.4 Subsequence Similarity-based Approaches . . . 30
2.5.5 New perspectives with graph-based approaches . . . 35
2.6 Supervised Subsequences Anomaly Detection in Data Series . . . 39

2.6.1 Semi-Supervised Machine Learning Approaches . . . 40
2.6.2 Supervised Machine Learning Approaches . . . 46
2.6.3 Finding Precursors of Abnormal Subsequences . . . 51
2.7 Summary . . . 53

3 Unsupervised Subsequence Anomaly Detection 55
3.1 Limitations of Current Approaches . . . 57
3.2 Proposed Approaches: a Generic idea . . . 57
3.3 NormA: Set-based modeling of the data series . . . 58
3.3.1 Normal Model Definition . . . 58
3.3.2 Computational Steps . . . 60
3.3.3 Overall Algorithm and Complexity Analysis . . . 67
3.4 Series2Graph: Graph-based modeling of the data series . . . 68
3.4.1 Subsequence Graph Definition . . . 68
3.4.2 Computational Steps . . . 70
3.4.3 Overall Algorithm and Complexity Analysis . . . 79
3.5 Experimental Evaluation . . . 81
3.5.1 Implementation . . . 81
3.5.2 Description of the datasets . . . 82
3.5.3 Description of the evaluation metrics . . . 82
3.5.4 Description of the baselines . . . 83
3.5.5 NormA Parameters influences . . . 83
3.5.6 Series2Graph Parameters Influences . . . 85
3.5.7 Accuracy Evaluation . . . 90
3.5.8 Execution Time Evaluation . . . 91
3.5.9 User Interfaces: Stand Alone Web Applications . . . 93
3.6 Summary . . . 95

4 Streaming Subsequence Anomaly Detection 97
4.1 Limitations of Previous approaches for the Streaming case . . . 99
4.2 Proposed approach: SAND . . . 99
4.2.1 Overview . . . 100
4.2.2 Model initialization . . . 102
4.2.3 Continuous Model Update . . . 103
4.2.4 Subsequence Scoring . . . 107
4.2.5 Execution Time Complexity Analysis . . . 108
4.2.6 Parameters Influences . . . 109
4.3 Experimental Evaluation . . . 112
4.3.1 Implementation . . . 112
4.3.2 Description of the evaluation metrics . . . 112
4.3.3 Description of the datasets . . . 113
4.3.4 Description of the baselines . . . 113
4.3.5 Accuracy Evaluation . . . 114
4.3.6 Time Performance Evaluation . . . 117
4.3.7 User Interface: StandAlone Web Application . . . 119
4.4 Summary . . . 120

5 Supervised Identification of precursors of anomaly 121
5.1 Class Activation Map: an interesting tool for anomaly precursor identification 123
5.1.1 Limitations for Multivariate Data Series anomaly detection 123
5.2 A first Baseline for Multivariate Data Series 123
5.2.1 cCNN: a first architecture 124
5.2.2 Limitations 124
5.3 Proposed Approach: Dimension-wise Class Activation Map 124
5.3.1 Dimension-wise Architecture 125
5.3.2 Limitation of CAM for Dimension-wise Architecture 127
5.3.3 Computation of dCAM 128
5.4 Experimental Evaluation 132
5.4.1 Experimental Setup 132
5.4.2 Baselines and Training Setup 134
5.4.3 Classification Accuracy evaluation 134
5.4.4 Discriminant Feature identification evaluation 136
5.4.5 $C\text{-}acc$ versus $Dr\text{-}acc$ 137
5.4.6 Execution time evaluation 139
5.4.7 Use Case: Surgeon skills explanation 140
5.5 Summary 143

6 Use Case: Pump Vibration Case 145
6.1 Use Case: Precursors of Anomalous Vibration Detection in Nuclear Power Plants 147
6.1.1 Dataset and Use case description 147
6.1.2 Unsupervised Approaches 149
6.1.3 Supervised Approaches 149
6.2 Experimental Analysis 149
6.2.1 Vibration Detection Evaluation 150
6.2.2 Precursors identification Quantitative Evaluation 158
6.2.3 Precursor identification Qualitative Evaluation 160
6.3 Summary and Conclusion 162

7 Conclusions and Perspectives 165
7.1 Contributions Summary 166
7.2 Open Research Directions 167
7.2.1 Series2Graph and NormA for Multivariate Data Series 167
7.2.2 Series2Graph for Streaming Data Series 167
7.2.3 Distributed implementations 168
7.2.4 Non-linear projection for Series2Graph first step 168
7.2.5 Optimization of Memory and Execution Time Complexity for dCAM 168
7.2.6 From local to global explanation with dCAM 169
7.2.7 Adding temporal explanation to dCAM 169
7.2.8 Applicability to non-continuous data series 169
7.2.9 Exploration of other industrial use cases 170

Bibliographie 182

List of Figures

2.1 Synthetic examples where (a) a point outlier can be detected by looking at its position in the distribution (for both univariate (a.1) and multivariate (a.2)). (b) a sequence outlier is taking part of the *normal* range of values (by taking every point independently without any temporal structure for both univariate (b.1) and multivariate (b.2)). 18

2.2 Unsupervised subsequence anomaly detection task illustrated using a nuclear power plant use case, in which (a) illustrates the sensors data series coming from (b) subsystems of a given electrical power plant. The goal is to find unknown abnormal subsequences in the sensors data series (illustrated by the red rectangle). 21

2.3 Supervised detection of anomaly precursors task illustrated using a nuclear power plant use case, in which (a) illustrates the sensors data series coming from (b) subsystems of a given electrical power plant. The goal is to find, from the known abnormal subsequences in the sensors data series (illustrated by the red rectangle), the potential precursors of this anomaly (illustrated by the light red rectangles). 22

2.4 Comparison between the alignment considered by Euclidian distance(a) and DTW distance (b). 25

2.5 (a) For a given data series T, illustration of the reachability distance between A and B and then A and C for $k = 4$ (with $A, B, C, D \in \mathbb{T}_\ell$). (b) Difference between $rd_k(A, X), X \in NN_k(A)$, when A is an anomaly and B, C, D are regular subsequences in data series T. 28

2.6 LOF_k computed for $k = 20, 30, 40, 50, 60, 70, 80$ using a 10000 snippet (middle plot) and 100000 snippet (bottom plot) of MBA(803). In the big snippet, due to the higher number of anomaly, LOF_k method is not able to detect them. 29

2.7 Set of isolation trees that randomly partition a dataset. In average, for a given data series T, the subsequence $N \in \mathbb{T}_\ell$ has a longer path to the root than the subsequence $A \in \mathbb{T}_\ell$, thus A anomaly score will be higher. 30

2.8 A dataset with 36 subsequences (of the same length ℓ) depicted as points in 2-dimensional space; 27 subsequences are normal (blue points), and 9 are anomalous (solid, red points). 32

2.9 (a) Matrix profile (a_2) applied on the SED (Nasa disk failure datasets) data series snippet (a_1). The highest value in the matrix profile (a_1) points to an anomaly of the SED data series. (b) Matrix profile (b_2) applied on a synthetic data series (b_1). The smallest value in the matrix profile (a_1) points to a motif pair in the data series. 34

2.10 (a) Synthetic data series generated with random noise with two similar random subsequences injected (red). (b) Matrix profile of the data series in (a). (c) Two similar random subsequences injected in (a). 34

2.11 Nearest neighbor euclidean distance (in green) computed using Matrix Profile for (a) an electrocardiogram containing similar supra-ventricular contractions and (b) an electrocardiogram containing similar premature heartbeats. 35

2.12 For a given data series T, (a) the corresponding Natural Visibility Graph (NVG), and (b) the corresponding Horizontal Visibility Graph (HVG). . . . 38
2.13 One class SVM illustration in which a point corresponds to a subsequence and only the green point are provided for the training step. 41
2.14 LSTM unit architecture. 43
2.15 Illustration of GAN Network. 44
2.16 Autoencoder architecture framework for data series subsequences anomaly detection. The loss $\mathcal{L}$ is used to both train the model (on the non anomalous subsequences), and score the new subsequences. 46
2.17 Convolutional Neural Network architecture for multivariate data series. . . 49
2.18 Residual Neural Network architecture for multivariate data series. 50
2.19 Illustration of Class Activation Map for CNN architecture. 51

3.1 Illustration of the two subsequence anomaly definitions proposed approaches: (a) NormA; (b) Series2Graph. 58
3.2 (a) Normal Model of series T (shown in (b)), composed of n cluster centroids (thick lines) of subsequences (thin lines) of T. Each subsequence of T is compared to all centroids N_M^i weighted by w^i (black box). (c) Anomaly score d_{N_M} of all subsequences of T: subsequence $T_{j,\ell}$ is normal (low score), while $T'_{j,\ell}$ is an anomaly (high score). 60
3.3 (a) $Norm$ cluster scoring of MBA ECG recordings (patient 803); (b) $Norm$ cluster scoring of the concatenation of two MBA ECG recordings (patient 803 and 805). 66
3.4 3-Normality, 2-Normality, 1-Normality, and 3-Anomaly, 2-Anomaly for two given graphs ((a),(b) and (c),(d)) representing the simplified model of two data series. Edge weights and node degrees are used to define the θ-Normality and θ-Anomaly subgraphs. 69
3.5 Series2Graph steps in order to build the graph from a data series (a): embed the subsequences (b), create the nodes (c), and extract the edges (d). . . . 70
3.6 (a) $Proj_r(T, \ell_G, \lambda)$ and (b) $SProj(T, \ell_G, \lambda)$ of a data series T corresponding to the movement of an actor's hand that (c) takes a gun out of the holster and points to a target (normal behavior); (d) the anomaly (red subsequence) corresponds to a moment when the actor missed the holster [59]. We rotate (a) into (b) such that $\vec{v}_{ref}$ is invariant in two dimensions. 73
3.7 Node extraction from $SProj(T, \ell_G, \lambda)$ by measuring the density of the intersected trajectories to a given vector. The densest points are added to the Node Set $\mathcal{N}$. 75
3.8 Synthetic datasets. (a) Random walk sequence (left), and sinusoid signal following the same trend (right) with injected anomalies (red/bold subsequences). (b) A second example, with 20% of Gaussian noise added on top. 82
3.9 Distance measure impact experiment. (a) NormA-smpl accuracy score for MBA(803) for sbd, DTW and $Euclidean$ distances. (b) Overall accuracy for all the MBA datasets. 84
3.10 Precision@k and execution time (in seconds) over MBA and NASA datasets (a) when we vary r for NormA-smpl, when we vary l_{N_M} (b) for NormA-SJ and (c) for NormA-smpl. 85

3.11 Precision@k and execution time (in seconds) over MBA and NASA datasets (a) when we vary λ/ℓ_G ratio, (b) when we vary the number of radius subset r (c) when we vary the graph subsequence length ℓ_G (while keeping the same targeted subsequence length ℓ to score) (d) when we vary the bandwidth ratio $h/\sigma((I)_\psi)$. 86
3.12 Effect of anomaly cardinality on Series2Graph detection accuracy 87
3.13 $G_\ell(\mathcal{N}, \mathcal{E})$ of the MBA(820) electrocardiogram data series for ℓ of $80, 100$ and 120. In the three cases, the different kinds of anomalies (S: blue, V: red) are well separable with lower edges weights. 89
3.14 On the MBA and SED datasets: (a) Precision@k of Series2Graph varying the input and query length ($2\ell/3 = \ell_G$) to build the graph G_{ℓ_G}. (b) STOMP Precision@k varying the input length ℓ_G, (c) STOMP and Series2Graph in Precision@k average compared to the input length ℓ_G. 89
3.15 Critical difference diagram ($\alpha = 0.05$) for the data series of Table 3.4. . . . 91
3.16 Execution time (in seconds) evaluation for the baselines and our two proposed approaches NormA and Series2Graph when we vary (a) the data series length, (b) the number of anomalies, (c) the subsequence length ℓ. . . . 92
3.17 Screenshots of user interfaces GraphAn, dedicated to Series2Graph. 94
3.18 Screenshots of user interfaces SAD, dedicated to NormA. 95

4.1 SAND computation framework. 101
4.2 Influence of a ($a = \ell_\Theta/\ell$) over all double normality datasets on Precision@k (a) and execution time in sec (b). 110
4.3 Influence of batch size b_{size}, rate of change α, and initial number of clusters k on accuracy (1st line), execution time (2nd line) and final number of clusters created (3rd line), over all double normality datasets. 111
4.4 Comparison between the original and our extended k-Shape on rand score (a), and execution time (b). 116
4.5 Critical difference diagrams using a Wilkoxon pair-wised signed rank test (with $\alpha = 0.05$) on both single and multiple normality datasets. 117
4.6 Throughput vs batch size (with fixed subsequence length $\ell = 75$), subsequence length (with fixed batch size $b_{size} = 20000$), and position in the stream (with $\ell = 75$ and $b_{size} = 10000$). 118
4.7 Screenshots of the user interfaces of SAND. 119

5.1 Illustration of Class Activation Map for cCNN architecture with three convolutional layers (n_{f1}, n_{f2}, and n_{f3} different kernels respectively of size all equal to ℓ). 125
5.2 dCNN architecture and application of the CAM. 126
5.3 Example of Class Activation Map results for different permutations. 128
5.4 Transformation $\mathcal{M}$ for a given data series T. 129
5.5 Example of one row of matrix $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ for a specific dimension D. 130
5.6 dCAM computation framework. 131
5.7 Synthetic datasets: (a) $Type\ 1$, in which the discriminant subsequence is two injected patterns from class 2 StarLightCurves dataset in random dimensions at random positions, (b) $Type\ 2$, in which the discriminant factor is the fact that the two injected patterns are injected at the same position. . 133

5.8 C-acc comparison of (a) dCNN with cCNN and CNN, (b) dResNet with cResNet and ResNet, and (c) dInceptionTime with cInceptionTime and InceptionTime on UCR/UEA datasets. 136

5.9 Evaluation of the influence of the number of dimensions on our approaches and the baselines C-acc and Dr-acc. 138

5.10 Evaluation of C-acc, Dr-acc and the ratio between the number of permutations k and the number of permutations correctly classified n_g for dCNN, dResNet and dInceptionTime. 139

5.11 Execution time (in second) for the (a) training computations when we vary (a.1) the data series length and (a.2) the number of dimensions. (b) Execution time for dCAM computation (using dCNN, dResNet and dInceptionTime) when we vary (b.1) the number of dimensions, (b.2) the data series length, (b.3) the number of permutations k. 140

5.12 Example of the result of dCAM on a multivariate data series of the JIGSAWS dataset. (a) the depicted data series corresponds to a novice surgeon performing a suture operation (joined with the corresponding dCAM in (b)). General statistical results over the entire novice class $\mathcal{C}_N$ are depicted such as (c) box-plots of the maximal activation value per sensor and (d) the averaged activation per sensor per gesture performed. 141

6.1 Simplified schema of the secondary circuit of EDF 1300MW nuclear power plant. We collect in total 120 sensors from 8 subsystems (solid black boxes) surrounding the feed-water pumps (TPA). Blue arrows: water flow. Red arrows: steam flows. 148

6.2 Example of anomaly score produce by Series2Graph algorithm for time series containing a vibration from three sensors. 151

6.3 For each sensor, maximal NormA anomaly score distribution for time series with a vibration (in red) and without any vibration (in blue). The distribution plots are sorted based on the difference of the red and blue distribution (in that order) as depicted in the title of each subplot. 152

6.4 For each sensor, maximal Series2Graph anomaly score distribution for time series with a vibration (in red) and without any vibration (in blue). The distribution plots are sorted based on the difference of the red and blue distribution (in that order) as depicted in the title of each subplot. 153

6.5 ROC, PR, and Accuracy curve for Series2Graph and NormA evaluated on each sensor separately and using the maximal anomaly score on the entire instance. 154

6.6 Detection Accuracy measures (Area Under the Curve for ROC, PR, and maximal accuracy that can be obtained with the best threshold) for Series2Graph and NormA evaluated on each sensor separately and using the maximal anomaly score on the entire instance. 155

6.7 Training phase of dCNN. Loss and accuracy evolution when the number of epoch are increasing. 157

6.8 Example of dimension-wise Class Activation Map (dCAM) for one multivariate data series instance from the vibration class (in blue: the non-activated parts of the data series, in yellow: the activated, and thus discriminant, subsequences of the data series). 158
6.9 Aggregated activation score for dCAM per sensors for every timestamps. In red: the sensors names that are overall highly activated and that possibly contains one or several precursors. 159
6.10 Aggregated NormA (A.1), Series2Graph (A.2) and dCAM activation score for all sensors (B.1). Aggregated dCAM activation scores for some specific sensors (B.2). Red shades correspond to quantile interval (0.05-0.95,0.10-0.90,0.15-0.85, etc for (A and B.1) and only 0.3-0.7, 0.4-0.6 for (B.2)). The solid red line is the median values for each timestamp. 161
6.11 Subsequences clusters (*.*.1) (and their time distribution compared to the vibration timestamps (*.*.2)) detected as precursors of vibration by dCAM for the 9 most activated sensors. 162

7.1 Local explanation with (b) dCAM over (a) a data series T. Open-research directions aims to obtain (c) a causal-dCAM and (d) a global explanation aggregating all the dCAM over the entire dataset. 168

List of Tables

2.1 Summary and taxonomy of methods listed in this chapter. Note that none of these approaches can handle missing data points and unsynchronized data series. Preprocessing steps are thus needed. 53

3.1 NormA table of symbols 60
3.2 Series2Graph table of symbols. 71
3.3 List of dataset characteristics: series length, anomaly length (ℓ), number of annotated anomalies ($\mathbf{N_A}$), field. 83
3.4 P@k accuracy for LOF, IF, DAD, STOMP, GV, LSTM-AD, NormA-smpl (standard deviation over 100 runs shown in parenthesis), and NormA-SJ and Series2Graph. We set k equal to the number of anomalies and ℓ to the length of the (annotated) anomalies. 90

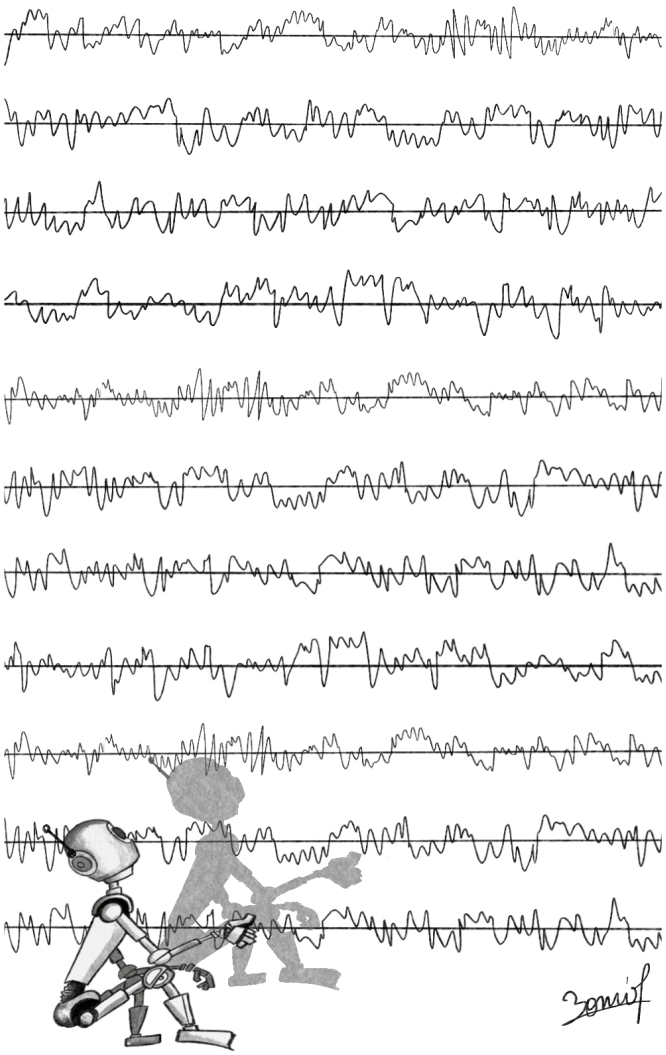
4.1 Table of symbols. 100
4.2 Precision@k accuracy for NormA (and NormA-batch), Isolation Forest (IF), STOMP, S2G (and S2G-batch), IMondrian Forest, STAMPI, and SAND applied to our datasets corpus (including concatenations of different datasets from same and different domains). The standard deviation of 10 runs is reported in parentheses. 115

5.1 dCAM Table of symbols. 124
5.2 C-acc averaged accuracy for 10 runs for CNN, ResNet, InceptionTime, cCNN, cResNet, cInceptionTime, dCNN, dResNet, dInceptionTime over UCR/UEA datasets. 135
5.3 C-acc and Dr-acc averaged accuracy for 10 runs for ResNet, cResNet, dCNN, dResNet, dInceptionTime over synthetic datasets. 137

6.1 Table of symbols and acronyme related to the use case 147

Chapter 1

Introduction



1.1 Introduction

Massive collections of data series are becoming a reality in virtually every scientific and social domain, and there is an increasingly pressing need by relevant applications for developing techniques that can efficiently analyze them [4, 91]. Examples of fields that involve data series are finance, environmental sciences, astrophysics, neuroscience, engineering, multimedia, etc. [90, 132, 91, 4]. Informally, a data series, or data sequence, is an ordered sequence of data points. If the ordering dimension is time, then we talk about time series. Though, series can be ordered over other measures (e.g., angle in astronomical radial profiles, frequency in infrared spectroscopy, mass in mass spectroscopy, position in genome sequences, etc). When these sequence collections are generated, users may need to query and analyze them as soon as they become available. These queries can be simple, such as range selection or mean computation, but can also be complex, such as similarity search, clustering, and anomaly detection.

Data series anomaly detection is a crucial problem with application in a wide range of domains [90] that all share the same well studied goal [8, 107, 124]: detecting anomalies the fastest as possible to avoid any critical event to come. The knowledge acquired by experts can be used to build an algorithm that aims to detect efficiently any kind of behavior that derives from a potential well-defined *normality*. However, such approaches can be complicated to concretize and might have difficulty adapting to unknown or unclear changes over time. On the other side, if the data available are representative enough to illustrate the system's health state correctly, a data-driven method could provide more flexibility. For instance, in the case of fraud detection, a knowledge-based model looks for known frauds, while a data-driven model might be helpful to find new patterns, which is crucial since frauds can change frequently and dynamically. Furthermore, in the general case of anomaly or outlier detection, the same statement can be made. Moreover, in the specific case when anomalies are known and stored in a dedicated database, data-driven methods can be used to detect precursors of these known anomalies. Such precursors (or symptoms) might indicate the user of potential future danger and failures and prevent critical damages.

1.2 Motivation: Industrial Context

As mentioned above, the theoretical task of finding anomalies meets real industrial needs. The desire to analyze a large quantity of data efficiently and being able to express complex queries (i.e., anomalies discovery) can be crucial for industrial actors. EDF (one of the major international electric utility) is one of these actors. One crucial goal for EDF is to improve the safety and availability of its electrical power plants by detecting anomalies that could occur. As massive gains are expected from reducing maintenance volumes, there is thus a serious need to have accurate and efficient algorithms to detect anomalies and understand their origins. Moreover, EDF has been collecting sensor data in every nuclear power plant for decades (at least more than 20 years). Knowing that one sensor can return one value per two seconds, an entire history of values (i.e., a data series) can reach 315,000,000 points. With a total of 58 nuclear power plants, with more than 2000 sensors per unit, it represents a data series database of more than 33 trillion points. Considering that one value requires

16 bits of storage, a rough estimation indicates that such a dataset corresponds to 500 TeraBytes of data. Thus, to benefit from the collected data of the past, one needs to have at disposal an efficient method to visualize and analyze subsequences, and more significantly, detect abnormal subsequences. In the following chapter, we describe and enumerate existing methods to solve subsequence anomaly detection by taking care of the applicability of such approaches to EDF database context. Moreover, despite the extensive work done by experts, anomaly annotations in data series are either non-existing or difficult to handle by hand or automatically by a computer code. Thus, one needs to present both supervised, semi-supervised and unsupervised methods.

Finally, whereas we discussed the electricity production side in the previous paragraph, EDF also collects a significant amount of data series on the consumer side. For instance, such data series can be either the electricity consumption in a private house or the global electricity consumption of an entire country. For the first case, detecting abnormal electrical consumption can help the consumer or EDF prevent an electric appliance from failing before causing any damage or unwanted electrical cost. In the second case, detecting abnormal days in a country's global consumption can help the experts adapt their production plan and optimize the electricity cost.

1.2.1 General Objectives

Overall, as described in the previous section, the anomaly detection task in data series is a large and challenging problem, finding application in almost all the activities of a multinational electric utility. On EDF side, the anomaly detection task can be divided into three main objectives:

1. **Retrospective subsequence anomaly detection**: As historical data series are stored in large databases, the first objective is to be able to identify in the databases possible anomalies that occurred in the past. Such analysis can help the expert to explore the historical data series and identify unusual behaviors. The latter can be helpful to either guide the expert analysis of a specific event that happened in the past or filter out possible anomalies from the data series to create anomaly-free datasets for supervised machine learning models. Here is an example: An unusual temperature increase is noticed inside an important component in a hydropower plant. This temperature is increasing slowly without any clear understanding. The experts then decide to analyze the historical values of sensors in the plant, looking for clues. As these events did not happen many times, one cannot create a database of similar events and train a supervised model to classify such events. Therefore, unsupervised methods are used to detect unusual events that might correlate with the slow temperature increase.

2. **Real-time analysis**: The second objective is to analyze data stream (such as sensors returning values in real-time) and raise alarms in real-time when an unusual behavior occurs. Such alarms can help the experts to prevent possible failures, damages, or extra costs that the unusual behavior could cause. Real-time analysis is strongly linked to retrospective analysis. Indeed, a real-time model can be learned efficiently

on anomaly-free datasets (built through a retrospective analysis) or to check for unusual behaviors that already happened in the past. Here is an example: Experts set levels of issues that might imply shutting down the power plant. For instance, Unwanted temperature changes (reaching a safety limit) need to be detected in real-time to alert the experts. Detecting such unusual evolutions (i.e., unusual subsequences) can help the expert make the right decision.

3. **Anomaly explanation and understanding**: The third objective is to assist experts' diagnosis by explaining the occurrence of an event. For instance, such explanation can be some specific subsequences in the data series generated by a sensor that could explain or help to predict a known anomaly. Such methods require human assistance (by providing the known events or anomalies) but can create a real-time model that uses identified subsequence to predict future known events or anomalies. Here is an example: Thanks to a boolean sensor positioned on a specific water pump in the power plant, experts noticed that this pump is sometimes vibrating and moving of few microns. As there is no clear understanding and explanation of the origin of these vibrations, one can use models that both learn to recognize and explain the vibrations. For example, a supervised model can be trained on an extensive collection of sensors surrounding the pump. The latter is trained to classify the events containing a vibration (labeled by the experts) from regular events (without any vibration). Once the model is trained, one can retrieve the discriminant features (i.e., the sections in the sensors that might explain why the vibration occurs) and use them to improve the expert understanding of the vibrations and prevent them in the future.

1.3 Contributions

Therefore, this thesis tackles the three objectives mentioned above. We first present the related state-of-the-art approaches for each of these goals. We then present our contributions in each of the three following topics: (i) We first tackle the case of unsupervised subsequence anomaly detection in data series. This topic is strongly related to the retrospective analysis objectives illustrated in the previous section. (ii) We then consider the case of streaming subsequence anomaly detection. The latter is closely related to the real-time analysis objectives described in the previous section. (iii) We finally deal with the supervised identification of anomaly precursors by solving the classification explanation task. This case is closely related to the Explanation and understanding objectives introduced in the previous section. We now describe in detail each contribution in each topic considered in this thesis.

1.3.1 Unsupervised subsequence anomaly detection

We first start by introducing the contribution related to the unsupervised subsequence anomaly detection task. In the specific context of sequences, which is the focus of this thesis, we are interested in identifying anomalous subsequences, which means that the outlier is not a single value but rather a sequence of values. This distinction is crucial

for the following reason: even though all individual values in a subsequence look normal when examined independently from one another, the sequence of these same values may be anomalous (e.g., the trend or shape of the subsequence may not be normal). Therefore, subsequence anomaly detection is a beneficial and essential operation for many real-world applications because it enables the early identification of problems that would otherwise remain undetected until too late [3].

1.3.1.1 Limitations of Current Approaches

Existing techniques either explicitly look for a set of pre-determined types of anomalies [44, 1], or identify as anomalies the subsequences with the largest distances to their nearest neighbors (termed *discords*) [124, 104]. We observe that these approaches pose limitations to the subsequence anomaly identification task for several reasons, explained below.

First, anomalous behavior is not always known. Therefore, techniques that use specific domain knowledge for mining anomalies (e.g., in cardiology [44], and engineering [3]) involve several finely-tuned parameters and do not generalize to new cases and domains. For example, early detection of anomalies in bearings (rolling elements in rotating machines, such as an aircraft engine) is of great importance for engine manufacturers.

Second, in the case of general, domain-agnostic techniques for subsequence anomaly detection, the state-of-the-art algorithms (e.g., [124, 104]) have been developed for the case of a *single* anomaly in the dataset, or multiple different (from one another) anomalies. The reason is that these algorithms are based on the distance of a subsequence to its nearest neighbor in the dataset: the subsequence that has the farthest nearest neighbor is marked as an anomaly.

Third, in order to remedy this situation, the m^{th} *discord* approach has been proposed [120]. This approach takes into account the multiplicity, m, of the anomalous subsequences that are similar to one another and marks as anomalies all the subsequences in the same group by computing the m^{th} (instead of the 1^{st}) nearest-neighbors for each subsequence. Nevertheless, this approach assumes that we know the multiplicity m, which is not true in practice (otherwise, we need to re-execute the algorithm for several different m values).

Fourth, another drawback of unsupervised methods for subsequence anomaly detection is the non-stationarity of data series: the data characteristics (e.g., basic statistics and trends) may change over time. These situations are hard to handle and confuse the *discord* and m^{th}-*discord* methods, since an anomalous subsequence may find a very near neighbor among the subsequences of a latter part of the series that involves a different set of normal (and anomalous) patterns.

In this thesis, we address the aforementioned problems, and propose two novel methods suitable for subsequence anomaly detection. The proposed approaches allow us to detect anomalies based on their (dis)similarity to a model representing normal (expected) behavior. These two methods are briefly described in the following two sections, and detailed presentations can be found in Chapter 3.

1.3.1.2 NormA

First, we propose NormA, an unsupervised set-based approach. NormA starts by carefully selecting some of the subsequences of the dataset based on a scoring mechanism. The selected set of subsequences are then used to build the normal behavior model, which is a set of sequences. This process is automatic (using the minimum description length principle), without the need for user intervention, and is effective even when the dataset contains multiple anomalies. We also propose a variant of NormA that is able to handle situations where a single series exhibits multiple normal behaviors. The latter is essential in practice, e.g., when the underlying data generation process changes among several normal states. In the end, NormA detects subsequence anomalies by comparing candidate subsequences to this normal behavior model. We note that NormA is unsupervised and computes the normal behavior model based on the original (unlabeled) dataset, despite the presence of anomalies in it. Our contributions can be summarized as follows.

- We summarize in Section 3.1, the state-of-the-art methods on subsequence anomaly detection and discuss their practical shortcomings. We propose a new definition of subsequence anomalies to overcome these problems, based on the distance to normal behavior.
- In Section 3.3.1, we formalize the concept of *Normal Model*, which is a set of data series that represents the recurrent (normal) behavior in a sequence. The *Normal Model* can be the basis for anomaly detection and can be instantiated in different ways.
- In Section 3.3.2, we describe a new subsequence anomaly detection algorithm that automatically constructs the *Normal Model* series, based on the principles of *frequency*, *coverage* and *centrality*. Subsequently, the algorithm uses the *Normal Model* in order to identify anomalies in an unsupervised and domain-agnostic manner. We propose two variants of this algorithm: NormA-SJ that is based on full computation, and NormA-smpl, based on sampling, which achieves almost the same accuracy but is considerably faster.
- Finally, in Section 3.5, we conduct a large experimental evaluation using several large and diverse datasets from various domains that demonstrates the strong accuracy and the efficiency of NormA.

1.3.1.3 Series2Graph

Second, we now present Series2Graph, a graph-based subsequence anomaly detection method for data series. Our approach does not need labeled instances (like supervised techniques do), or clean data that do not contain anomalies (like zero-positive learning techniques require). It also allows the same model to be used for the detection of anomalies of different lengths.

Series2Graph is based on a graph representation of a novel low-dimensionality embedding of subsequences. It starts by embedding subsequences into a vector space, where

information related to their shapes is preserved. This space is then used to extract overlapping trajectories that correspond to recurrent patterns in the data series. Subsequently, we construct a graph with nodes derived from the overlapping trajectories, and edges represent transitions (among subsequences in different nodes) in the original series.

Intuitively, this graph encodes all the subsequences of a (single or collection of) data series and encodes the recurring patterns in these subsequences. The latter allows us to differentiate between normal behavior, i.e., frequently occurring patterns, and anomalies, i.e., subsequences that rarely occur in the data series.

Our contributions are the following.

- In Section 3.1, we propose a new formalization for the subsequence anomaly detection problem, which overcomes the shortcomings of existing approaches. Our formalization is based on the intuitive idea that anomalies are the subsequences that are not similar to the common behavior, which we call normal.
- In Section 3.4.1, we describe a novel low-dimensionality embedding for subsequences and use a graph representation for these embeddings. This representation leads to a natural distinction between recurring subsequences that constitute normal behavior and rarely occurring subsequences that correspond to anomalies.
- Based on this representation, in Section 3.4.2, we develop Series2Graph, an unsupervised method for domain agnostic subsequence anomaly detection. Series2Graph supports the identification of previously unseen single and recurring anomalies and can be used to find anomalies of different lengths.
- Finally, in Section 3.5, we conduct an extensive evaluation using several large and diverse datasets from various domains that demonstrates the effectiveness and efficiency of Series2Graph.

1.3.2 Streaming Unsupervised subsequence anomaly detection

We now describe the contributions related to the streaming unsupervised subsequence anomaly detection task. As mentioned in the previous section, anomaly, or outlier detection is a well-studied problem [8, 107, 124] relevant to several scientific domains [91]. For the specific case of sequences and data series, we are also interested in identifying *anomalous subsequences*, where outliers are not single values but rather a sequence of values. Moreover, data measurements arriving continuously in several real-world cases require anomaly detection to take place in real-time. Because drifts in data distribution are common, the detection needs to be independent of these changes. To illustrate this point, let us consider a sensor measuring the acceleration on the x-axis of a device positioned on the chest of a human performing different actions [64]. One can observe that the data characteristics corresponding to subsequences are different for actions such as Nordic walking and rope jumping. As changes of actions happen in real-time, the detection of abnormal subsequences needs to be able to adapt to changes in the data generation process.

1.3.2.1 Limitations of Current Approaches

In addition to the limitations enumerated in Section 1.3.1.1, the specific case of streaming data series arises new limitations. Among all previous methods for subsequence anomaly detection, only discords methods (such as Matrix Profile incremental implementation [124]) and tree-based methods [79] can be used. The remaining methods (including our proposed approaches introduced in Section 1.3.1.2 and Section 1.3.1.3) cannot adapt to changes and learn new data characteristics, both of which are required when dealing with data streams due to their design. In such cases, the methods need to learn and modify their parameters as new data arrive.

1.3.2.2 SAND

To address the problems mentioned above we propose SAND, a novel approach suitable for subsequence anomaly detection in data streams. SAND builds a data set of subsequences representing the different behaviors of the data series. These subsequences are weighted using statistical characteristics, such as their cardinality (i.e., how many times the subsequence occurred) and their temporality (i.e., the time difference this subsequence has been detected for the last time). SAND enables this data structure to be updated from one batch to another while computing an anomaly score at every timestamp. Thus, SAND proposes a solution to the subsequences anomaly detection task on streaming data. SAND benefits from k-Shape [93], a state-of-the-art data-series clustering method, which we extend to enable the clustering result to be updated without storing any of the previous subsequences. We demonstrate experimentally that our method outperforms the current (static and streaming) state-of-the-art approaches. Our contributions, described in Chapter 4, are as follows.

- We describe the concepts and ideas used by the state-of-the-art methods on subsequence anomaly detection (static and streaming) and discuss their practical shortcomings.
- We extend k-Shape for streaming scenarios by enabling batch updates of the clustering partition. Our approach avoids entirely the storage of the previous subsequences, a critical step for operating over unbounded data series.
- We present SAND, our subsequence anomaly detection method specifically designed for operation over streaming sequence data. SAND exploits our streaming k-Shape to scale in memory and in execution time for unbounded streams. Furthermore, we propose a new weighting scheme for clusters and an automatic cluster creation procedure to handle distribution drifts. We finally propose a novel anomaly score computation that adapts dynamically to the current batch and gives less importance to old subsequences.
- We perform an experimental analysis using a large corpus of real datasets from different domains (including a ground truth of annotated anomalies). We evaluate both

subtle changes in data characteristics (by concatenating datasets from the same domain) and drastic changes (by concatenating datasets from different domains). We empirically evaluate the influence of SAND's parameters on accuracy and execution time. Finally, we compare SAND with several SOTA approaches and show that our method outperforms the strongest competitor up to 27% while executing one order of magnitude faster.

1.3.3 Supervised identification of anomaly precursors

We now describe the contributions related to the supervised identification of anomaly precursors task. We treat the latter task as a classification task in which the anomaly is one specific class (the precursors are thus discriminant features of the anomaly class).

More generally, data series classification is a crucial and challenging problem in data science [119, 33]. To solve this issue, various data series classification algorithms have been proposed in the past few years [5], applied on a large number of use cases. Standard data series classification methods are based on distances to the instances' nearest neighbors, with k-NN classification (using the Euclidean or Dynamic Time Warping (DTW) distances) being a popular baseline method [30]. Nevertheless, recent works have shown that ensemble methods using more advanced classifiers achieve better performance [6, 73]. Following recent breakthroughs in the computer vision community, new studies successfully propose deep learning methods for data series classification [34, 28, 127, 126, 42, 112, 25], such as Convolutional Neural Network (CNN), Residual Neural Network (ResNet) [114], and InceptionTime [52].

While having a trained and accurate classification model, finding explanations of the classification result (i.e., finding the discriminative features that made the model decide which class to attribute to each instance) is a challenging but important problem, e.g., in manufacturing for anomaly-based predictive maintenance [125], or in medicine for robot-assisted surgeon training [51]. Such discriminant features can be based on patterns of interest that occur in a subset of dimensions at different timestamps or at the same timestamp. For some CNN-based models, the Class Activation Map (CAM) [128] can be used as an explanation for the classification result. CAM has been used for highlighting the parts of an image that contribute the most to a given class prediction and has also been adapted to data series [34, 114].

1.3.3.1 Limitations of Current Approaches

Nevertheless, CAM for data series suffers from one important limitation. Since CAM is a univariate time series (of the same length as the input instances) with high values aligned with the subsequences of the input that contribute the most for a given class identification, in the specific case of multivariate data series as input, no information can be retrieved from CAM on the level of contribution of specific dimensions. Addressing this significant limitation is a sought-after challenge for several domains. For example, in a nuclear power plant, we would like to determine both timestamp and sensors (i.e., the exact patterns) that

could lead to a potential anomaly.

1.3.3.2 dCAM

In this thesis (see Chapter 5), we present a novel approach that fills in the gap by addressing this limitation. We propose a novel data organization and a new Class Activation Map that highlights both the *temporal and dimensional* informations. Our contributions are as follows.

- We first develop a new method that transforms convolutional-based neural network architectures: whereas previous network architectures can only provide an explanation for all the dimensions together, our transformation represents the only deep learning solution that enables explanation in individual dimensions. Our approach can be used with any deep network architecture with a Global Average Pooling layer.

- We demonstrate how we can apply the method to three modern deep learning classification architectures. We first describe dCNN (dimension-wise Convolutional Neural Network), inspired from a traditional CNN (Convolutional Neural Network) architecture. We then describe how more advanced architectures, such as ResNet and InceptionTime, can be transformed as well. We name these transformed architectures dResNet and dInceptionTime.

- We introduce dCAM (dimension-wise Class Activation Map), a novel method that (based on dCNN/dResNet/dInceptionTime) returns a multivariate CAM (Class Activation Map), which identifies the important parts of the input series for *each* dimension.

- We experimentally demonstrate with several synthetic and real datasets that our dimension-wise Class Activation Map is not only more accurate (in classification) than previous approaches but the only viable solution for discriminant feature discovery and classification explanation in multivariate time series using Class Activation Map-based approaches.

1.3.4 Pump Vibration Case

We finally illustrate the applicability and the interest of our proposed methods in a real-world industrial application. We study the detection precursors of unwanted vibrations occurring in turbine-driven feed-water pump systems inside EDF nuclear power plants. We first describe the multivariate data series dataset and the industrial context. Next, we describe the label used to collect the timestamp of unwanted vibrations. We finally describe the sensors chosen for this use case. We then explore the following two scenarios.

1.3.4.1 Unsupervised Detection of Vibrations

We first tackle the task as if the experts had provided no label and apply NormA and Series2Graph. Then, we run in an unsupervised manner the two methods on the two classes (i.e., multivariate data series containing a vibration or not). We then compute an anomaly score for both classes. We finally evaluate their accuracy using the labels of the experts.

1.3.4.2 Supervised Detection of Vibrations and Identification of Precursors

We then tackle the task as a supervised problem (using the label provided by the experts) and apply dCNN/dCAM. We first compare the accuracy of both unsupervised and supervised methods. For that matter, we compare the structural and temporal consistencies of the three methods. We then investigate the precursors detected by dCAM. We depict the recurrent subsequences that have been identified as discriminant by dCAM for the most relevant sensors. We discuss the validity of these results compared to expert knowledge.

1.4 Thesis Outline

In **Chapter 2**, we first introduce basic definitions, taxonomy, and context related to data series and anomaly detection methods (from Section 2.1 to Section 2.3). We then define the different tasks related to anomaly detection in data series (Section 2.4). We finally review state-of-the-art approaches and their related limitations to solve both unsupervised (Section 2.5), semi-supervised (Section 2.6.1) and supervised (Section 2.6.2) anomaly detection task in data series.

In **Chapter 3**, we discuss the unsupervised subsequence anomaly detection task over static data series. We first briefly discuss the limitations of current state-of-the-art approaches. We then introduce two new approaches suitable for domain-agnostic anomaly detection. We first describe NormA (Section 3.3), a novel approach based on a new data series primitive, which permits to detect anomalies based on their (dis)similarity to a model that represents normal behavior. We then introduce Series2Graph (Section 3.4), a graph-based approach that aims to embed the data series into a directed graph that emphasizes the unusual and potentially abnormal subsequences. We then present an experimental analysis (Section 3.5) comparing both NormA and Series2Graph with current state-of-the-art approaches with regards to the anomaly detection accuracy and the execution time.

In **Chapter 4**, we discuss the unsupervised subsequence anomaly detection task over streaming data series. We first present the limitation that current state-of-the-art approaches for static data series can have when applied to data streams. We then introduce SAND (Section 4.2), a novel online method suitable for domain-agnostic anomaly detection. The latter relies on a novel streaming methodology to incrementally update its model, which adapts to distribution drifts and omits obsolete data. We finally present an experimental analysis (Section 4.3) that compares SAND to both static and streaming state-of-the-art approaches over a large corpus of datasets containing both data series with and without variation of normal behaviors.

In **Chapter 5**, we discuss the supervised identification of precursors task. This task can be solved by first considering the problem of supervised classification of data series, and then analyzing the problem of classification explanation. In this chapter, we focus on the classification explanation problem. We first briefly discuss the limitation of current classification explanation approaches. We then introduce dCNN/dCAM (Section 5.3), a new deep convolutional architecture and a new dimension-wise Class Activation Map. Contrary to the usual Class Activation Map, dCAM can identify discriminant features (or subsequences) across every dimension. We then present an experimental analysis (Section 5.4.1) in which we compare our approach with the state-of-the-art techniques.

In **Chapter 6**, we illustrate the applicability and the interest of our developed methods in a real-world application. We study the detection precursors of unwanted vibration occurring in turbine-driven feed-water pump systems inside EDF nuclear power plants. We first describe the dataset and the industrial context. We then explore two scenarios: we first tackle the task as if the experts had provided no label and apply NormA and Series2Graph. We evaluate their accuracy using the labels of the experts. We then tackle the task as a supervised problem (using the label provided by the experts) and apply dCNN/dCAM.

1.5 List of publications

Below we list the publications relevant to this thesis. The publications are sorted by publication dates (from the soonest to the oldest) and grouped by relevant chapter.

Chapter 3 Publications

- Paul Boniol, Themis Palpanas, Michele Linardi, Federico Roncallo, Mohammed Meftah, Emmanuel Remy: Unsupervised and Scalable Subsequence Anomaly Detection in Large Data Series, VLDBJ (2021)
- Paul Boniol, Themis Palpanas: Subsequence Anomaly Detection with Series2Graph, BDA, Paris, France (2020)
- Paul Boniol: Unsupervised Subsequence Anomaly Detection in Large Sequences, Ph.D. Workshop VLDB, Tokyo, Japan (2020)
- Paul Boniol, Themis Palpanas: Series2Graph: Graph-based Subsequence Anomaly Detection for Time Series, Proceedings of the VLDB endowment, Tokyo, Japan (2020)
- Paul Boniol, Themis Palpanas, Mohammed Meftah, Emmanuel Remy: GraphAn: Graph-based Subsequence Anomaly Detection, Proceedings of the VLDB endowment, demonstration track, Tokyo, Japan (2020)
- Paul Boniol, Michele Linardi, Federico Roncallo, Themis Palpanas: Automated Anomaly Detection in Large Sequences, IEEE ICDE, Dallas, USA

(2020)

- Paul Boniol, Michele Linardi, Federico Roncallo, Themis Palpanas: SAD: An Unsupervised System for Subsequence Anomaly Detection, IEEE ICDE, demonstration track, Dallas, USA (2020)

Chapter 4 Publications

- Paul Boniol, John Paparrizos, Themis Palpanas, Michael J. Franklin: SAND: Streaming Subsequence Anomaly Detection, Proceedings of the VLDB endowment, Copenhagen, Danemark (2021)
- Paul Boniol, John Paparrizos, Themis Palpanas, Michael J. Franklin: SAND in action: subsequence Anomaly Detection for Stream, Proceedings of the VLDB endowment, demonstration track, Copenhagen, Danemark (2021)

Chapter 5 Publications

- Paul Boniol, Mohammed Meftah, Emmanuel Remy, Themis Palpanas: dCAM: Dimension-wise Class Activation Map for Explaining Multivariate Data Series Classification **[Submitted for publication]**

Chapter 6 Publications

- Paul Boniol, Mohammed Meftah, Emmanuel Remy, Bruno Didier, Themis Palpanas: dCNN/dCAM: Anomaly Precursors Discovery in multivariate Time series with Deep Convolutional Neural Network **[Under submission]**

Chapter 2

Background and Related Work

Chapter Outline and Abstract

2.1 Anomaly Detection Primer . . . 17
2.1.1 Point versus Subsequences . . . 17
2.2 Preliminaries on Data Series . . . 17
2.2.1 *Univariate* versus *Multivariate* . . . 17
2.2.2 *Static* versus *Streaming* . . . 18
2.2.3 *Discrete* versus *Continous* . . . 18
2.2.4 Real-World constraints: *Missing* points and *non-synchronized* data series . . . 19
2.2.5 Data Series Notations . . . 19
2.3 Preliminaries on Anomaly Detection Methods . . . 19
2.3.1 *Unsupervised*: No prior information . . . 20
2.3.2 *Supervised*: prior information . . . 20
2.4 Problems Formulation . . . 20
2.4.1 Unsupervised Subsequence Anomaly Detection . . . 21
2.4.2 Supervised Detection of Anomaly: Identification of their precursors . . . 22
2.5 Unsupervised Subsequences Anomaly Detection in Data Series . . . 24
2.5.1 Data Series Distances Definitions . . . 24
2.5.2 Data-Series Clustering . . . 26
2.5.3 Density based Approaches . . . 27
2.5.4 Subsequence Similarity-based Approaches . . . 30
2.5.5 New perspectives with graph-based approaches . . . 35
2.6 Supervised Subsequences Anomaly Detection in Data Series . . . 39
2.6.1 Semi-Supervised Machine Learning Approaches . . . 40
2.6.2 Supervised Machine Learning Approaches . . . 46
2.6.3 Finding Precursors of Abnormal Subsequences . . . 51
2.7 Summary . . . 53

In this chapter, we define the relevant taxonomy related to data series and anomaly detection methods. We then formally define the problem of subsequence anomaly detection in data series tackled in this manuscript. We finally study state-of-the-art methods to perform unsupervised anomaly detection and supervised identification of anomaly precursors in data series. Thus, the chapter is structured as follows. We first underline the preliminary definitions and the context of anomaly detection in data series. We describe the fundamental differences between point and subsequence anomalies and argue our choice of subsequence anomalies as our primary focus. We then formally describe the problem of subsequence anomaly detection in data series. As previously outlined, we inferred two variants of the problem. On the one hand, the anomalies are unknown, and a fully unsupervised framework has to be adopted. On the other hand, anomalies are known, and precursors of these anomalies have to be found. We finally enumerate state-of-the-art methods that could solve the problems mentioned above.

2.1 Anomaly Detection Primer

First of all, one should note that no single, universal, precise definition of outliers or anomalies exists. In general, an anomaly is an observation that appears to deviate markedly from other members of the sample in which it occurs. This fact may raise suspicions that the specific observation was generated by a different mechanism than the rest of the data (Hawkins definition [47]). This mechanism may be an erroneous procedure of data measurement and collection or an inherent variability in the domain of the data under inspection. Nevertheless, such observations are interesting in both cases, and the analyst would like to know about them. The above general definition can then take different forms, according to the specific problem and data characteristics. For example, in statistics, anomalies can be termed the data points that are further than three standard deviations away from the mean of known data distribution. This case implies that we need to perform extensive tests to find the distribution (and its parameters) that best fits the dataset. Nevertheless, in several real-world problems, we do not know precisely the data distribution. Consequently, defining and identifying anomalies using their distance from a mean value defined by experts is sometimes hardly practical.

2.1.1 Point versus Subsequences

At this point, we are interested in finding anomalies in data series. This goal can be tackled by either looking at single values or a sequence of points. In the specific context of points, we are interested in finding points that are far from the normal distribution of values that correspond to *healthy* states. In the specific context of sequences, we are interested in identifying anomalous subsequences that are, unlike an outlier, not a single abnormal value but rather an abnormal evolution of values. In real-world applications, this distinction becomes crucial for the following reason: even though every point individually picked seems normal, the shape generated by the sequence of these same values may be anomalous and could lead to major issues that would have been undetected until too late. Figure 2.1 underlines the aforementioned distinction between point and sequence outlier. In this work, we will study the specific case of subsequence anomaly detection in data series.

2.2 Preliminaries on Data Series

We are interested in detecting abnormal subsequences in data series. For instance, such data series can correspond to the evolution of points of one or several sensors on a fixed or infinite period. We now define the different categories of data series type as follows.

2.2.1 *Univariate* versus *Multivariate*

We define a *univariate* data series as an ordered sequence of real values on a single dimension. For instance, a *univariate* data series corresponds to the history of values of one

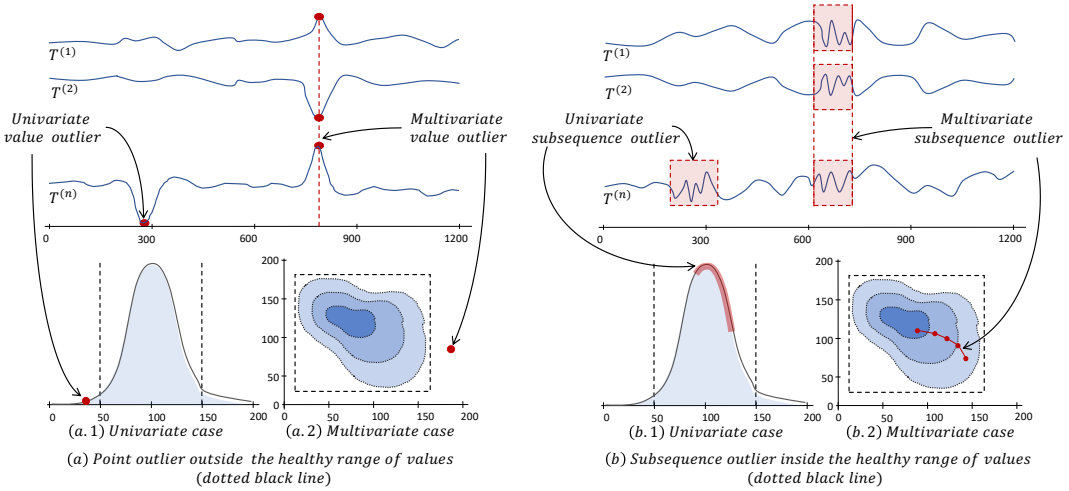


Figure 2.1: Synthetic examples where (a) a point outlier can be detected by looking at its position in the distribution (for both univariate (a.1) and multivariate (a.2)). (b) a sequence outlier is taking part of the *normal* range of values (by taking every point independently without any temporal structure for both univariate (b.1) and multivariate (b.2)).

given sensor. In this case, a subsequence can be represented as a vector. Then, we define a *multivariate* data series as either a set of ordered sequences of real values (with each ordered sequence having the same length) or an ordered sequence of vectors composed of real values. For instance, a *multivariate* data series can be a set of sensors installed on the same system. In this specific case, a subsequence is a matrix in which each line corresponds to a subsequence of one single dimension.

2.2.2 *Static* versus *Streaming*

We now have defined the difference between univariate and multivariate data series. Nevertheless, for data series ordered by time, a core characteristic is their evolution with time. Therefore, we define *static* data series as sequences that have a fixed length. In this case, one does not expect any more values to be added and can analyze points or subsequences all at once. For instance, analyzing a fixed period before a collection of events will consist in analyzing a collection of *static* data series. On the opposite case, we define *streaming* data series as sequences with an infinite length with new points or subsequences arriving with a given acquisition rate (not always constant in time). In this case, one model needs to be updatable dynamically as new points are arriving. For instance, monitoring sensors and detecting abnormal subsequences in real-time requires analyzing *streaming* data series.

2.2.3 *Discrete* versus *Continous*

We define a *Discrete* data series as a sequence of points with categorical values (ordered or not). On the contrary, *Continous* data series are sequence of points with real values.

For instance, boolean sensors (that returns only zero or one as possible values) generate $Discrete$ data series, and temperature sensors usually return $Continous$ data series. The latter two different types of data series have to be handled with different type of methods. In this thesis, we analyze methods that deal with $Continous$ data series.

2.2.4 Real-World constraints: $Missing$ points and $non\text{-}synchronized$ data series

Real-world constraints implied by the data acquisition step can make the data series harder to analyze. The first constraint is related to missing data points. Such constraint might happen because of sensors issues returning wrong values or a specific acquisition protocol (for instance, some sensors might return a value only when the measured value changes). It results in data series with missing data points that need to be filled. The second constraint is related to non-synchronized multivariate data series. It is implied by the difference in the acquisition rate of the different sensors. In this case, one has to choose a fixed acquisition rate and then either downsampling (i.e., losing data points and potential accuracy) or upsampling (i.e., creating a missing data points constraint) the data series with a different acquisition rate. These two constraints are serious and typical for many real-world use cases. Nevertheless, most of the subsequence anomaly detection methods proposed in the literature do not tackle them.

2.2.5 Data Series Notations

We now introduce some formal notations related to data series. We define a data series $T \in \mathbb{R}^n$ as a sequence of real-valued numbers $T_i \in \mathbb{R}$ $[T_0, T_2, ..., T_{n-1}]$, where $n = |T|$ is the length of T, and T_i is the i^{th} point of T.

A multivariate, or D-dimensional data series $\mathbf{T} \in \mathbb{R}^{(D,n)}$ is a set of D univariate data series of length n. We note $\mathbf{T} = [T^{(0)}, ..., T^{(D-1)}]$ and for $j \in [0, D]$, we note the univariate data series $T^{(j)} = [T_0^{(j)}, T_1^{(j)}, ..., T_{n-1}^{(j)}]$. A subsequence $T_{i,\ell}^{(j)} \in \mathbb{R}^\ell$ of the dimension $T^{(j)}$ of the multivariate data series T is a subset of contiguous values from $T^{(j)}$ of length ℓ (usually $\ell \ll n$) starting at position i; formally, $T_{i,\ell}^{(j)} = [T_i^{(j)}, T_{i+1}^{(j)}, ..., T_{i+\ell-1}^{(j)}]$. The multivariate subsequence is defined as $T_{i,\ell} = [T_{i,\ell}^{(0)}, ..., T_{i,\ell}^{(D)}]$. For a given univariate data series T, the set of all subsequences in T of length ℓ is defined as $\mathbb{T}_\ell = \{T_{0,\ell}, T_{1,\ell}, ..., T_{|T|-\ell,\ell}\}$.

2.3 Preliminaries on Anomaly Detection Methods

In the previous section, we have defined the input type characteristics. We now define categories and taxonomy of methods commonly used in the literature [10, 23]. We first define method categories based on the output type returned. We then categorize methods based on the input required.

2.3.1 *Unsupervised*: No prior information

We define *Unsupervised* methods as methods that only require the subsequences as input and do not need any annotations as prior information. Such methods suit well the case of anomaly discovery, visualization, and automatic annotation. Nevertheless, one should note that such methods are, in general, less accurate than supervised and semi-supervised methods.

2.3.2 *Supervised*: prior information

2.3.2.1 *Semi supervised*: Only normal subsequences annotated

We define *Semi supervised* methods as methods that require annotated normal subsequences to learn to detect abnormal subsequences. The latter is the classical case for most of the anomaly detection methods in the literature. One should note that this category is often defined as *Unsupervised*. However, we consider it unfair to group such methods with the category mentioned above, knowing that they require much more prior knowledge than the previous one.

2.3.2.2 *fully Supervised*: Both abnormal and normal subsequences annotated

We finally define *Supervised* methods as methods that need both normal and abnormal subsequences annotations to learn to discriminate them together. Hence, such methods can be seen as classifiers.

In addition to that, one should note that any parameter can be tuned to fit a specific objective. For example, if labels of normal or abnormal subsequences are provided, one can tune parameters of unsupervised methods to maximize the objective of detecting the provided abnormal labels (or exclude the provided normal labels). Any unsupervised method can be supervised. Nevertheless, the opposite is not valid. We formally define in the coming sections the issues associated with these categories in the paradigm of anomaly detection in data series.

2.4 Problems Formulation

As mentioned in the previous section, we are interested in finding abnormal subsequences in the data series. This task can be divided into two cases: (i) knowledge experts do not have precise (or not at all) information on what anomalies to detect. (ii) knowledge experts have specific examples of which anomalies they have to detect (and have a collection of known anomalies).

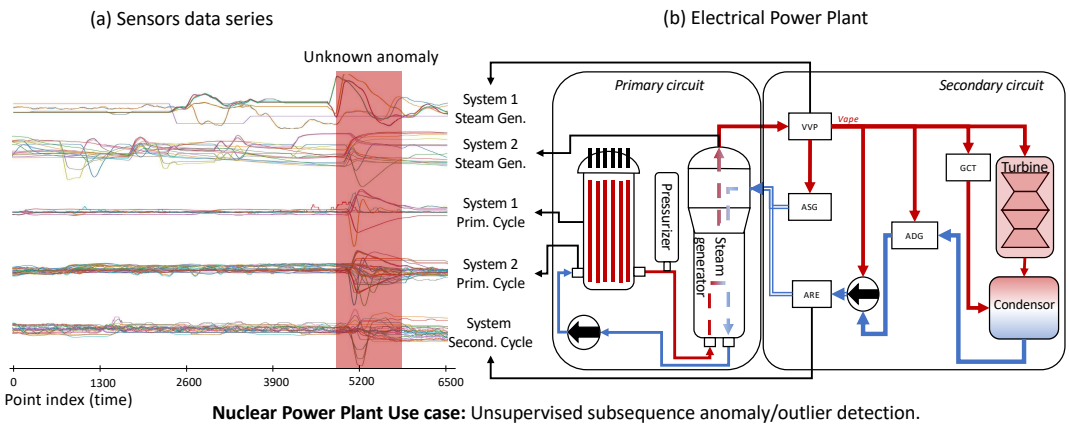


Figure 2.2: Unsupervised subsequence anomaly detection task illustrated using a nuclear power plant use case, in which (a) illustrates the sensors data series coming from (b) subsystems of a given electrical power plant. The goal is to find unknown abnormal subsequences in the sensors data series (illustrated by the red rectangle).

2.4.1 Unsupervised Subsequence Anomaly Detection

In the first case (i), one can decide to adopt a fully unsupervised method. These methods benefit from working without needing a large collection of known anomalies and automatically detecting unknown abnormal behaviors. Such methods can be used either to monitor the health state of a system or to mine the historical data series of a system (to build a collection of abnormal behaviors that can then be used on a supervised framework). Figure 2.2 illustrates the aforementioned task using an electrical power plant use case, in which Figure 2.2(a) illustrates the sensors data series coming from Figure 2.2(b) subsystems of a given electrical power plant. The goal is to find unknown abnormal subsequences in the sensors data series (illustrated by the red rectangle in Figure 2.2(a)). We now propose a formal definition of the above problem:

Definition 1 (Unsupervised Detection Problem Definition)**.** *Given a monitored system $\mathcal{M}$, and a set of data series $T_{\mathcal{M}}$ that represents the state of $\mathcal{M}$, the function f that takes as inputs $T_{\mathcal{M}}$ and returns $s \in \{\mathcal{N}, \mathcal{A}\}$ has to be found (where $\mathcal{N}$ indicates that the data series corresponds to a healthy state, whereas $\mathcal{A}$ indicates that the data series corresponds to an anomaly). Thus, f can be written as follows:*

$$f : T_{\mathcal{M}} \rightarrow \{\mathcal{N}, \mathcal{A}\}$$

This definition has the benefit of being simple. Nevertheless, it has to be joined with several significant constraints, listed as follows:

• **Abnormality Formalization Constraint**: The $\mathcal{N}$ and $\mathcal{A}$ categorization is most of the time difficult to infer. Thus, a different variant of this problem can be considered. For instance, a level of $abnormality$ could be defined, which brings function f to return a

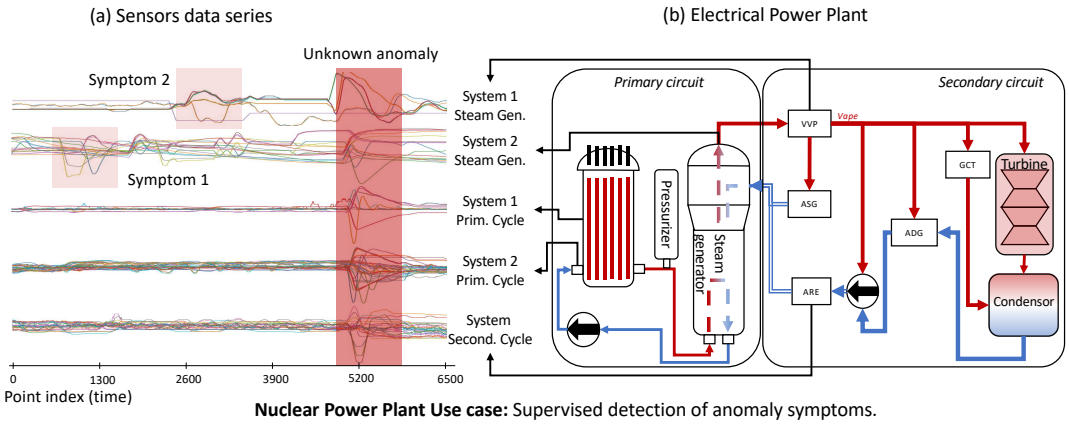


Figure 2.3: Supervised detection of anomaly precursors task illustrated using a nuclear power plant use case, in which (a) illustrates the sensors data series coming from (b) subsystems of a given electrical power plant. The goal is to find, from the known abnormal subsequences in the sensors data series (illustrated by the red rectangle), the potential precursors of this anomaly (illustrated by the light red rectangles).

value p which should be proportional to the degree of $abnormality$, or can be seen as $p = P(s = \mathcal{A})$.

• **Dataset Size Constraint**: Given the fact that the size of the datasets to analyze has increased significantly for the last decades, and the detection should be as fast as possible to let enough time to experts to fix the possible issues, the problem must be achieved using algorithms that scale as good as possible with the dataset size.

• **Subsequence Length Constraint**: As we are focusing on subsequences anomaly detection, a subsequence length has to be chosen. It can be chosen either by the user or automatically by the algorithm. In practice, it is hard to compare and detect both anomalies of very different lengths. Unfortunately, we notice that fixing the subsequence length automatically is impossible. Therefore, every method analyzed in this study will require a user parameter subsequence length of the anomalies to detect.

• **Streaming Constraint**: Data measurements are arriving continuously in several real-world cases. Thus, it requires anomaly detection to take place in real-time. Furthermore, because drifts in data distribution are common, the detection needs to be independent of these changes. Thus function f needs to be updated in real-time.

2.4.2 Supervised Detection of Anomaly: Identification of their precursors

In the first case (i) described in the previous section, anomalies were considered unknown. In the second case (ii), we consider that the expert precisely knows which event he wants to detect and has a collection of data series corresponding to these anomalies. In that case, we

have a database of anomalies at our disposal. As a consequence, one can decide to adopt supervised methods. A question that naturally arises is the following one: is it possible to detect subsequences that happened before the known anomaly that might lead to an explanation of the anomaly (and potentially predict it). Such subsequences can be called *precursors* of anomalies.

Detecting such subsequences might be significantly valuable for knowledge experts to prevent future anomalies from occurring or understanding why an anomaly occurred (or facilitate its understanding). Thus, the task is to detect, in a supervised manner, known anomalies and retrieve potential symptoms. Figure 2.3 illustrates the aforementioned task using an electrical power plant use case, in which Figure 2.3(a) illustrates the sensors data series coming from Figure 2.3(b) subsystems of a given nuclear power plant. The goal is to find unknown precursors (T_1 and T_2) of abnormal subsequences in the sensors data series (illustrated by the red rectangle in Figure 2.3(a)). We now propose a formal definition of the problem mentioned above:

Definition 2 (Precursors Detection Problem Definition)**.** *Given a monitored system $\mathcal{M}$, a set of data series $T^{\mathcal{N}}_{\mathcal{M}}$ that represents the healthy state of $\mathcal{M}$ (healthy state labeled $\mathcal{N}$), a set of data series $T^{\mathcal{A}}_{\mathcal{M}}$ that represents the state of $\mathcal{M}$ before an anomalous state (anomalous state labeled $\mathcal{A}$), we first have to find function f that takes as input $T^{\mathcal{A}}_{\mathcal{M}}$ and $T^{\mathcal{N}}_{\mathcal{M}}$, and returns $s \in \{\mathcal{N}, \mathcal{A}\}$. We then have to find function g that takes as input $T^{\mathcal{A}}_{\mathcal{M}}$ and f, and returns $S \subset T^{\mathcal{A}}_{\mathcal{M}}$ (S being the ensemble of subsequences in $T^{\mathcal{A}}_{\mathcal{M}}$ precursors of the upcoming anomalies). Formally, f and g can be written as follows:*

$$f : T^{\mathcal{N}}_{\mathcal{M}}, T^{\mathcal{A}}_{\mathcal{M}} \rightarrow \{\mathcal{N}, \mathcal{A}\}$$
$$g : T^{\mathcal{A}}_{\mathcal{M}}, f \rightarrow S$$

As mentioned earlier, such definition has to be joined with several constraints, listed as follows:

• **Precursor Formalization Constraint**: In this case, $\mathcal{N}$ and $\mathcal{A}$ categorization is well defined (labels provided by the user). However, the binary categorization between subsequences belonging to S and others is hard to infer. Thus, a different variant of this problem can be considered. For instance, a $precursor$ level could be defined, which brings function g to return a value p for each subsequence $sq \in T^{\mathcal{A}}_{\mathcal{M}}$ which should be proportional to the $precursor$ degree, or can be seen as $p = P(sq \in S)$.

• **Dataset Size Constraint**: Similarly to the previous dataset size constraint, scalable algorithms are desirable for a very large database. In addition to that, the dataset size has an impact on the accuracy of function f. A big enough database is required to train an accurate function f. An inaccurate function f will lead naturally to an inaccurate function g.

2.5 Unsupervised Subsequences Anomaly Detection in Data Series

Most unsupervised anomaly detection methods rely on similarity (or distance) measures between sequences (or subsequences). The latter is a significant feature to measure the deviation, or the isolation of a sequence compare to the others. We thus start by formally defining the standard distance measures for data series.

2.5.1 Data Series Distances Definitions

Given two sequences (or univariate data series), $A \in \mathbb{R}^\ell$ and $B \in \mathbb{R}^\ell$, of the same length, ℓ, we define the distance between A and B as $d(A, B) \in \mathbb{R}$, such as $d(A, B) = 0$ when A and B are the same. There exist different definitions of d in the literature. We enumerate the usual distance definition in the following sections.

2.5.1.1 Euclidean Distances

The first is to use the classical and widely used euclidian distance defined as follows:

$$ED(A, B) = \sqrt{\sum_{i}^{\ell}(A_{i,1} - B_{i,1})^2}$$

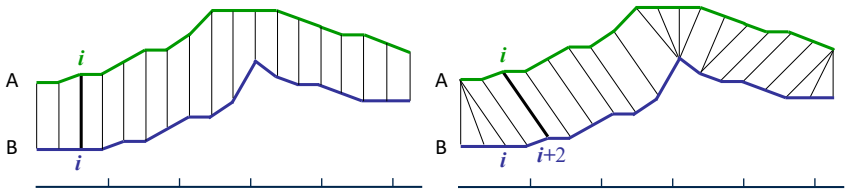
Nevertheless, using that distance, the similarity between subsequences is influenced by their $mean$. Two subsequences with the same shape but a different $mean$ will have a high euclidian distance. To cope that issue, the Z-normalized Euclidean distance [26, 86, 113, 115, 122] can be used:

$$ED_z(A, B) = \sqrt{\sum_{i}^{\ell}\left(\frac{A_{i,1} - \mu_A}{\sigma_A} - \frac{B_{i,1} - \mu_B}{\sigma_B}\right)^2}$$

In the above equation, μ and σ represent the mean and standard deviation of the sequences. Moreover, the complexity is $O(|A|)$ for the two Euclidian distance variants. Nevertheless, Z-normalized distance requires computing the mean and the standard deviation but remains from the same order of magnitude as the classical Euclidian distance.

2.5.1.2 Dynamic Time Wrapping

A major drawback of Euclidean distances is their lack of flexibility to time misalignments. Moreover, previous distances cannot be used for subsequences of different lengths. Thus



(a) Euclidian Distance (b) DTW distance

Figure 2.4: Comparison between the alignment considered by Euclidian distance(a) and DTW distance (b).

the Dynamic Time Wrapping (DTW) is commonly used to cope with these issues. Note that DTW is not a distance as the triangular inequality does not hold. However, in the following section, we will refer to it as DTW distance. Given $i \in [0, m]$ and $j \in [0, n]$ with m and n the length of the sequences A and B, the DTW distance between A and B can be defined as follows:

$$DTW(A, B) = \sqrt{\delta(A_{0,m}, B_{0,n})}$$

with δ recursively defined as follows:

$$\begin{aligned}
\delta(A_{0,0}, B_{0,0}) &= (A_0 - B_0)^2 \\
\delta(A_{0,i}, B_{0,j}) &= (A_i - B_j)^2 + \min \left\{ \begin{smallmatrix} \delta(A_{0,i-1}, B_{0,j}) \\ \delta(A_{0,i-1}, B_{0,j-1}) \\ \delta(A_{0,i}, B_{0,j-1}) \end{smallmatrix} \right\}
\end{aligned}$$

DTW distance corresponds to the distance between A and B considering the best alignment of points. Figure 2.4 depicts the difference between Euclidian and DTW distance for subsequences. However, finding the best alignment has a cost. The complexity in time for DTW distance computation is $O(|A||B|)$, which is one order of magnitude higher than Euclidian distance. This version of DTW is difficult to use for large sequences or a large collection of subsequences in practice. In order to cope with that issue, one can bound the maximal misalignment using a window length w to set a Sakoe-Chuba Band [98]. This window length will only allow alignment (i, j) such that $|i - j| < w$. Thus δ_w is recursively defined as follows:

$$\delta_w(A_{0,i}, B_{0,j}) = (A_i - B_j)^2 + \min \left\{ \begin{smallmatrix} \delta_w(A_{0,max(i-1,j-w)}, B_{0,j}) \\ \delta_w(A_{0,i-1}, B_{0,j-1}) \\ \delta_w(A_{0,i}, B_{0,max(j-1,i-w)}) \end{smallmatrix} \right\}$$

From the latter formulation, the complexity becomes $O(max(|A|, |B|)w)$ and can be nearly linear for small w. However, the accuracy of the metric depends significantly on which w to choose. The other way to cope with the previous issue is to set an Itakura Parallelogram [53] that divides by two the Sakoe-Chuba Band number of possible alignment. Nevertheless, the time complexity remains of the same order of magnitude.

2.5.1.3 Shape Based Distance

Recently proposed for clustering purposes, Shape-Based Distance (SBD) has shown state-of-the-art performance in data series clustering [93, 92]. SBD uses cross-correlation to find the appropriate alignment between two sequences. Formally, given A, B, two sequences of length ℓ, the SBD distance is defined as follows:

$$SBD(A,B) = 1 - max_w\left(\frac{R_{w-\ell}(A,B)}{\sqrt{R_0(A,A).R_0(B,B)}}\right)$$
$$\text{with: } R_k(A,B) = \begin{cases} \sum_{i=1}^{\ell-k} A_{i+k}.B_i, & k \geq 0 \\ R_{-k}(B,A), & k < 0 \end{cases}$$

As the cross-correlation is in practice computed using Fast Fourier Transform, the SBD distance complexity is $O(max(|A|,|B|) * log(max(|A|,|B|)))$. Thus for equal length sequences, SBD complexity ($O(|A| * log(|A|))$) is expensive but can be a decent compromise between the linear complexity of ED ($O(|A|)$) and the quadratic complexity of DTW ($O(|A|^2)$).

2.5.2 Data-Series Clustering

Despite the proliferation of anomaly detection methods, a set of effective methods are those that either determine anomalies by comparing subsequences to previous subsequences or with an estimated normal behavior.

Clustering can summarize the underlying patterns in data and, therefore, can be used to extract the recurring behavior in datasets for anomaly detection purposes. Formally, given a set of observations (or subsequences which are the topic of this paper), clustering methods partition this set into k distinct clusters, such that the within-cluster sum of squared distances is minimized. For a given set of subsequences $\mathbb{T}_\ell$, we note $\mathbb{C} = \{\mathcal{C}_0, ..., \mathcal{C}_k\}$ the optimal partition of k cluster $\mathcal{C}_i$ with $\forall \mathcal{C}_i, \mathcal{C}_j \in \mathcal{C}, \mathcal{C}_i \cap \mathcal{C}_j = \emptyset$. We note $\bar{\mathcal{C}_i}$ the centroid of cluster $\mathcal{C}_i$.

2.5.2.1 k-means

The k-means algorithm solves this partitioning problem using the Z-normalized Euclidean Distance. k-means centroids correspond to the arithmetic mean of the subsequences in their corresponding clusters. Other algorithms based on Euclidean distance have been proposed (such as hierarchical clustering). Moreover, it is easy to extend k-means to a streaming context [43], since the centroids can be incrementally updated as new points arrive.

2.5.2.2 k-Shape

Euclidean distance-based algorithms may miss some properties related to the alignment in data series. Recently, k-Shape (clustering algorithm based on SBD) has shown strong performance in data series clustering [93, 92]. The k-Shape centroid computation corresponds to an optimization problem in which we are computing the minimizer (i.e., sequence) of the sum of squared distances to all other sequences using the SBD distance. Formally, as described in Equation 15 in [93], centroid $\bar{\mathcal{C}_j}$ is computed as follows:

$$\begin{aligned} &\bar{\mathcal{C}_j} \leftarrow \underset{\bar{\mathcal{C}_j}}{argmax} \frac{(\bar{\mathcal{C}_j})^T.M.\bar{\mathcal{C}_j}}{(\bar{\mathcal{C}_j})^T.\bar{\mathcal{C}_j}} \\ &\text{with: } M = Q^T.S_j.Q,\ Q = I - \frac{1}{|\bar{\mathcal{C}_j}|}O, \text{ and } S_j = \sum_{A \in \mathcal{C}_j} A.A^T \end{aligned} \tag{2.1}$$

Note that $\bar{\mathcal{C}_j}$ is considered as a vector ($(\bar{\mathcal{C}_j})^T$ is its transpose) in the above equation. The dot operator is the dot product between two matrices/vectors. Moreover, I is the identity matrix, and O is a matrix with all ones. In practice, the centroid that maximizes Equation 2.1 corresponds to the eigenvector with the largest eigenvalue of the real symmetric matrix M. Moreover, as depicted in Equation 2.1, the centroid computation requires all the sequences $A \in \mathcal{C}_j$ for every cluster $\mathcal{C}_j$ in memory and can be used for non-streaming data series only.

2.5.3 Density based Approaches

A first way to detect anomalies (or unusual subsequences) is to measure their isolation from others. The notion of isolation or neighborhood density has been used in methods unrelated to data series subsequences anomaly detection. These methods, usually called density-based methods, aim to evaluate a subspace of the points (in our specific case, a point is a subsequence) to identify either isolated subsequences or dense neighborhoods. Even though these methods are not dedicated to detecting abnormal subsequences in data series, they still bring interesting insight into abnormality detection and seem to bring accurate results (even for data series).

2.5.3.1 Local Outlier Factor

Local Outlier Factor [20] is a degree of being an outlier assigned to each subsequence (of a given length ℓ). This degree depends on how the instance is isolated (in the distance) to the surrounding neighborhood. First, given a data series T and two subsequences $A, B \in \mathbb{T}_\ell$, k-$d(A)$ is the distance between A and its k^{th} nearest neighbor ($NN_k(A)$ the set of these k nearest neighbors). The Local Outlier Factor (LOF) is based on the following reachability distance definition:

$$rd_k(A, B) = max(k\text{-}d(B), d(A, B))$$

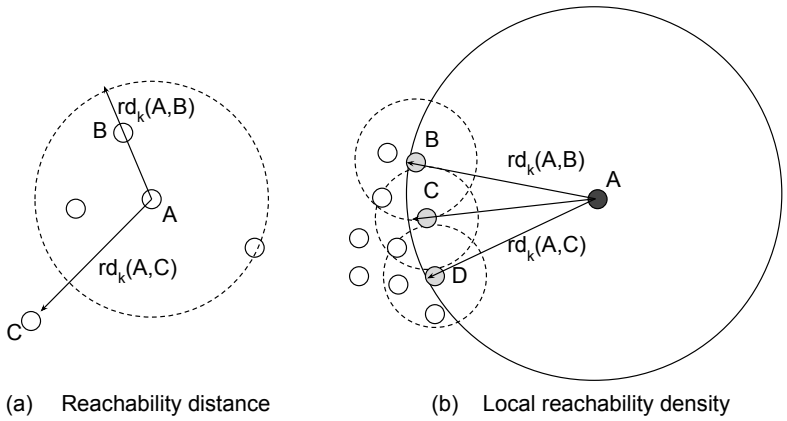


Figure 2.5: (a) For a given data series T, illustration of the reachability distance between A and B and then A and C for $k = 4$ (with $A, B, C, D \in \mathbb{T}_\ell$). (b) Difference between $rd_k(A, X), X \in NN_k(A)$, when A is an anomaly and B, C, D are regular subsequences in data series T.

The central concept behind this distance definition is to stress the homogeneity of distances between instances within the k-neighborhood (i.e., the k-neighborhood will have the same distance between each other). Thus the local reachability can be defined as follows:

$$lrd_k(A) = \frac{|NN_k(A)|}{\sum_{B \in NN_k(A)} rd_k(A, B)}$$

Given a subsequence, $A \in \mathbb{T}_\ell$, $lrd_k(A)$ is the inverse of the averaged reachability of A from its neighborhood, i.e., the averaged distance at which A can be reached from its neighbors. Therefore, LOF is given by:

$$LOF_k(A) = \frac{\sum_{B \in N_k(A)} \frac{lrd_k(B)}{lrd_k(A)}}{|N_k(A)|} = \frac{\frac{\sum_{B \in N_k(A)} lrd_k(B)}{|N_k(A)|}}{lrd_k(A)}$$

LOF_k of an instance is thus the average of the local reachability density of the neighbors divided by its reachability density. Therefore, if we set subsequences of length ℓ as instances for which the ℓ points will be the features, this factor can be used as an anomaly score. However, it highly depends on the parameter k to use. Figure 2.5 illustrate the reachability distance and the local reachability density for $k = 4$. Figure 2.6 depicts LOF_k on the MBA(803) datasets (MIT-BIH Supra-ventricular Arrhythmia Database [40, 83], in which the anomalies are 75 points long). The plot in the middle is LOF_k computed using a small snippet of 10,000 points and different k (from 20 to 80), and the bottom plot using a large snippet of 100,000 points. Logically, the small snippet contains significantly fewer anomalies (3 supra-ventricular contractions) than the large snippet (62 supra-ventricular contractions). Thus, for the small snippet, LOF_k correctly identifies the anomaly for $k > 30$. However, due to many anomalies in the snippet, LOF_k is failing to detect the anomaly regardless of

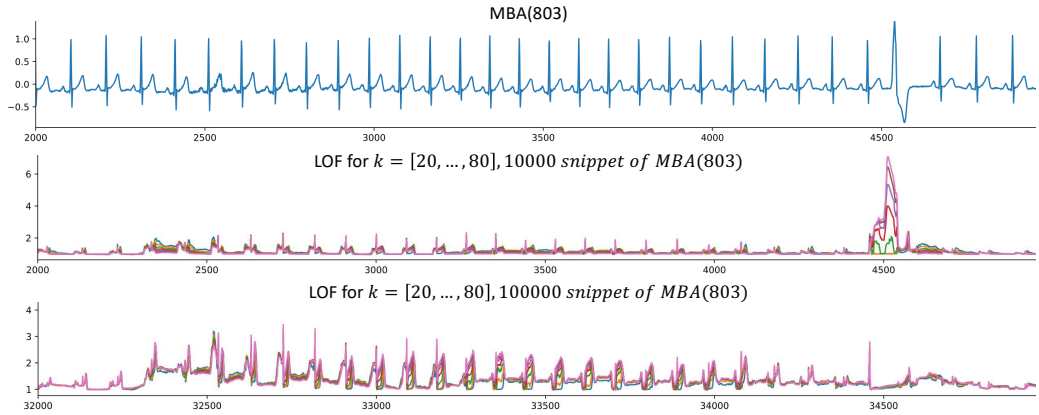


Figure 2.6: LOF_k computed for $k = 20, 30, 40, 50, 60, 70, 80$ using a 10000 snippet (middle plot) and 100000 snippet (bottom plot) of MBA(803). In the big snippet, due to the higher number of anomaly, LOF_k method is not able to detect them.

the k used.

Most of the time, it is nearly impossible to know the exact value of k to use for two reasons: (i) knowing in advance the number of anomalies that compose the data series is, in practice, impossible. (ii) several overlapping subsequences that refer to the same anomaly increase the neighborhood size. Therefore, regarding the length of the anomaly, the value of k can be even larger than the number of anomalies. It is the case in Figure 2.6 (bottom plot) where LOF_{80} is not able to detect all anomalies of the same type that repeats 65 times.

2.5.3.2 Isolation Forest

Isolation Forest is a density-based method that, instead of modeling the dataset's normal distribution, tries to isolate the outlier from the rest [75]. The key idea remains on the fact that, in a normal distribution, anomalies are more likely to be isolated (i.e., requiring fewer random partitions to be isolated) than normal instances.

If we assume that the latter statement is true, one only has to produce a partitioning process indicating the isolation degree (i.e., anomalous degree) of subsequences.

Let first define the concept of Isolation Tree as stated in [75]. Let be Tr a binary where each node has zero or two children and a test which consists of an attribute q and a split p such that it divides data points into the two children based on the condition $p < q$. Tr is built by dividing recursively $\mathbb{T}_\ell$ randomly selecting p and q until the maximal depth of the tree is reached, or the number of different instances is equal to 1. Figure 2.7 depicts an example of isolation trees. Using that kind of data structure, the length of the path into the tree Tr to reach a subsequence $A \in \mathbb{T}_\ell$ is directly correlated to the anomaly degree of that instances. Therefore, we can define the anomaly score as follows:

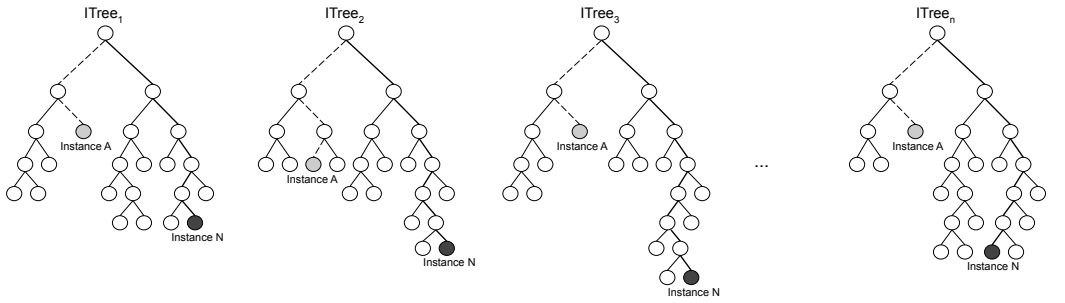


Figure 2.7: Set of isolation trees that randomly partition a dataset. In average, for a given data series T, the subsequence $N \in \mathbb{T}_\ell$ has a longer path to the root than the subsequence $A \in \mathbb{T}_\ell$, thus A anomaly score will be higher.

$$
\begin{aligned}
S(A, n) &= 2^s \\
s &= \frac{\sum_{Tr \in \mathcal{T}} h(A, Tr)}{|\mathcal{T}| c(n)} \\
c(n) &= H(n-1) - \frac{2(n-1)}{n}
\end{aligned}
$$

In the above equation, $h(x, Tr)$ the length of the path to reach x in tree Tr, $\mathcal{T}$ a set of different isolation trees built, n the number of instances in the training set and H the harmonic number which can be estimated using the Euler constant.

2.5.3.3 Extended Methods on Similar concepts

Several studies have extended the concepts proposed by these density-based methods. First, several approaches such as COF [109] have been proposed adding a "connectivity" concept to the existing LOF method. Similarly, ILOF [95] and DILOF [87] have been proposed to enable an incremental computation of LOF (which then can be used for streaming applications). Finally, extensions of Isolation Forest, such as Extended IForest [46], Hybrid IForest [82], and IMondrian Forest [79] have been proposed and brought an extension that (i) improves the accuracy when labels are provided [82], (ii) improves the splitting nodes with linear combinations of the dimensions [46], (iii) changes the structure of the trees such that they can be changed incrementally [79].

2.5.4 Subsequence Similarity-based Approaches

In this section, we highlight the different methods for anomaly detection based on the subsequences similarity principle. The problem addressed in the following work corresponds to the identification of anomalous subsequences (of a given length) within a long data sequence based on their similarity/distance measure.

2.5.4.1 Discord Definition

Similarly for density based techniques (LOF and IF), the state-of-the-art solutions for subsequences anomaly detection use a definition that characterizes the isolation of subsequences. The latter definition, also called $discords$, is defined as follows:

Definition 3 (Discord). *[124, 104, 60, 76, 36, 22, 77] A subsequence $T_{i,\ell}$ is a discord, if the distance between its nearest neighbor (NN), namely $T_{j,\ell}$, is the largest among all the nearest neighbors distances computed between subsequences of length ℓ in T. We require that $T_{j,\ell}$ is not a trivial match of $T_{i,\ell}$.*

The latter is an intuitive definition: a subsequence is a discord if its nearest neighbor (NN) is very far away. However, this definition fails when we have two neighboring discords, with a small distance to each other and a very large distance to all the remaining subsequences. In order to capture these situations, the m^{th}-discord has been proposed:

Definition 4 (m^{th} Discord). *[121] A subsequence $T_{i,\ell}$ is a m^{th}-discord, if the distance between its m^{th} nearest neighbor, namely $T_{i,\ell}$, is the largest among all the m^{th} nearest neighbors distances computed between subsequences of length ℓ in T.*

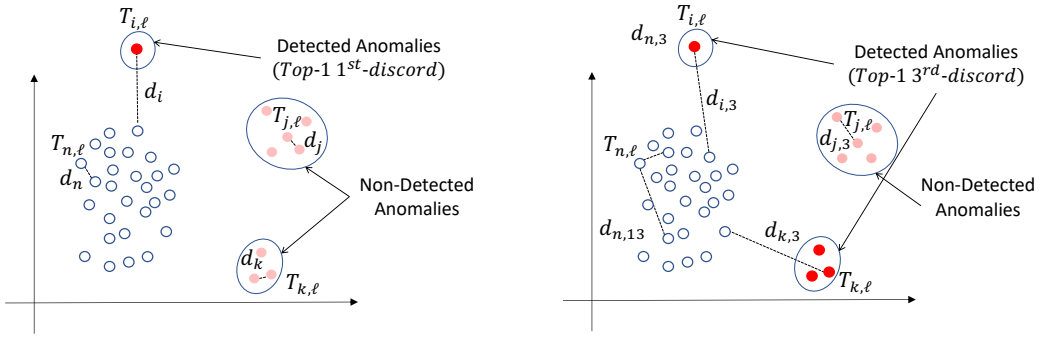
Naturally, in anomaly detection, we are not only interested in the most significant anomaly. Thus one can propose a definition that extends the previous two for the case of the k most significant anomalies:

Definition 5 (Top-k m^{th}-discord). *A subsequence $T_{i,\ell}$ is a Top-k m^{th}-discord if it has the k^{th} largest distance to its m^{th} NN, among all subsequences of length ℓ of T.*

Note that this definition subsumes the previous two: the simple discord (Definition 3) is equivalent to Top-1 1^{st}-discord, and the m^{th}-discord (Definition 4) is equivalent to Top-1 m^{th}-discord. Figure 2.8 illustrates these notions with an example, where for ease of exposition, we represent each subsequence as a point in 2-dimensional space. The figure depicts two 1^{st}-discords (Figure 2.8(a)): the discord in the top (Top-1) has its 1-NN at a larger distance than the other subsequences. Nevertheless, a group of similar anomalies (light-red points) is not correctly detected using Top-1 1^{st}-discords. Figure 2.8(b) also shows a group of three anomalous subsequences: using the Top-1-3^{rd}-discord is enough to identify the isolated point and the group of three anomalies. Nevertheless, the subsequences in the group of five anomalies are not detected.

There exist several studies that have proposed fast and scalable discord discovery algorithms in various settings [104, 60, 76, 36, 124, 22, 120, 77], including simple and m^{th}-discords[1], in-memory and disk-aware techniques, exact and approximate algorithms, using either SAX [60, 104] (Symbolic Aggregate approXimation) or Haar wavelets [22, 36]. In the following sections, we present the state-of-the-art solutions to the subsequence anomaly detection problem. Note that in this discussion we focus on the Top-k anomalies (using instead a threshold ϵ to detect anomalies would be a straightforward modification of the solution).

[1]The authors of these papers define the problem as k^{th}-discord discovery.



(***a***) *d defined as the discord distance* (Top-1 1^{st}-$discord$)

(***b***) *d defined as the* m^{th}-*discord distance* (Top-1 3^{rd}-$discord$ *for* $d_{[i,j,k,n],3}$ *and* Top-1 13^{rd}-$discord$ *for* $d_{n,13}$)

Figure 2.8: A dataset with 36 subsequences (of the same length ℓ) depicted as points in 2-dimensional space; 27 subsequences are normal (blue points), and 9 are anomalous (solid, red points).

2.5.4.2 Disk Aware Discord Discovery

Disk Aware Discord Discovery method [120] (DAD) is a method that proposes a new exact algorithm to discover $discord$ requiring only two linear scans of the disk thought for managing very large datasets. The algorithm uses the raw sequences directly. First, it addresses the more straightforward problem of detecting what is called *range discord*, and then it generalizes the problem to detect the Top-k discord.

Definition 6 (range discord). *Given a data series T and a range value r, are retrieved all the sub-sequences $T_{i,\ell}$ at distance at least r from their nearest neighbor, among all subsequences of length ℓ of T.*

Providing the range r may require some prior domain knowledge, but it can be obtained by studying the nearest neighbor distance distribution of a uniformly random sample of the overall dataset. DAD can be divided into two separate phases: candidate selection and discover refinement phases. During the first phase, one linear scan through the database is performed, and a set of candidates C that will contain all the discords with 1-NN larger than r (eventually, some false positives) is created. Then, the set C is refined during the refinement phase by removing all the false positives, adopting an early abandoning procedure. In the end, it is possible to refine the algorithm by ranking the obtained discords and retrieving the Top-k discords.

2.5.4.3 GrammarViz

$GrammarViz$ [104] adopts a different approach to find anomalies, based on the concept of Kolmogorov complexity where the randomness in a sequence is a function of its algorithmic incompressibility. An essential feature of this algorithm is the ability to find discords of

different lengths. The main idea is that it is possible to represent a data series as context-free grammar, and the parts of data series that map with few rules of the grammar is the anomalies. The algorithm can be divided into different phases. First, the whole data series is summarized using SAX (Symbolic Aggregate Approximation) to have discrete values and not continuous. Then a numerosity reduction procedure is applied to limit the problem of finding trivial matches [129] (a trivial match of $T_{i,\ell}$ is a subsequence $T_{a,\ell}$, where $|i - a| < \ell/2$, i.e., the two subsequences overlap by more than half their length). Next, context-free grammar is built using Sequitur, a linear space and time algorithm able to derive context-free grammar from a string incrementally. Finally is built a rule density curve which is the metadata that allows detecting anomalies. It is possible to obtain a rule density curve by iterating over all grammar rules and incrementing a counter for each data series that points to the rule spans. Once the rule density curve is obtained, it is possible to discover approximate (considering the discord definition) anomalies by picking the minimum values of the curve. Otherwise, by applying the algorithm RRA (Rare Rule Anomaly proposed in [104]), it is possible to discover exact discords.

2.5.4.4 Matrix Profile

Matrix Profile [124, 129] proposes a metadata data series computed efficiently, able to provide several useful information about the analyzed data series, among them the discords. For simplicity, we can call this metadata series matrix profile, and we can define it as follows:

Definition 7 (Matrix Profile). *A matrix profile MP of a data series T of length n is a data series of length $n - \ell - 1$ where the i^{th} element of MP contains the Euclidean normalized distance of the subsequence of length ℓ of T starting at i to its nearest neighbor.*

However, the latter definition does not tell us where that neighbor is located. This information is recorded in the matrix profile index:

Definition 8 (Matrix Profile Index). *A matrix profile index I_{MP} is a vector of index where $I_{MP}[i] = j$ and j is the index of the nearest neighbor of i.*

Two general definitions of Join matrix computation can be inferred. The first called $selfJoin$ corresponds exactly to the matrix profile. The second called $Join$ corresponds to the same operation but for two different data series. Formally we have:

Definition 9 (Data Series Self-Join). *Given a data series T, the self-join of T with subsequence length ℓ, denoted by $T \bowtie_\ell T$, is a metadata series, where: $|T \bowtie_\ell T| = |T| - \ell + 1$ and $\forall i, 1 \leq i \leq |T \bowtie_\ell T|$, $(T \bowtie_\ell T)_{i,1} = dist(T_{i,\ell}, 1^{st}NN \text{ of } T_{i,\ell})$.*

Definition 10 (Data Series Join). *Given two data series A and B, and a subsequence length ℓ, the $Join$ between A and B, denoted by $(A \bowtie_\ell B)$, is a metadata series, where: $|A \bowtie_\ell B| = |B| - \ell + 1$ and $\forall i, 1 \leq i \leq |A \bowtie_\ell B|$, $(A \bowtie_\ell B)_{i,1} = min(dist(B_{i,\ell}, A_{1,\ell}), ..., dist(B_{i,\ell}, A_{|A|-\ell+1,\ell}))$.*

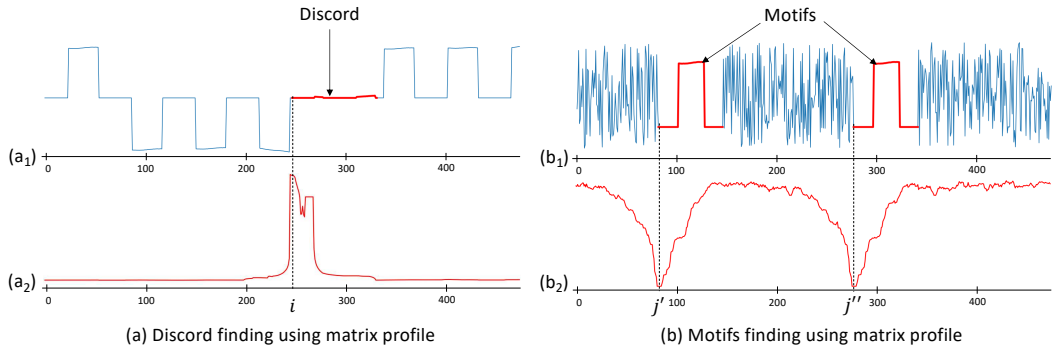


Figure 2.9: (a) Matrix profile (a_2) applied on the SED (Nasa disk failure datasets) data series snippet (a_1). The highest value in the matrix profile (a_1) points to an anomaly of the SED data series. (b) Matrix profile (b_2) applied on a synthetic data series (b_1). The smallest value in the matrix profile (a_1) points to a motif pair in the data series.

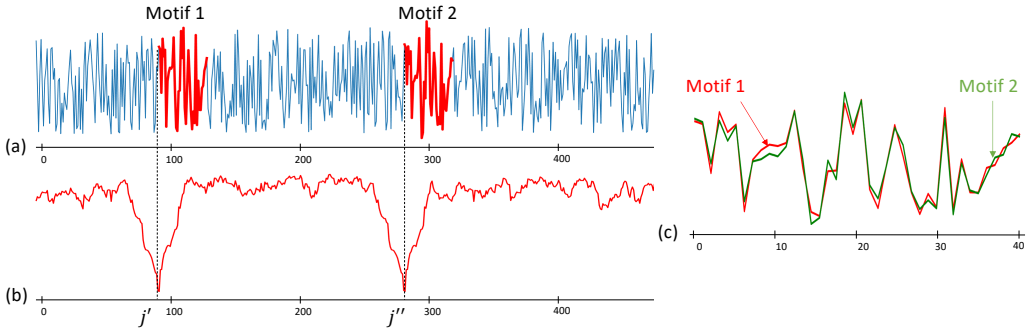


Figure 2.10: (a) Synthetic data series generated with random noise with two similar random subsequences injected (red). (b) Matrix profile of the data series in (a). (c) Two similar random subsequences injected in (a).

These metadata are computed using Mueen's ultra-fast Algorithm for Similarity Search (MASS) [85] that requires just $O(n * log(n))$ time by exploiting the FFT (Fast Fourier Transform) to calculate the dot products between the query and all the sub-sequences of the data series. Once these metadata are generated, it is possible to retrieve the $Top\text{-}k$ discord by considering the maximum value of the Matrix Profile and order it, excluding the trivial matches (overlapping subsequences). It is also possible to retrieve the subsequences with the shortest distance to their nearest neighbor (called $motifs$). These subsequences correspond to a recurrent motif in the data series and can, therefore, be helpful in the anomaly search.

Figure 2.9 shows an example of the Matrix Profile metadata. On the one hand, Figure 2.9 (a) shows that the discord found corresponds to a very different subsequence from the regular cycles. On the other hand, Figure 2.9 (b) shows that the distinct shapes are well identified as motifs.

Figure 2.10 depicts another example of motifs search. In this case, two similar randomly

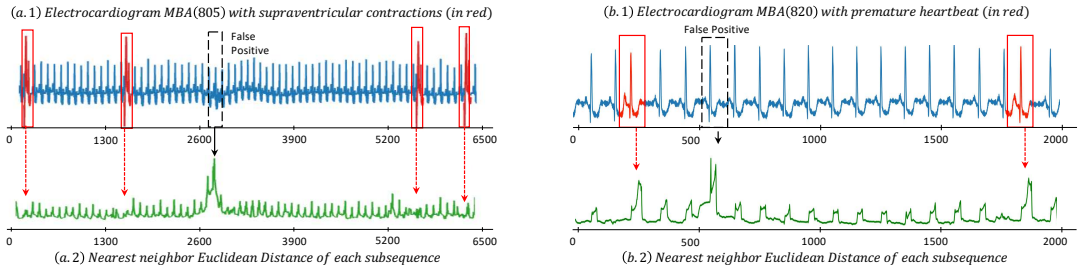


Figure 2.11: Nearest neighbor euclidean distance (in green) computed using Matrix Profile for (a) an electrocardiogram containing similar supra-ventricular contractions and (b) an electrocardiogram containing similar premature heartbeats.

generated subsequences (same random subsequences combined with small random noise shown in Figure 2.10(c)) are injected in a random noise synthetic data series (Figure 2.10(a)). Therefore, the motifs are not straightforward to identify. Nevertheless, the smallest value in the matrix profile points to the correct motifs.

However, if one anomaly occurs several times, the corresponding subsequences will be similar and thus considered as motifs. Thus the identification of multiple similar anomalies is hard using matrix profile. Figure 2.11 illustrates this limitation with two examples. Figure 2.11(a) corresponds to an electrocardiogram containing supra-ventricular contractions (in red) and Figure 2.11(b) corresponds to an electrocardiogram containing premature heartbeat. In these two cases, the abnormal subsequences are similar to each other, and thus the discord found using matrix profile does not correspond to the abnormal heartbeats.

2.5.4.5 Extended Methods on Similar concepts

The matrix profile algorithm has attracted much attention in the past few years, and several extensions have been proposed. Extended methods such as STAMPI [124] proposes an incremental implementation of the matrix profile algorithm permitting its usage for streaming applications. VALMOD [71] has been proposed in order to permit the detection of variable length motifs and discords. Finally, to accelerate the matrix profile computation, SCRIMP [130] and SCAMP [131] have been proposed to either (i) efficiently provide an accurate approximate matrix profile at any time while the exact computation is still running or (ii) modify matrix profile implementation in order to use GPUs.

2.5.5 New perspectives with graph-based approaches

The other way to detect anomalies in data series (and more generally detect any specific pattern) is to map the data series into a complex network (graph). This idea is attractive since this data structure benefits from rich and well-developed methods on complex network and graph analysis for various tasks as community detections. In our specific case of data series anomaly detection, the task would be to find a way to map the data series into

a graph that would help us to highlight the community of interest (the anomalies in our case).

More specifically, the graph should represent all the possible subsequence evolution as paths, in which the edge's weight could correspond to the number of times this evolution occurred. Having this kind of information would give us a way to build indicators to detect unusual subsequences. Secondly, the graph should be small to be used and analyzed easily, moving the problem from big data series pattern mining to small graph mining tasks. This compression has a cost and should be measured such as the best compromise is found between the accuracy (reducing the loss of information) and the speed (keeping the size such as classical graph algorithm are scalable to the anomaly detection task).

No such graph-based method has been proposed in the specific case of subsequence anomaly detection in data series. Nevertheless, we analyze the different methods to map a data series into a graph in the following section.

We first define some basic notions related to graph theory. We define a Node Set $\mathcal{N}$ as a set of unique integers (we set this type as default). Given a Node Set $\mathcal{N}$, an Edge Set $\mathcal{E}$ is then a set composed of tuples (x_i, x_j), where $x_i, x_j \in \mathcal{N}$; $w(x_i, x_j)$ is the weight of that edge. Formally, we define a graph as follows:

Definition 11 (Graph)**.** *Given a Node Set $\mathcal{N}$, an Edge Set $\mathcal{E}$ (pairs of nodes in $\mathcal{N}$), a Graph G is an ordered pair $G = (\mathcal{N}, \mathcal{E})$.*

In some specific cases, the directions of the edges matter. We thus define a directed graph as follows:

Definition 12 (Directed Graph)**.** *A directed graph or digraph G is an ordered pair $G = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N}$ is a Node Set, and $\mathcal{E}$ is an ordered Edge Set.*

2.5.5.1 Visibility Graph

Visibility Graph [65] is a way to model a data series into a complex network where each data series point is a node. The main idea is to connect the nodes such as one node can "see" the other one. This visibility criterion is formally defined as follows [65]: two arbitrary data points T_i and T_j will have visibility and consequently will be connected by an edge if any other point T_k placed between them ($i < k < j$) fulfills:

$$T_k < T_j + (T_i - T_j)\frac{j-k}{j-i}$$

Therefore, the resulted graph (also called Natural Visibility Graph) $NVG = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N} = \{T_0, T_1, ..., T_n\}$, and $\mathcal{E}$ are the edges (T_i, T_j) such as T_i and T_j satisfy the previous equation. This graph is illustrated in Figure 2.12 (a) (blue arrows without considering the direction).

This simple formulation also provides us some strong guarantees about the inner structure of the graph and its construction invariance to some transformation. These guaranties

are listed as follows: (i) Since each point is linked to at least its nearest neighbors in time (left and right), the complete graph is connected. (ii) The complete graph is undirected and allows us more freedom in which algorithm to use. (iii) The graph construction is invariant under the affine transformation of the data series.

Nevertheless, this modeling manner has some severe drawbacks. Even though the time complexity is somewhat limited ($O(n * log(n))$ in the worst case, in nearly linear in average), the memory complexity is more problematic. Since every point of the data series will correspond to a node, this method does not fit the case of a very large data series. Moreover, the number of edges being at least equal to the number of nodes (and orders of magnitude more considerable on average), it becomes very challenging to deals with such a big graph to perform any analysis like anomaly detection.

Horizontal Visibility Graph [78] (HVG) is a specific type and a subgraph of Natural Visibility Graph. Both of them have the same set of nodes but do not share the same set of edges. As regards HVG, two nodes are connected if one can draw a horizontal line without crossing any other points. Formally, the visibility criterion can be defined as follows: two arbitrary data points T_i and T_j will have visibility and consequently will be connected by an edge if any other point T_k placed between them ($i < k < j$) fulfills:

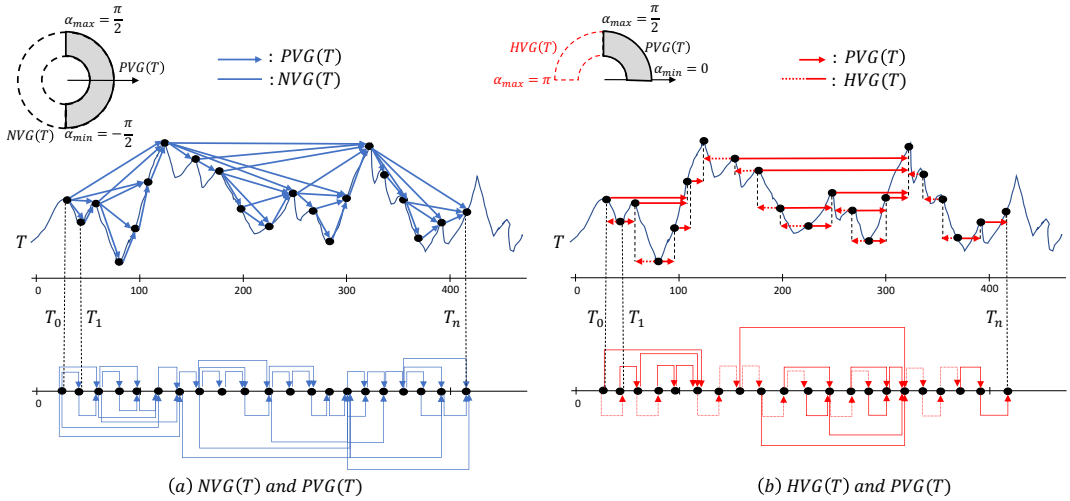
$$T_i > T_k \text{ and } T_j > T_k$$

Therefore, the resulted graph (also called Horizontal Visibility Graph) $HVG = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N} = \{T_0, T_1, ..., T_n\}$, and $\mathcal{E}$ are the edges (T_i, T_j) such as T_i and T_j satisfy the previous equation. This graph is illustrated in Figure 2.12 (b) (both dotted line and red line without considering the directions of the arrows). As one can see, the size of the produced HVG is smaller and easier to use than the corresponding NVG underlined in Figure 2.12 (a).

In order to have more control, one can decide to parametrize the graph such as the visibility is constrained in one direction. This direction can be seen as the angle between two nodes (i.e. data series points), and the visibility criterion can be formally defined as follows [9]: two arbitrary data points T_i and T_j will have visibility and consequently will be connected together by an edge, if any other point T_k placed between them ($i < k < j$) fulfills:

$$T_k < T_j + (T_i - T_j)\frac{j-k}{j-i}$$
$$\alpha_{min} < arctan\big(\frac{T_j - T_i}{j-i}\big) < \alpha_{min}$$

Therefore, the resulted graph (also called Parametric Visibility Graph) $PVG = (\mathcal{N}, \mathcal{E})$ where $\mathcal{N} = \{T_0, T_1, ..., T_n\}$, and $\mathcal{E}$ are the edges (T_i, T_j) such as T_i and T_j satisfy the previous equation. This graph is illustrated in Figure 2.12 (a) and (b) (full blue and red arrows). Note that whatever the data series, NVG and PVG using $(\alpha_{min} = -\frac{\pi}{2}, \alpha_{max} = \frac{\pi}{2})$ have the same edges. The only difference is that the edges are oriented from left to right.



(a) $NVG(T)$ and $PVG(T)$ (b) $HVG(T)$ and $PVG(T)$

Figure 2.12: For a given data series T, (a) the corresponding Natural Visibility Graph (NVG), and (b) the corresponding Horizontal Visibility Graph (HVG).

In the case when we have $(\alpha_{min} > -\frac{\pi}{2}, \alpha_{max} < \frac{\pi}{2})$ (as illustrated in Figure 2.12 (b)), PVG has less edges than NVG (and HVG in Figure 2.12 (b)).

2.5.5.2 Introduction to State-space reconstruction

State-space reconstruction is the foundation of nonlinear data series analysis [54, 19] first proposed by Packard et al. [89]. In principle, it allows the reconstruction of the dynamics and the behavior of a univariate data series. As claimed by Bradley et al. [19], these reconstructions can be useful to find data series behaviors insight with guarantees to hold in the original data series. The classical strategy for state-space reconstruction is *time delay embedding*, where a data series value and its predecessors separated with a given delay is used as a vector to embed the given data series value. Formally, given a data series T and a time delay τ, a m dimensional state-space reconstruction vector R_i can be defined as follows:

$$R_i = \langle T_i, T_{i-\tau}, T_{i-2\tau}, ..., T_{i-m\tau} \rangle$$

As stated by Bradley et al. [19], mathematically, one can equivalently take forward delays instead of backward ones, but for practical purposes, it is better to obey causality laws. If τ is very small, the coordinates in each of these vectors are strongly correlated. In the particular case when $\tau = 1$, it becomes a simple subsequence embedding. As τ is getting larger, reconstruction unfolds off the data series space domain and highlights some deeper (i.e., long periods) features and insights.

The primary interrogation that remains is: how to pick τ and the number of dimensions m? Of course, this question is dependent on the task one wants to achieve. Nevertheless, a commonly used technique called the C-C method has been defined to find the best delay [61], and false nearest neighbors methods [56] appears to be very useful to determine

m.

Regarding our application to anomaly detection, it is important to link these parameters and the commonly used ones. Indeed the combination of the number of dimensions m and the time delay τ can be compared to the subsequence length ℓ used by both supervised and unsupervised methods previously enumerated. As mentioned earlier, given a dimension m and for the specific case when $\tau = 1$, the state space reconstruction vector is simply the subsequence of length m. While τ increases, the vector becomes a downsampling version of the subsequence length $\tau * m$.

2.5.5.3 Complex Network Analysis

Sharing the same motivation as Lacasa et al. [65], several other works also provide a way to build a complex network from a data series. These methods are recurrent-based and distance-based. We will develop these two types in the following sections.

Several works (Gao et al. [38, 39]) proposed to build the graph using the recurrence matrix Rc. For a given data series T, R its state space reconstruction vectors, Rc is defined as follows:

$$Rc_{i,j} = \Theta(\epsilon - ||R_i - R_j||), \text{for } i, j \in [0, |T|] A = R - I$$

The function Θ is the Heaviside function, and $\epsilon \in \mathbb{R}$. Matrix R is then used to build the adjacency matrix of the corresponding complex network. Matrix A is the adjacency matrix of our future graph. Matrix I is the identity matrix. The recurrence matrix itself has become a basic tool of nonlinear data series analysis and was first introduced by Eckmann et al. [32]. The constructed complex graph is supposed to store most of the helpful information that the Rc matrix has.

2.6 Supervised Subsequences Anomaly Detection in Data Series

Each method listed in the previous sections was fully unsupervised. However, one can argue that the best way to detect anomalies is to learn what an anomaly should look like from the past. It is possible if and only if the experts provide a large number of annotations. (i) In this case, the anomaly detection task can be considered as a classification class with two classes: normal subsequences and abnormal subsequences. Nevertheless, (ii) in several other cases, the construction of an abnormal subsequences class is impossible. Thus, the only supervision possible is by using the normal subsequence class as labels. We discuss in the following sections the solutions in the literature to handle cases (i) and (ii).

2.6.1 Semi-Supervised Machine Learning Approaches

As mentioned earlier, it is sometimes easier to build a dataset composed of normal behavior (real or simulated), and one can try to train a model to detect the normality and consider as abnormal everything that the model cannot detect. In this case, we supervise the model to learn the normality, but we complete the detection in an unsupervised way. We discuss in the following section the various machine and deep learning methods that apply this strategy. However, one should note that all the following methods have a serious advantage (the training step) over the methods mentioned in the previous sections, and therefore, should be compared very carefully.

2.6.1.1 Control Chart

Control Chart [27] is a proposed method that aims to measure the stability of a process usually applied to control the quality and the deviation of a process (or a time series) to an unwanted state. The deviation is measured by defining statistics of interest (such as mean, median, standard deviation) and then by defining a range (using lower and upper bound thresholds) in which these statistics should belong. These bounds can be set manually, but then it requires a strong knowledge of the analyzed process.

2.6.1.2 Hidden Markov Model

Hidden Markov Model [24] (HMM) is a statistical Markov model in which we assume that the system is a Markov process. Thus, such a model can be used for data series.

The latter aims to learn unobservable states (hidden states) from the data series. For the specific case of anomaly detection, the hidden states are learned using data series without any anomalies (provided by the expert). From the Hidden Markov Model, one can compute the probability of any new sequence of points. Therefore, one can expect that the probability of an abnormal sequence will be lower than a regular sequence of points. Thus, a threshold can be chosen in order to detect abnormal sequences.

2.6.1.3 One-class SVM

One-class Support Vector Machine (SVM) for novelty detection and anomaly discovery aims to separate the instances from an origin and maximize the distance from the hyperplane separation [101] or spherical separation [111]. This method is a variant of classical Support Vector Machine for classification tasks [48]. Originally, given ℓ-dimensional training data points $x_0, ...x_n \in \mathcal{X}$ (in our case these points correspond to subsequences of length ℓ), a non linear function ϕ that maps the feature space $\mathcal{X}$ to a dot product space F, a kernel $k(x, y) = (\phi(x), \phi(y))$ (usually set to a Gaussian kernel $k(x, y) = e^{-||x-y||^2/c}$), the quadratic program to solve using a hyperplane is the following:

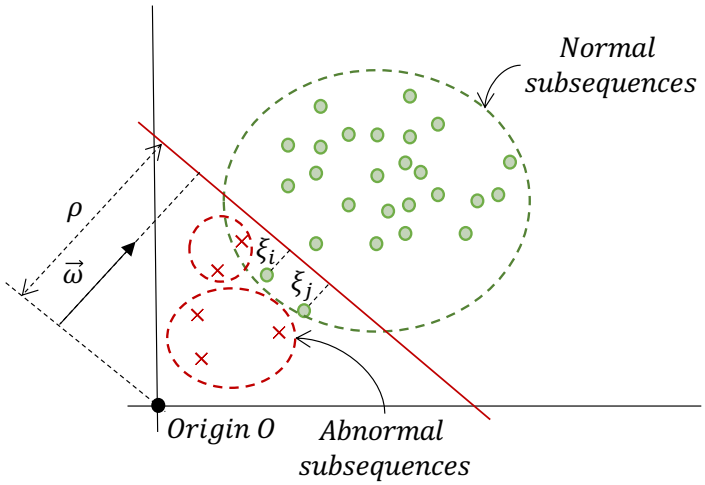


Figure 2.13: One class SVM illustration in which a point corresponds to a subsequence and only the green point are provided for the training step.

$$\min_{\omega \in F, \xi \in \mathbb{R}, \rho \in \mathbb{R}} \frac{1}{2}||w||^2 + \frac{1}{\nu\ell}\sum_i \xi_i - \rho$$
$$\text{subject to: } (\omega.\phi(x_i)) \geq \rho - \xi_i,$$
$$\xi_i \geq 0.$$

Figure 2.13 illustrates the principle of One-class SVM. For a given new instance x, by deriving the dual problem, the decision function can be defined as follows:

$$f(x) = sgn(\sum_i \alpha_i k(x_i, x) - \rho)$$

Note that the instances x_i associated with non-zero α_i are called the support vectors. Their value can be found by solving the following dual problem:

$$\min_\alpha \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j k(x_i, x_j)$$
$$\text{subject to: } 0 \leq \alpha_i \leq \frac{1}{\nu\ell},$$
$$\sum_i \alpha_i = 1.$$

The quadratic program to solve using a spherical separation, with a given center a (linear combination of the support vector for which the Lagrange multiplier is non-zero), a radius R and a penalty parameter C is defined as the following:

$$\min_{a \in F, R \in \mathbb{R}} R^2 + C \sum_i \xi_i$$
$$\text{subject to: } ||x_i - a||^2 \geq R^2 - \xi_i$$
$$\xi_i \geq 0.$$

If we assume that such an optimization problem can be solved, we can use the decision function as an anomaly score. To do so, one has to ensure to train the One-class SVM model on a normal section of the data series only (those have to be annotated by a knowledge expert and therefore require extra work to be used).

2.6.1.4 LSTM-based anomaly detection method

Long Short Term Memory (LSTM) [50] network has demonstrated to be particularly efficient to learn inner features for subsequences classification or data series forecasting. Moreover, such a model can also be used for anomalies detection purpose [81, 35].

The principle of the last paper is simple as follows. A stacked LSTM model is trained on *normal* parts of the data series T. The objective is to predict the point $T_i \in T$ or the subsequence $T_{i,\ell}$ using the subsequence $T_{i-\ell,\ell}$. Consequently, the model will be trained to forecast a healthy state of the data series, and therefore, will fail to forecast when it will encounter an anomaly. A LSTM unit (composed of n_h cells) is composed of 4 trainable matrix tuple defined as $(W_q, U_q, b_q) \in \mathbb{R}^{n_h, D}$ (with $D = 1$ if the input is a univariate data series). The subscript q can be equal to f, j, o, c, which corresponds to the four gates in the LSTM unit. Figure 2.15 depicts the components and interactions within a LSTM unit (in blue the gates containing trainable weights). The various components are given by:

$$\begin{aligned}
f_i &= \sigma_g(W_f * T_i + U_f * h_{i-1} + b_f) \\
j_i &= \sigma_g(W_j * T_i + U_i * h_{i-1} + b_j) \\
o_i &= \sigma_g(W_o * T_i + U_0 * h_{i-1} + b_o) \\
c_i &= f_i \circ c_{i-1} + j_i \circ \sigma_c(W_c * T_i + U_c * h_{i-1} + b_c) \\
h_i &= o_i \circ \sigma_h(c_i)
\end{aligned}$$

In the latter equation, the operator $\circ$ denotes the Hadamard product. By combining a large number of cells (outlined in Figure 2.15), and stacking them [81], one can fit the weights to forecast the data series in two different ways described as follows: (i) the first is to train the network using a fixed subsequence length $[T_{i-\ell}, ..., T_{i-1}]$ to predict T_i, (ii) or using the same input to predict the incoming sequence $[T_i, ..., T_{i+\ell'}]$. For the specific purpose of anomaly detection, we will assume that such a model can be trained to achieve both of the previously enumerated tasks. Then, this model has to be trained on the normal section of the data series (a priori annotated by knowledge expertise), and the forecasting error can be used as an anomaly score. Therefore, similarly to the one class SVM, one can expect to obtain a more significant forecasting error for a subsequence that the model has never seen during its training (like an anomaly) rather than a usual subsequence.

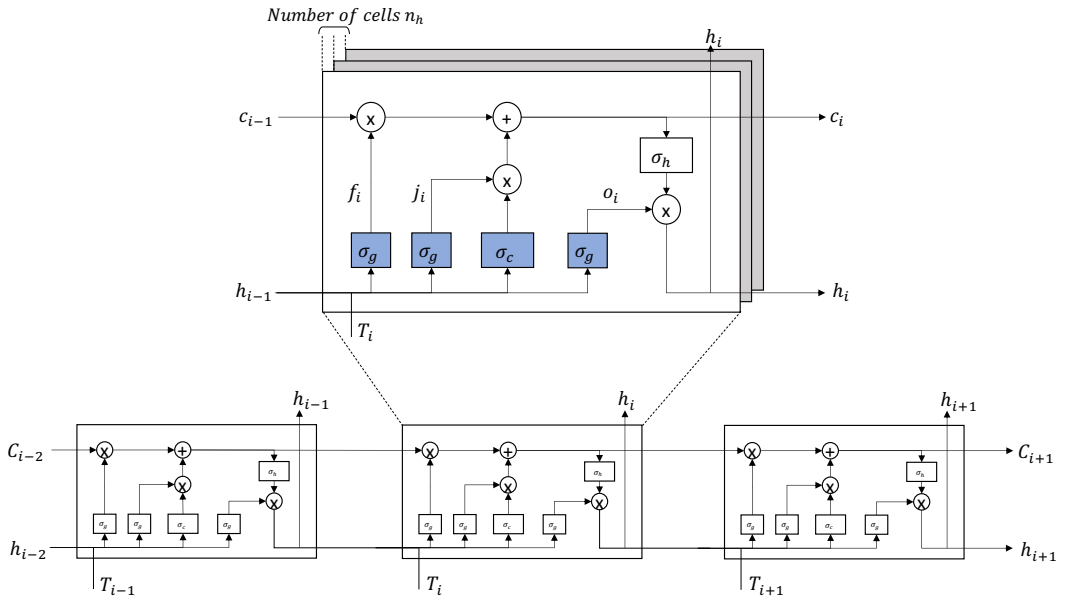


Figure 2.14: LSTM unit architecture.

2.6.1.5 GAN-based anomaly detection method

Generative Adversarial Network (GAN) is another network architecture that has attracted some serious interest. It was initially proposed for image generation purposes [41] but can be used to generate data series as well.

Let us first consider multilayer perceptrons. A GAN is composed of two networks. The first one is called the generator $G(z, \theta_g)$ with θ_g its parameters. The second one is called the discriminant $D(x, \theta_d)$ with θ_d its parameters and x a sequence of a given length ℓ. The output of G is the same shape as the input, and the output of D is a scalar $D(x)$ that represents the probability that x came from the original dataset, and therefore $1 - D(x)$ is the probability of x to have been generated by G. Formally G and D have to be optimized such as the two-players min-max game with value function $V(G, D)$ defined as follows:

$$\min_G \min_D V(G, D) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))]$$

For $\mathbb{T}_\ell$ the set of subsequences to train on, and $\mathbb{Z}$ the corresponding set of vectors from the latent space, we have the following stochastic gradient descent:

$$Discriminant : \nabla_{\theta_d} \frac{1}{|\mathbb{T}_\ell|} \sum_{(T_{i,\ell}, Z) \in (\mathbb{T}_\ell, \mathbb{Z})} [-logD(T_{i,\ell}) - log(1 - D(G(Z)))]$$

$$Generator : \nabla_{\theta_g} \frac{1}{|\mathbb{Z}|} \sum_{Z \in \mathbb{Z}} [1 - log(1 - D(G(Z)))]$$

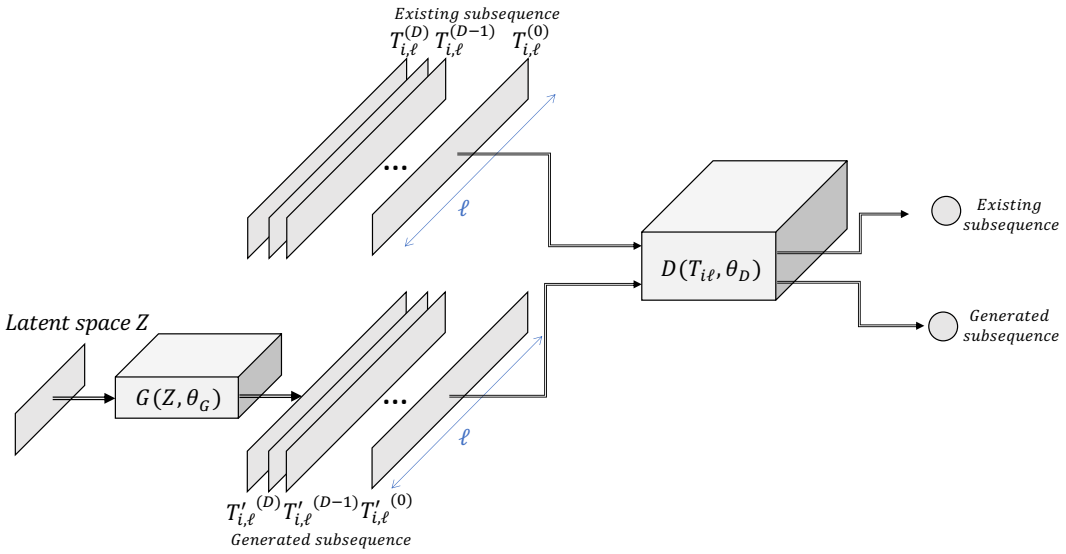


Figure 2.15: Illustration of GAN Network.

This architecture has been tried for the specific case of data series anomaly detection [69]. Nevertheless, a sliding window method has been used to go deeper into the subsequences and adapt to anomaly detection within a data series. We describe in this section the possible adaptation.

For the specific case of anomaly detection, the generator is trained to produce subsequences labeled as normal, and the discriminator is trained to discriminate the anomalies. Thus training such a model requires having a training dataset with normal subsequences only. However, subsequence anomaly detection is not required.

To detect the anomaly, one can use both the discriminator and the generator simultaneously. First, given that the discriminator has been trained to separate real (i.e., normal) from fake (i.e., anomaly) subsequences, it can be used as a direct tool for anomaly detection. Nevertheless, the generator can also be helpful. Given that the generator has been trained to produce realistic subsequences, it will most probably fail to produce real anomalies. Therefore, the Euclidian distance between the subsequences to evaluate and what would have simulated the generator with the same latent input can be used to discriminate anomalies.

Formally, given a subsequence $T_{i,\ell}$ (univariate or multivariate) in the data series T, a trained generator G, a $distance$ function (in practice Euclidian, Mannathan, or covariance), one has to find $\tilde{Z}_i$ such that:

$$\tilde{Z}_i = \min_{Z_i} distance(T_{i,\ell}, G(Z_i))$$

Thus the residual error of $T_{i,\ell}$ between its best latent mapping and itself is defined as follows:

$$Res(T_{i,l}) = \sum_{j=0}^{\ell} |T_{i+j,1} - G(\tilde{Z}_i)_{j,1}|$$

Finally, given a subsequence $T_{i,\ell}$, a value λ to tune based on the importance we give to the generator and the discriminant, the anomaly score can be defined as a convex combination of the residual error and the answer of the discriminant:

$$L_{T_{i,\ell}} = \lambda Res(T_{i,l}) + (1 - \lambda)D(T_{i,l})$$

This subsequence score is then combined with a cumulative sum to build a global score for the data series instance [69]. However, in the specific case of subsequence anomaly detection within a data series, which is the focus of this work, the latter subsequence score is enough.

2.6.1.6 Autoencoder

Autoencoder is a type of artificial neural network used to learn to reconstruct the dataset given as input using a smaller encoding size to avoid identity reconstruction. As general idea, autoencoder will try to learn the best latent representation (also called encoding) using a reconstruction loss. Therefore, it will learn to compress the dataset into a shorter code, and then uncompress that code into a dataset that closely match the original. Formally, given two transition functions ϕ and ψ, respectively called encoder and decoder, the task of an autoencoder is the following one:

$$\begin{aligned} \phi &: \mathbb{T}_\ell \to \mathcal{Z} \\ \psi &: \mathcal{Z} \to \mathbb{T}_\ell \\ \phi, \psi &= arg \min_{\phi,\psi} \mathcal{L}(T_{i,\ell}, \psi(\phi(T_{i,\ell}))) \end{aligned}$$

$\mathcal{L}$ is a loss function that is usually set to the mean square error of the input and its reconstruction, formally written $||T_{i,\ell} - \psi(\phi(T_{i,\ell}))||^2$. Regarding subsequences in data series, this loss fits well the task since it coincides with the Euclidian distance.

For the specific task of anomaly detection, the reconstruction error can be used as an anomalous score. The model trained on the non-anomalous subsequence of the data series is optimized to reconstruct the normal subsequences. Therefore, all the subsequences far from the training set will have a more considerable reconstruction error. Figure 2.16 depicts the framework of autoencoder for data series anomaly detection task. Moreover, Figure 2.16 depicts the reconstructed subsequences using an autoencoder with encoder($Conv(64, 3)$-$Relu()$-$Dense()$-$Tanh()$), and decoder($DeConv(64, 3)$-$Relu()$-$Dense()$-$Tanh()$). Figure 2.16, where the record 803 of the electro-cardiogram physiobank dataset is used as example, illustrates the significance of the reconstruction error in the anomaly found.

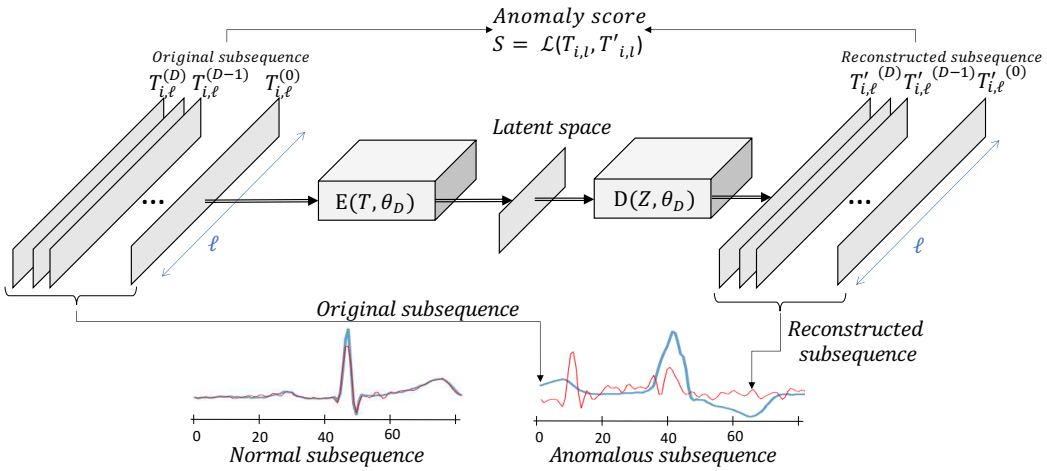


Figure 2.16: Autoencoder architecture framework for data series subsequences anomaly detection. The loss $\mathcal{L}$ is used to both train the model (on the non anomalous subsequences), and score the new subsequences.

2.6.2 Supervised Machine Learning Approaches

As mentioned earlier, if annotations are available (for both normal and abnormal subsequences), one can consider the anomaly detection task as a classification task. Data series classification is considered as a challenging problem in data mining and a well studied task [119, 33]. To address the task mentioned above, various data series classification algorithms have been proposed in the past few years [5], applied in a large number of use cases. Among them, one of the most popular and traditional data series classification methods is based on distances to the instances nearest neighbors. While the Euclidian distance seems to miss the similarity features needed to classify correctly, the Dynamic Time Warping (DTW defined in Section 2.5.1) distance is a solid baseline [30]. Naturally, it has been confirmed that ensembling all the individual distance-based classifiers outperforms all of them separately. Nevertheless, recent works have shown that ensemble-based methods using other kinds of classifiers than just distance-based methods are now state-of-the-art [6]. Finally, recent works have been conducted using a deep learning method for data series classification [34]. Most of the famous network architectures from the computer vision literature have been tried for data series classification [114]. Some variants, including multi-length and standard transformation steps, have also been proposed [28]. In this section, we briefly describe the methods mentioned above.

2.6.2.1 Distance-Based methods

One of the most popular and traditional data series classification methods is based on distances to the instances nearest neighbors. It consists of computing for each instance the distance to its nearest neighbor (or to its k^{th} nearest neighbor) in a given training set. We then attribute to the instance the class of its nearest neighbor. For that purpose, two dis-

tance measures are commonly used: the Euclidian and the Dynamic Time Wrapping one. The first distance seems to fail to catch the similarity features to classify correctly. On the contrary, the Dynamic Time Warping (DTW) distance is accurate [30]. Moreover, even though DTW computation is slow, DTW first Nearest Neighbor (DTW 1-NN) classifier does not require any training phase and represents an efficient solution.

2.6.2.2 Heterogeneous Ensemble-based methods

Rather than simply using one classifier to predict which class one instance belongs to, several works have been proposed to use ensembling methods to improve the classification accuracy [73, 6]. Such methods either ensemble several distance-based methods [72], but ensemble also dictionary-based [99], shapelet-based [49, 74, 84, 123], frequency-based and other general transformed-based methods [73, 7]. Such methods have shown to be significantly more accurate than all the previous state-of-the-art classifiers for data series. The proposed Hierarchical Vote Collective of Transformation-Based Ensembles (HIVE-COTE) remains until now the most accurate approach to classify data series. However, its execution time complexity and memory make it hardly usable in practice.

2.6.2.3 Deep Learning methods

Following the recent advances made for image analysis and classification, deep learning methods and neural networks have been used to classify univariate and multivariate data series. Such methods have demonstrated solid results and are now considered to be solid baselines. We now discuss how neural networks are used.

2.6.2.4 Neural Network Notations

We are interested in classifying data series using a neural network architecture model. We now define the essential components of neural networks.

Dense layer: The basic layer of neural network is a fully connected layer (also called $Dense\ layer$) in which every input neuron is weighted and summed before passing through an activation function. For univariate data series, given an input data series $T \in \mathbb{R}^n$, given a vector of weights $W \in \mathbb{R}^n$ and a vector $B \in \mathbb{R}^n$, we have:

$$h = f_a\left(\sum_{T_i, w_i, b_i \in (T,W,B)} w_i * T_i + b_i\right) \tag{2.2}$$

f_a is called the activation function and is a non-linear function. The commonly used activation function f_a is the rectified linear unit (ReLU) [88] that prevents the saturation of the gradient. Nevertheless, other activations are also used in the literature as $Tanh$, $Leaky\ ReLU$ [118] and many others. For the specific case of multivariate data series, all

dimensions are concatenated to give input $T, W \in \mathbb{R}^{D*n}$. Finally, one can decide to have several output neurons. In this case, each neuron is associated with a different W and B, and Equation 2.2 is executed independently.

Convolutional layer: Convolutional layer has shown significant quality to detect shape based and sub features in images [63, 68, 114], and recently in data series classification [34]. In general, convolutional layers are used as feature extractors and stacked together before being passed through a dense layer that plays the role of classifier. Formally, for multivariate data series, given an input vector $T \in \mathbb{R}^{(D,n)}$, and given matrices weights $\mathbf{W}, \mathbf{B} \in \mathbb{R}^{(D,\ell)}$, the output $h \in \mathbb{R}^n$ of a convolutional layer can be seen as a univariate data series. The tuple (W, B) is also called kernel, with (D, ℓ) the size of the kernel. Formally, for $h = [h_0, ..., h_n]$, we have:

$$h_i = f_a\Bigg(\sum_{\substack{T^{(j)},W^{(j)},B^{(j)} \in \\ (T,\mathbf{W},\mathbf{B})}} \sum_{\substack{T_k,w_k,b_k \in \\ (T^{(j)}_{i-\lfloor \frac{\ell}{2} \rfloor, i+\lfloor \frac{\ell}{2} \rfloor}, W^{(j)}, B^{(j)})}} w_k * T_k + b_k \Bigg) \tag{2.3}$$

In practice, we have several kernels of size (D, ℓ). The result is a multivariate series with dimensions equal to the number of kernels, n_f. For a given input $T \in \mathbb{R}^{(D,n)}$, we define $A \in \mathbb{R}^{(n_f,n)}$ to be the output of a convolutional layer $conv(n_f, \ell)$. A_m is thus a univariate series corresponding to the output of the m^{th} kernel (also called filter).

Global Average Pooling: Another type of layer, Pooling layers compute average/max/min operations. A specific type of pooling layer is Global Average Pooling (GAP). This operation is averaging an entire output A_m of the m^{th} kernel of a convolutional layer into one value, thus providing invariance to the position of the discriminative features. Formally, for a set of filters $A_m \in A$, a Global Average Pooling (GAP) is defined as follows:

$$y_m = \sum_{j \in A_m} \sum_{i \in A_m} (A_m)^{(j)}_i \tag{2.4}$$

Learning Phase: The learning phase uses a loss function $\mathcal{L}$ that measures the accuracy of the model and optimizes the various weights. For the sake of simplicity, we note Ω the set containing all weights (e.g., matrices $\mathbf{W}$ and $\mathbf{B}$ defined in the previous sections). Given a set of data series $\mathcal{T}$, we define the average loss as:

$$J(\Omega) = \frac{1}{|\mathcal{T}|} \sum_{T \in \mathcal{T}} \mathcal{L}(T) \tag{2.5}$$

Then for a given learning rate α, the average loss is back-propagated to all the weights in the various layers. Formally the back-propagation is defined as follows:

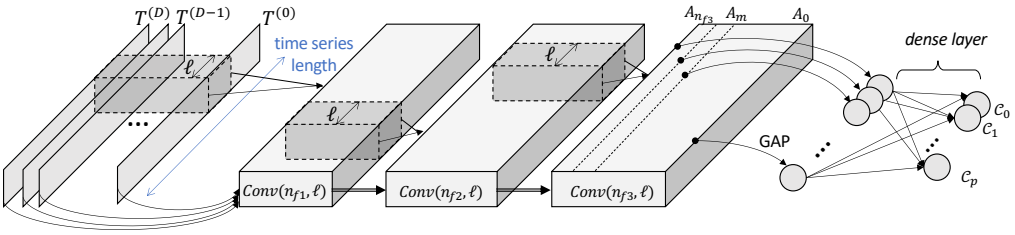


Figure 2.17: Convolutional Neural Network architecture for multivariate data series.

$$\forall \omega \in \Omega, \omega = \omega - \alpha \frac{\partial J}{\partial \omega}$$

In the section, we will use the stochastic gradient descent using the ADAM optimizer [62]. Our contribution aims to focus on which loss function to back-propagate and what combination of the inputs should be applied to maximize the classification accuracy.

2.6.2.5 Convolutional Neural Network Architecture

A classical deep learning architecture used to perform data series classification is Convolutional Neural Network Architecture [34, 114] (CNN). It corresponds to a concatenation of convolutional layers (joined with the $ReLU$ activation function, batch normalization, and $MaxPooling$ layer). The last convolutional layer is then connected with a Global Average Pooling layer and a dense layer. Unlike Multi-Layer Perceptron (MLP) architecture, the same convolutions kernel (the weight w and the bias b) will be used for all the subsequences in the data series. Thus, the features learned are invariant of the position in the data series. Moreover, instances of multiple lengths can be used with the same network. Figure 2.17 depicts the CNN architectures (with a multivariate data series, for which each dimension is stacked in one channel) with three convolutional layers and a GAP layer.

2.6.2.6 Residual Neural Network Architecture

A second classical neural network architecture is Residual Neural Network architecture [34, 114] (ResNet). This architecture is based on the classical CNN, to which we add residual connexions between successive blocks of convolutional layers. These connections prevent the gradient from exploding or vanishing during the learning phase and allow experts to use deeper (i.e., a high number of layers) architecture. Figure 2.18 illustrates the ResNet architecture.

2.6.2.7 Advanced variant Architectures

Other advanced methods have been proposed in the literature. Most of them are based on the same components mentioned above (convolutional layers). A first approach proposed

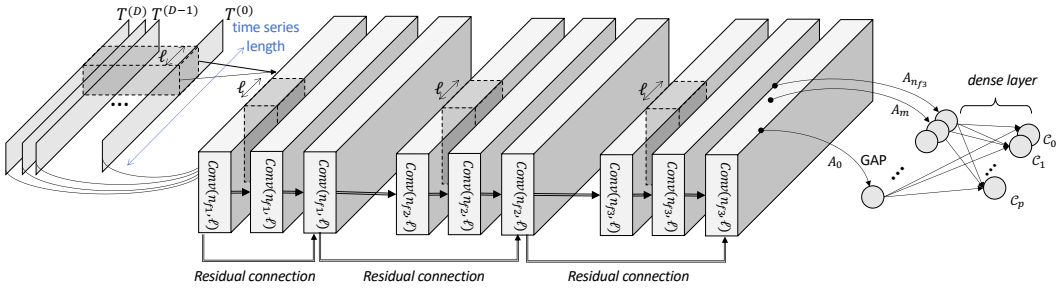


Figure 2.18: Residual Neural Network architecture for multivariate data series.

in the literature and mentioned in deep learning methods surveys [34, 114] is called Encoder [105]. The latter is a hybrid deep CNN for which an attention layer replaces the GAP layer. It enables the network to learn which parts of the data series (in the time domain) are important for a certain classification. However, this approach seems to be less accurate than ResNet or CNN [34].

A second approach proposed is Multi-scale Convolutional Neural Network (MCNN) [28]. This architecture is trained using identical input subsequences with different lengths (using either downsampling or smoothing operation). The goal is to learn features of different temporalities and to be able to detect patterns and trends of different lengths using only one architecture. Nevertheless, this approach seems to be less accurate than ResNet or CNN [34].

A third variant approach is Multi-Channel Deep Convolutional Neural Network (MCDCNN) [127], which was originally proposed for multivariate data series. The main difference between this architecture and the classical convolutional neural network architecture is that the convolutions are computed independently for each dimension.

Moreover, Time LeNet [42] is another variant architecture proposed inspired by the LeNet architecture from document recognition task [67]. This architecture is very similar to the classical CNN architecture (concatenation of convolutional layers) but contains $MaxPooling$ operations between the convolutional layers. It enables the network to learn general features that are invariant to small perturbations. The last difference between CNN and Time-CNN remains in the usage of a fully connected dense layer (connected to every neuron of the last convolutional layer) instead of a GAP layer. Thus, due to the very high number of weights on the last dense layer, this architecture is significantly slower to train. Moreover, this approach seems to be less accurate than the other proposed methods [34].

InceptionTime [52] is a recently proposed architecture as an equivalent of AlexNet for time series. InceptionTime is an ensemble of five deep learning models, and each one is created using multiple Inception modules [108]. Each individual model has the same architecture. The core idea of an Inception module is to use multiple filters simultaneously to an input time series. The module includes kernels of varying lengths, which permits the network to learn relevant features of variable lengths.

Finally, other architectures have been proposed like Time Warping Invariant Echo State Network (TWIESN) [110] (a recurrent architecture originally proposed for data series fore-

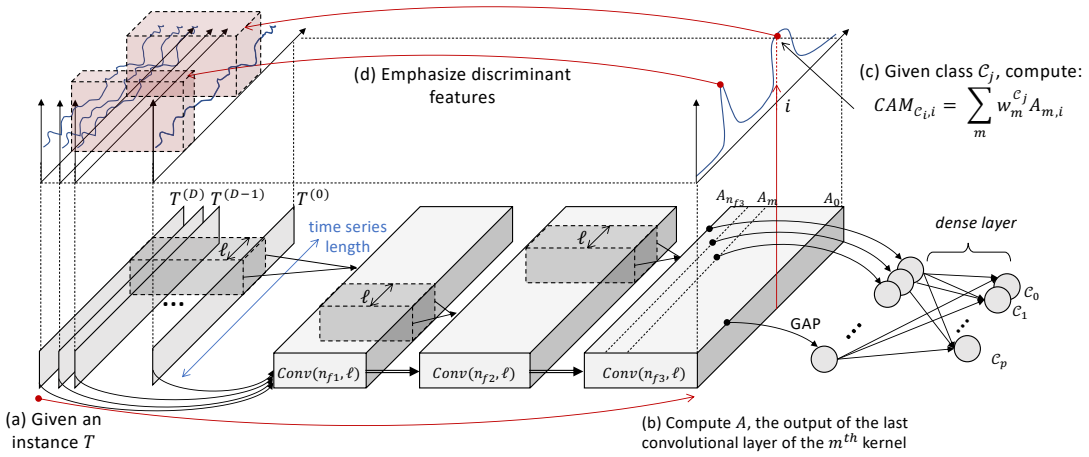


Figure 2.19: Illustration of Class Activation Map for CNN architecture.

casting) tested and implemented for data series in [34], and Time Convolutional Neural Network (Time-CNN) [126], an architecture that uses MSE instead of the classical cross-entropy loss function. However, these architectures seem to have significantly worse results than the classical architectures [34].

2.6.3 Finding Precursors of Abnormal Subsequences

The previously mentioned supervised methods have two significant benefits: (i) as annotations and supervision have been provided, these methods are naturally more accurate than unsupervised and semi-supervised approaches, (ii) the anomaly investigation can be further extended in order to identify *precursors* or *symptoms* of the classified anomaly. The latter can be seen as discriminant features or subsequences characterizing the abnormal class. We now discuss methods that identify such discriminant features.

2.6.3.1 Discriminant features identification

In order to identify discriminant features, one needs to interpret the decision made by the model and identify which section of the input data is used to make this decision. One requirement is to have a trained model with a high enough accuracy. If the accuracy is low, the discriminant features will be irrelevant. Moreover, the discriminant features might not be one entire dimension or one point for the specific case of data series but might be a subsequence within a specific dimension. For that purpose, discriminant features identification can be seen as object detection in a picture, where the object is a subsequence, and the picture is a data series.

2.6.3.2 Class Activation Map for Univariate Data Series

Class Activation Map [128] (CAM) has been proposed to highlight the parts of an image that contributed the most for a given class identification. The latter has been adapted and experimented on data series [34, 114] (univariate and multivariate). This method explains the classification of a certain deep learning model by highlighting the subsequences that contributed the most to a certain classification. One should note that the Class Activation Map method can only be used if and only if a Global Average Pooling layer has been used before the softmax classifier. Thus, only the classical architecture CNN and ResNet proposed in the literature can benefit from the Class Activation Map. As mentioned in [34, 114] one should note that this is a very novel research area (especially for data series classification) which is most of the time considered secondary for improving accuracy. As mentioned earlier, only 2 out of the 9 data series classification approaches listed above provide a method that explains the decision taken. We now elaborate on the mathematical formulation of the Class Activation Map method. Formally, let A be the result of the last convolutional layer $conv(n_f, \ell)$, which is a multivariate data series with n_f dimensions and of length n. A_m is the univariate time series for the dimension $m \in [1, n_f]$ corresponding to the m^{th} kernel.

Let $w_m^{\mathcal{C}_j}$ be the weight between the m^{th} kernel and the output neuron of class $\mathcal{C}_j \in \mathcal{C}$. Since a Global Average Pooling layer is used, then the input to the neuron of class $\mathcal{C}_j$ can be expressed by the following equation:

$$z_{\mathcal{C}_j} = \sum_m w_m^{\mathcal{C}_j} \sum_{(A_M)_i \in A_m} (A_M)_i$$

The second sum represents the averaged time series over the whole time dimension. Note that weight $w_m^{\mathcal{C}_j}$ is independent of index i (thanks to the Global Average pooling layer). Thus, $z_{\mathcal{C}_j}$ can also be written by the following equation:

$$z_{\mathcal{C}_j} = \sum_{(A_M)_i \in A_m} \sum_m w_m^{\mathcal{C}_j} (A_M)_i$$

Finally, $CAM_{\mathcal{C}_j} = [CAM_{\mathcal{C}_j,0}, ..., CAM_{\mathcal{C}_j,n}]$ that underlines the discriminative features of class $\mathcal{C}_j$ is defined as follows:

$$\forall i \in [0, n], CAM_{\mathcal{C}_j,i} = \sum_m w_m^{\mathcal{C}_j} (A_M)_i$$

As a consequence, $CAM_{\mathcal{C}_j}$ is a univariate data series where each element at index i indicates the significance of index i (regardless of the dimensions) for the classification as class $\mathcal{C}_j$. Figure 2.19 illustrates the process of computing the Class Activation Map and identifying the significant subsequences in the initial data series.

Method name	unsup.	semi-sup.	sup.	univ.	multiv.	static	streaming	complexity
Density-based								
Local Outlier Factor (Section 2.5.3.1)	✓	✓	✓	✓	✗	✓	✗	$O(\|T\|^2)$
Isolation Forest (Section 2.5.3.2)	✓	✓	✓	✓	✗	✓	✗	$O(\alpha\|T\|log(\|T\|))$
IMondrian Forest (Section 2.5.3.3)	✓	✓	✓	✓	✗	✓	✓	$O(\alpha\|T\|log(\|T\|))$
Distance-based								
STOMP (Section 2.5.4.4)	✓	✓	✓	✓	✗	✓	✗	$O(\|T\|^2)$
STAMPI (Section 2.5.4.4)	✓	✓	✓	✓	✗	✓	✓	$O(\|T\|^2)$
GrammarViz (Section 2.5.4.3)	✓	✓	✓	✓	✗	✓	✗	$O(\alpha\|T\|)$
DAD (Section 2.5.4.2)	✓	✓	✓	✓	✗	✓	✗	$O(\alpha\|T\|)$
Graph-based								
Visibility Graph (Section 2.5.5.1)	✓	✓	✓	✓	✗	✓	✗	$O(\|T\|log(\|T\|))$
Complex Network (Section 2.5.5.3)	✓	✓	✓	✓	✗	✓	✗	$O(\|T\|^2)$
Machine and Deep learning-based								
Control Chart (Section 2.6.1.1)	✗	✓	✓	✓	✓	✓	✓	-
Hidden Markov Model (Section 2.6.1.2)	✗	✓	✓	✓	✓	✓	✓	-
OCSVM (Section 2.6.1.3)	✗	✓	✓	✓	✓	✓	✓	-
LSTM (Section 2.6.1.4)	✗	✓	✓	✓	✓	✓	✗	-
GAN (Section 2.6.1.5)	✗	✓	✓	✓	✓	✓	✗	-
AE (Section 2.6.1.6)	✗	✓	✓	✓	✓	✓	✗	-
CNN (Section 2.6.2.5)	✗	✗	✓	✓	✓	✓	✗	-
ResNet (Section 2.6.2.6)	✗	✗	✓	✓	✓	✓	✗	-
InceptionTime (Section 2.6.2.7)	✗	✗	✓	✓	✓	✓	✗	-

Table 2.1: Summary and taxonomy of methods listed in this chapter. Note that none of these approaches can handle missing data points and unsynchronized data series. Preprocessing steps are thus needed.

2.6.3.3 Limitation of Class Activation Map for Multivariate Data Series

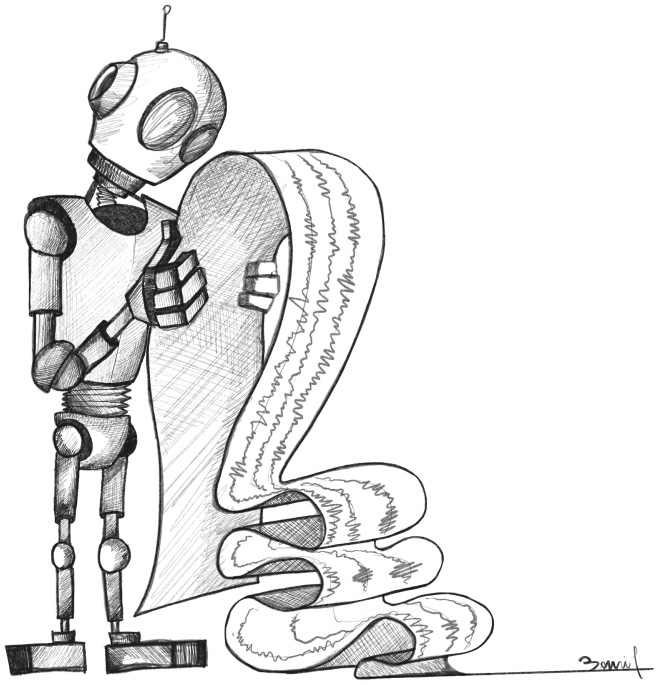
As mentioned earlier, a CAM that highlights the discriminative subsequences of class $\mathcal{C}_j$, $CAM_{\mathcal{C}_j}$, is a univariate data series. The information provided by $CAM_{\mathcal{C}_j}$ is sufficient for the case of univariate series classification. Nevertheless, the explanation provided by $CAM_{\mathcal{C}_j}$ is deficient in the specific case of multivariate data series classification. For example, even though the significant temporal index is correctly highlighted, no information can be retrieved on which dimension is significant or not. Solving this severe limitation is a significant challenge for several domains.

2.7 Summary

Table 2.1 depicts a summary of all the methods listed in this chapter and their characteristics and taxonomy described in the previous section. We included graph-based approaches, even though these methods were not introduced explicitly for anomaly detection purposes. We denote by $Complex\ Network$ the methods that are trying to build a graph based on the phase space of the data series. Finally, we report in Table 2.1 the execution time complexity to the data series length that we note $|T|$ (α is used to embed the different parameters and inner components of the methods that have a significant impact on the execution time).

Chapter 3

Unsupervised Subsequence Anomaly Detection



Chapter Outline

3.1 Limitations of Current Approaches . . . 57
3.2 Proposed Approaches: a Generic idea . . . 57
3.3 NormA: Set-based modeling of the data series . . . 58
3.3.1 Normal Model Definition . . . 58
3.3.2 Computational Steps . . . 60
3.3.3 Overall Algorithm and Complexity Analysis . . . 67
3.4 Series2Graph: Graph-based modeling of the data series . . . 68
3.4.1 Subsequence Graph Definition . . . 68
3.4.2 Computational Steps . . . 70
3.4.3 Overall Algorithm and Complexity Analysis . . . 79
3.5 Experimental Evaluation . . . 81
3.5.1 Implementation . . . 81
3.5.2 Description of the datasets . . . 82
3.5.3 Description of the evaluation metrics . . . 82
3.5.4 Description of the baselines . . . 83
3.5.5 NormA Parameters influences . . . 83
3.5.6 Series2Graph Parameters Influences . . . 85
3.5.7 Accuracy Evaluation . . . 90
3.5.8 Execution Time Evaluation . . . 91
3.5.9 User Interfaces: Stand Alone Web Applications . . . 93
3.6 Summary . . . 95

Unsupervised subsequence anomaly (or outlier) detection in long sequences is an important problem with applications in many domains. However, the approaches that have been proposed so far in the literature have severe limitations: they either require prior domain knowledge or become cumbersome and expensive to use in situations with recurrent anomalies of the same type. This chapter addresses these problems and proposes two generic approaches, NormA and Series2Graph, two novel approaches suitable for domain-agnostic anomaly detection. NormA is based on a new data series primitive, which permits to detect anomalies based on their (dis)similarity to a model that represents normal behavior. Series2Graph aims to embed the data series into a directed graph that emphasizes the unusual and potentially abnormal subsequences. The experimental results on several real datasets demonstrate that the proposed approaches correctly identify both single and recurrent anomalies of various types, with no prior knowledge of the characteristics of these anomalies (except for their length). Moreover, it outperforms the current state-of-the-art algorithms in terms of accuracy while being faster in execution time.

3.1 Limitations of Current Approaches

Anomaly, or outlier detection is an old problem, finding applications in a wide range of domains. In the specific context of sequences, we are interested in identifying anomalous subsequences. That is, the outlier is not a single value but rather a sequence of values. As described in the previous chapter, existing techniques either explicitly look for a set of pre-determined types of anomalies [44, 1], or identify as anomalies the subsequences with the largest distances to their nearest neighbors [124, 104]. We observed and illustrated in the previous chapter that these approaches pose limitations to the subsequence anomaly identification task for several reasons. We summarize below these limitations.

First, anomalous behaviors are not always known. Therefore, techniques that use specific domain knowledge for mining anomalies (e.g., in cardiology [44], and engineering [3]) involve several finely-tuned parameters and do not generalize to new cases and domains. Second, in the case of general, domain-agnostic techniques for subsequence anomaly detection, the state-of-the-art algorithms (e.g., see Section 2.5.4.3 and Section 2.5.4.4) have been developed for the case of a *single* anomaly in the dataset, or multiple different (from one another) anomalies. The reason is that these algorithms are based on the distance of a subsequence to its Nearest-Neighbor (NN) in the dataset: the subsequence that has the farthest NN is marked as an anomaly. Third, in order to remedy this situation, the m^{th} *discord* approach has been proposed (see Section 2.5.4.2). This approach takes into account the multiplicity, m, of the anomalous subsequences that are similar to one another and marks as anomalies all the subsequences in the same group by computing the m^{th} (instead of the 1^{st}) NNs for each subsequence. Nevertheless, this approach assumes that we know the multiplicity m, which is not valid in practice (otherwise, we need to re-execute the algorithms for several different m values).

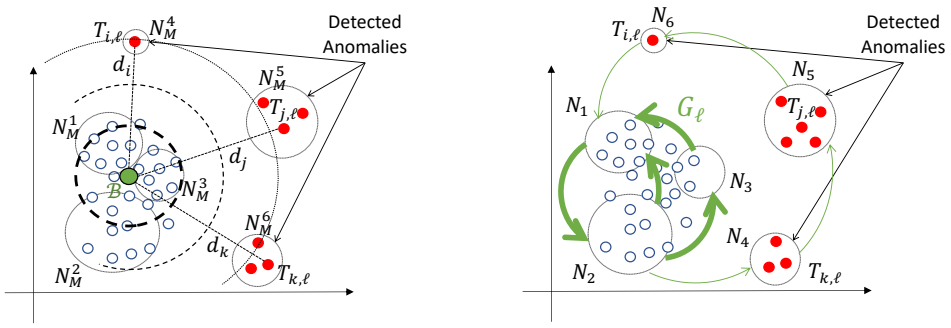
Finally, another drawback of existing unsupervised methods for subsequence anomaly detection is the execution time complexity. Discord-based methods and Local Outlier Factor methods usually require a high execution time. As unsupervised methods are commonly used for exploratory purposes, the execution time can be a significant limitation for users.

In the following section, we address the problems mentioned above and propose two generic approaches. Then, we empirically demonstrate their advantages for the specific task of unsupervised subsequence anomaly detection in accuracy and execution time.

3.2 Proposed Approaches: a Generic idea

In this section, we describe our proposed approaches to the problems of unsupervised subsequence anomaly detection in data series (as defined in Problem 1). The generic idea of our proposed approaches is to focus on the construction of a data structure that summarizes the normal behaviors of a targeted data series [11]. We abstractly define the normal behavior of the data series as follows:

Definition 13 (Normal Behavior, N_B)**.** *Given a data series T, N_B is a model that represents the normal (i.e., not anomalous) trends and patterns in T.*



(a) d defined as the distance to the Normal Model $N_M = \{(N_M^1, w^1), \ldots, (N_M^6, w^6)\}$; $\mathcal{B}$ is the weighted barycenter of N_M

(b) Normal representation of the data series as G_ℓ . The path to reach the red dots has smaller weight than those to reach the blue dots.

Figure 3.1: Illustration of the two subsequence anomaly definitions proposed approaches: (a) NormA; (b) Series2Graph.

The above definition is not precise on purpose: it allows several interpretations, leading to different kinds of models. Nevertheless, subsequence anomalies can then be defined uniformly: anomalies are the subsequences with the largest distances to the expected, normal behavior, N_B (or their distance is above a set threshold). Therefore, both of the proposed approaches will define their data structure to represent N_B, as well as their distance measure.

Overall, the problem (derived from Problem 1) we solve in this chapter is defined as follows.

Problem 1 (Subseq. Anom. Detection). *Given a data series T, and a targeted anomaly subsequence length ℓ, propose a function $f : T \rightarrow \{\mathcal{N}, \mathcal{A}\}$ decomposed into two different functions: (i) $f_1 : T \rightarrow N_B$ and (ii) $d_{N_B} : T \rightarrow \{\mathcal{N}, \mathcal{A}\}$. Overall, f returns $\mathcal{A}$, a set containing the η most abnormal subsequences of length ℓ.*

In the following sections, we describe the two proposed approach that introduce a specific data structure that enables fast and accurate detection. Figure 3.1(a) is an illustration of the data structure used by NormA (called Normal Model and defined in Section 3.3) and Figure 3.1(b) illustrates the graph data structure used by Series2Graph (defined in Section 3.4).

3.3 NormA: Set-based modeling of the data series

3.3.1 Normal Model Definition

We now present NormA. We propose a formalization for N_B, called the Normal Model, denoted N_M, and defined as follows [18, 13]:

Definition 14 (Normal Model, N_M). *N_M is defined as the following set of sequences:*

$$N_M = \{(N_M^0, w^0), (N_M^1, w^1), ..., (N_M^n, w^n)\}$$

where N_M^i is a subsequence of length ℓ_{N_M} (the same for all N_M^i) that corresponds to a recurring behavior in the data series T, and w^i is its normality score (the higher this score is, the more usual the behavior represented by N_M^i is).

In other words, this model averages (with proper weights) the different recurrent behaviors observed in the data, such that all the normal behaviors of the data series will be represented in the normal model, while unusual behaviors will not (or will have a very low weight).

Figure 3.1(a) is an illustration of a Normal Model. As depicted, the Normal Model N_M is a weighted combination of a set of subsequences (points within the dotted circles). The combination of these subsequences and their related weights returns distances d_i, d_j, d_k that are high enough to be differentiated from the normal points/subsequences. These distances can be seen as the distance between subsequences and a weighted barycenter $\mathcal{B}$ (in green) that represents N_M. Note that we do not actually compute this barycenter; we illustrate it in Figure 3.1(a) for visualization purposes. Moreover, we choose $\ell_{N_M} > \ell$ in order to make sure that we do not miss useful subsequences, i.e., subsequences with a large overlap with an anomalous subsequence. For instance, for a given subsequence of length ℓ, a normal model of length $\ell_{N_M} = 2\ell$ will also contain the subsequences overlapping with the first and last half of the anomalous subsequence.

As we have defined the data structure N_M that represents the normal behavior of a given data series T, we now define the distance to N_M which is used as anomaly score:

Definition 15 (Subsequence Anomaly Score: Distance to N_M)**.** *Assume a data series T, the set $\mathbb{T}_\ell$ of all its subsequences of length ℓ, and the Normal Model N_M of T. Then, the subsequence $T_{j,\ell} \in \mathbb{T}_\ell$ with* anomaly score*, i.e., distance to N_M, $d_{N_M}(T_{j,\ell}) = \sum_{N_M^i} w^i * min_{x \in [0, \ell_{N_M} - \ell]} \left\{ d(T_{j,\ell}, N^i_{M_{x,\ell}}) \right\}$ is an anomaly if $d_{N_M}(T_{j,\ell})$ is in the η largest distances among all subsequences in $\mathbb{T}_\ell$, or $d_{N_M}(T_{j,\ell}) > \epsilon$, where $\epsilon \in \mathbb{R}_{>0}$ is a threshold.*

Note that the only essential input parameter is the length ℓ of the anomaly (which is also one of the inputs in all relevant algorithms in the literature listed in Table 2.1). The parameter η (or ϵ) is not essential, as long as the algorithm can *rank* the anomalies. We stress that in practice, experts start by examining the most anomalous pattern and then move down in the ranked list since there is (in general) no rigid threshold separating anomalous from non-anomalous behaviors [8].

As we mentioned above and will detail later on, we choose to define N_M as a set of sequences that summarizes normality in T by representing the average behavior of a set of normal sequences. Intuitively, N_M is the set of data series which tries to minimize the sum of distances (distance function d in Definition 15) between itself and some of the subsequences in T. The Normal Model and subsequence anomaly definition are illustrated in Figure 3.2. Last but not least, we need to compute N_M in an unsupervised way, i.e., without having normal and abnormal labels for the subsequences in $\mathbb{T}_\ell$.

Observe that this definition of N_M implies the following challenge: even though N_M summarizes the normal behavior only, it needs to be computed based on T, which may

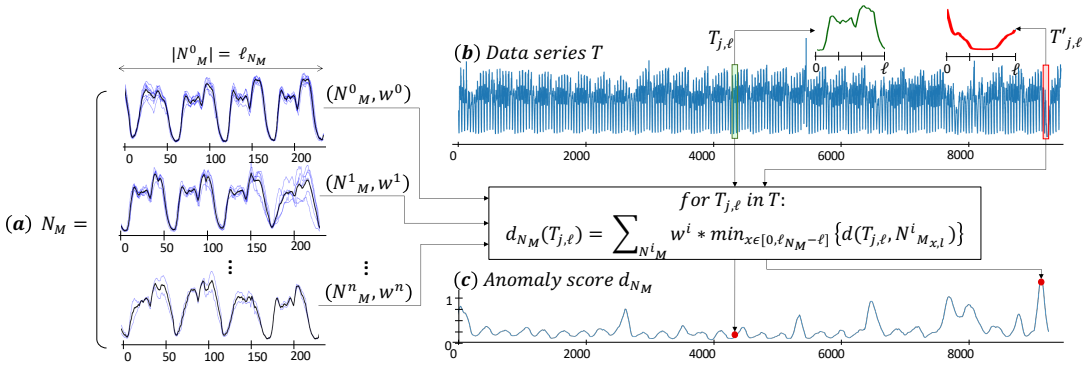


Figure 3.2: (a) Normal Model of series T (shown in (b)), composed of n cluster centroids (thick lines) of subsequences (thin lines) of T. Each subsequence of T is compared to all centroids N_M^i weighted by w^i (black box). (c) Anomaly score d_{N_M} of all subsequences of T: subsequence $T_{j,\ell}$ is normal (low score), while $T'_{j,\ell}$ is an anomaly (high score).

Symbol	Description
N_M	Normal Model of T
N_M^i	the i^{th} sequence of Normal Model of T
w^i	Normality score of N_M^i
ℓ_{N_M}	length of Normal Model N_M
$N_M^i \bowtie_\ell T$	join between N_M^i and T with subsequence length ℓ
$T \bowtie_\ell T$	self-join of T with subsequence length ℓ_{N_M}
$\mathbb{S}$	a subset of subsequences of T, of length ℓ_{N_M}
$\mathbb{C}$	a set of clusters of subsequences of length ℓ_{N_M}
c	one cluster in $\mathbb{C}$
$Center(c)$	the centroid of cluster c

Table 3.1: NormA table of symbols

include (several) anomalies. We address these challenges by taking advantage of the fact that anomalies are a minority class. Table 3.1 summarizes the symbols we use to describe NormA (and its computational steps) in the following sections.

3.3.2 Computational Steps

We now define the computational steps involved to build automatically the Normal Model N_M and the anomaly score computation based on N_M. Recall that N_M should capture (summarize) the normal behavior of the data. It may not be tough to do for a sequence T that does *not* contain any anomalous subsequences. In practice, however, we would like to apply the NormA approach in an unsupervised way on any sequence, which may contain several anomalies. The challenge is then how to compute N_M based on a sequence T that contains

anomalies, without user intervention and prior knowledge of the anomalies (except for their length), and then identify the anomalous subsequences in this same sequence T.

Note that the N_M length, ℓ_{N_M}, is larger than the anomaly length ℓ, so that we do not miss subsequences that have a large overlap with an anomalous subsequence: given a subsequence of length ℓ, if we choose a normal model of length $\ell_{N_M} = 2\ell$, it will contain the subsequences overlapping with the first and last half of the anomalous subsequence, which is desirable.

NormA approach (N_M extraction and anomaly score computation) is thus composed of the following four steps.

1. **Sampling of Subsequences:** We extract the subsequences, which can serve as candidates for building the N_M (blue subsequences in Figure 3.2(b)).
2. **Subsequences Clustering:** We group these subsequences according to their similarity, adopting a hierarchical clustering strategy, augmented by automated identification of the right number of clusters, based on the Minimum Description Length principle (the centroids of the resulting clustering are illustrated by the black data series in Figure 3.2(b)).
3. **Cluster Scoring:** We then score the clusters computed in the previous step. Finally, we set the Normal model $N_M = \{(N_M^0, w^0), (N_M^1, w^1)..., (N_M^n, w^n)\}$, with N_M^i the centroid of the i^{th} cluster, and w^i its score.
4. **Anomaly Score Computation:** We finally score each subsequence in the data series (illustrated in Figure 3.2(c)) and extract the most anomalous subsequences (illustrated as the red subsequences $T'_{j,\ell}$ in Figure 3.2).

We now elaborate on these N_M computational steps and the anomaly score computation.

3.3.2.1 Sampling of Subsequences

Remember that we are interested in describing the normal behavior of a system. Hence, we need to identify the subsequences (of the data series in which we wish to detect anomalies) that occur approximately the same along with the data series. These subsequences are a form of recurrent patterns and should represent the normal behavior. Good candidate subsequences are those that satisfy the following properties: (i) they are similar to one another (normal behavior repeats approximately the same); (ii) they cover a large percentage of the data (not all extracted from the same part of the series); and (iii) they have high cardinality (appear frequently in the series).

In order to discover groups of *recurrent patterns* we adopt a strategy that groups similar subsequences without knowing beforehand their range and frequency. Since subsequence clustering has high time and memory complexity, considering every possible subsequence of a large input data series would not be a suitable solution, both in execution time efficiency

and in accuracy [58]. We thus decide to ignore some subsequences [96] and select only a subset of them in the original data series.

We describe two variations of our candidate subsequence selection strategy, one motif-based strategy and one random selection strategy. In the first strategy, we select subsequences from T that have high similarity to T (excluding overlapping subsequences). To that extent, we sort the subsequences of T according to the distances to their $1^{st}NN$ in T. We can achieve this with the self-join (defined in Section 2.5.4.4). For each position i in the data series T, the self-join sequence contains the nearest neighbor distance of the subsequence $T_{i,\ell}$ (an example is shown in Figure 2.9). Given the self-join of T, we can discard the isolated occurrences, namely, the subsequences that do not have a close match and thus have the highest self-join values.

Given an input data series T and its self-join ($T \bowtie_\ell T$), we define the set of the clustering candidate patterns (subsequences), $\mathbb{S}^{selfjoin}$, selected by means of the self-join:

Definition 16 (Motif Set: $\mathbb{S}^{selfjoin}$)**.** *Given a data series T and a subsequence length ℓ, we define $\mathbb{S}^{selfjoin}$ as:*

$$\mathbb{S}^{selfjoin} = \{T_{i,\ell_{N_M}} | 1 \leq i \leq |T| - \ell_{N_M} + 1 \wedge (T \bowtie_\ell T)_i < \epsilon\} \tag{3.1}$$

where $\epsilon \in \mathbb{R}^+$. Moreover, if $T_{i,\ell_{N_M}}, T_{j,\ell_{N_M}} \in \mathbb{S}^{selfjoin} \implies |i-j| \geq \ell_{N_M}$.

The $\mathbb{S}^{selfjoin}$ set contains non-overlapping subsequences of T which are not isolated occurrences.

In the second selection strategy, we use a random sampling strategy. Even though random motif selection could be performed [70], we decide to use uniform random sampling as a first baseline. We sample from T a subset of non-overlapping subsequences, generating the candidate set as follows:

Definition 17 (Random Set: $\mathbb{S}^{sample}$)**.** *Given a data series T, a subsequence length ℓ_{N_M}, and a sampling rate $0 < r < 1$, we define $\mathbb{S}^{sample}$ as:*

$$\mathbb{S}^{sample} = \{T_{i,\ell_{N_M}} | 0 \leq i \leq |T - \ell_{N_M} + 1|\} \tag{3.2}$$

such that $|\mathbb{S}^{sample}| < r * |\mathbb{T}_{\ell_{N_M}}|/\ell_{N_M}$. Moreover, if $T_{i,\ell_{N_M}}, T_{j,\ell_{N_M}} \in \mathbb{S}^{sample} \implies |i-j| \geq \ell_{N_M}$.

In $\mathbb{S}^{sample}$, we place the subsequences that are randomly chosen until we reach the maximum size of $|\mathbb{S}^{sample}|$ that respects the constraint in Definition 17. Thanks to the uniform distribution of the random sampling, the subsequences in $\mathbb{S}^{sample}$ also cover the entire length of the data series T.

Note that in the optimal case, where T is a *periodic* data series, we know that there are at most $|\mathbb{T}_{\ell_{N_M}}|/\ell_{N_M}$ non-overlapping recurrent patterns, assuming that ℓ_{N_M} is the length of

the period. We thus consider this value as an upper bound for the $\mathbb{S}^{selfjoin}$ cardinality. This value also represents the maximum number of fixed length cycles occurring in an *aperiodic* data series. Among the datasets we consider in the empirical evaluation the maximum value of $|\mathbb{T}_{\ell_{N_M}}|/\ell_{N_M}$ corresponds to the *1.3%* of $|\mathbb{T}_{\ell_{N_M}}|$. Moreover, we notice that setting the threshold $\epsilon = \mu(T \bowtie_\ell T)$ in $\mathbb{S}^{selfjoin}$ always allows to filter isolated subsequences in T.

3.3.2.2 Clustering Step

At this point, we are ready to present the adopted clustering technique to group subsequences in $\mathbb{S}$ ($\mathbb{S}^{selfjoin}$, or $\mathbb{S}^{sample}$). In that regard, we consider their complete-linkage (dendrogram), resulting from the agglomerative hierarchical clustering [21]. Following previous work, we select a dendrogram cut by applying the *Minimum Description Length* principle [97, 96].

We define description length as the total number of bits used to represent a subsequence, namely its *entropy*. Given a data series T, we measure its entropy $H(T)$ as:

$$H(T) = -\sum_{i=1}^{|T|} P(T = T_{i,1}) log_2 P(T = T_{i,1}) \tag{3.3}$$

The notation $P(T = T_{i,1})$, denotes the probability of finding the value $T_{i,1}$ in T. The description length DL of T is then defined as $DL(T) = |T| * H(T)$, and quantifies the storage requirement of a sequence. It is minimized, when a data series contains the highest number of repeated values. In this case, bits compression reduces the space.

Once the subsequences are grouped, we can represent them by using their distances to the cluster centers. If the clustering is optimal, we expect the sequences to have high similarity to their cluster centers. We consider the subsequences at the clustering stage in their SAX form (Symbolic Aggregate approXimation), where each real value is assigned a discrete label [106].

We introduce the conditional description length of a data series T (that quantifies the bits needed to store it), when knowing its cluster center sequence $\mathcal{C}$:

$$DL(T|\bar{\mathcal{C}}) = DL(T - \bar{\mathcal{C}}) \tag{3.4}$$

Given a cluster of subsequences, $\mathcal{C}$ (with the centroid $\bar{\mathcal{C}}$), we compute the conditional cluster description length DLC, namely the number of bits used to encode the cluster using its center:

$$DLC(\mathcal{C}|\bar{\mathcal{C}}) = DL(\bar{\mathcal{C}}) + \sum_{T \in \mathcal{C}} (DL(T|\bar{\mathcal{C}})) \tag{3.5}$$

where the non-conditional $DLC(\mathcal{C}) = \sum_{T \in \mathcal{C}} (DL(T))$. Given a set of clusters $\mathbb{C}$, in order to quantify the compression achieved by $\mathbb{C}$, we compare the bits needed to store all the subsequences, with and without knowing $\bar{\mathcal{C}}$. We thus apply the $bitsave$ measure:

$$bitsave(\mathbb{C}) = \sum_{\mathcal{C} \in \mathbb{C}} DLC(\mathcal{C}) - DLC(\mathcal{C}|\bar{\mathcal{C}}) \tag{3.6}$$

Algorithm 1: Subsequences Clustering

input : subsequences set $\mathbb{S}$
output: a cluster set $\mathbb{C}$

```
1  Dendrogram ← CompleteLinkage(𝕊);
2  ℂ ← ∅;
3  lastBitsave ← −∞;
4  foreach cut in Dendrogram in top-down order do
5      ℂ′ ← get subsequences clusters from cut;
6      if bitsave(ℂ′) > lastBitsave then
7          ℂ ← ℂ′;
8          lastBitsave ← bitsave(ℂ′);
9      else
10         break;
11     end
12 end
```

In Algorithm 1, we report the clustering procedure, which selects and outputs the clusters of a dendrogram cut. The subsequences linkage is computed in Line 1. Subsequently, we iterate over the cuts in a top-down manner (Line 4). Therefore, we start by considering the cuts that produce the least number of clusters. We expect that the highest bitsave is attained by grouping subsequences in the smallest amount of groups if cluster intra-similarity is maximized. Hence, we iterate the cuts until their clusters $bitsave$ stops to increase (Line 6). We thus pick the clusters resulting from the last encountered cluster. This permits to group the subsequences, maximizing their *similarity* and *frequency*.

3.3.2.3 Weights Computations

Each cluster we compute in Algorithm 1 becomes the candidate group of subsequences (candidate cluster) that are considered to build the Normal Model. We now propose a scoring function, which permits to compute w^i (that can be seen as the normality degree) for each candidate cluster i. Intuitively, the cluster and subsequences with the top score are the most representative of the *different*, *recurring* patterns in the entire data series; the next cluster is less representative (but still contains subsequences that are close to normal behavior).

Let $\mathbb{S} \subseteq \mathbb{T}_{\ell_{N_M}}$ be a subset of subsequences in T of length ℓ_{N_M}. We can then compute the *coverage* of $\mathbb{S}$, $Coverage(\mathbb{S}) = MaxOffset(\mathbb{S}) - MinOffset(\mathbb{S})$, which measures the distance between the maximum and minimum offsets in T (of two $\mathbb{S}$ subsequences), and corresponds to the span of T from where the subsequences in $\mathbb{S}$ were extracted. We will also refer to the *frequency* of $\mathbb{S}$, $Frequency(\mathbb{S}) = |\mathbb{S}|$ (equal to the cardinality of $\mathbb{S}$).

Moreover, we want to consider an *inter-clustering* property, namely the *centrality*. We borrow this definition from the graph analysis literature [117], which states that the most central node in a graph denotes its influence. Given a cluster set $\mathbb{C}$ and a cluster $\mathcal{C} \in \mathbb{C}$, we

define *centrality* as:

$$Centrality(\mathcal{C}, \mathbb{C}) = \frac{1}{\sum_{\mathcal{C}_i \in \mathbb{C}} d(\bar{\mathcal{C}}, \bar{\mathcal{C}}_i)} \tag{3.7}$$

Recall that a cluster of subsequences, denoted by $\mathcal{C}$, formally coincides with a set of subsequences $\mathbb{S}$. The $Center$ function we adopt in our work is the *centroid*, which is the arithmetic mean vector of the subsequences in a cluster c.

Intuitively, to set the weights w^i for all clusters i, we need to consider the subsequences that most often occur along the largest part of the data. It translates to identifying the cluster with the highest frequency and the broadest coverage. In order to account for the most recurrent subsequence, we also adopt the centrality measure. If a subsequence is the most recurrent, we expect that all its occurrences are grouped in the cluster with the highest centrality.

We are now ready to score the candidate clusters, taking into account the *frequency* and *coverage* of the subsequences in each cluster, and its *centrality* as well. After normalizing $Frequency(\mathcal{C})$, $Coverage(\mathcal{C})$, and $Centrality(\mathcal{C})$ so that each lies in the $[1, 2]$ interval for all $\mathcal{C} \in \mathbb{C}$ (normalization is needed so that all three criteria have equal weight), the score we assign to a cluster $\mathcal{C}$, given also the complete clusters set $\mathbb{C}$, is the following:

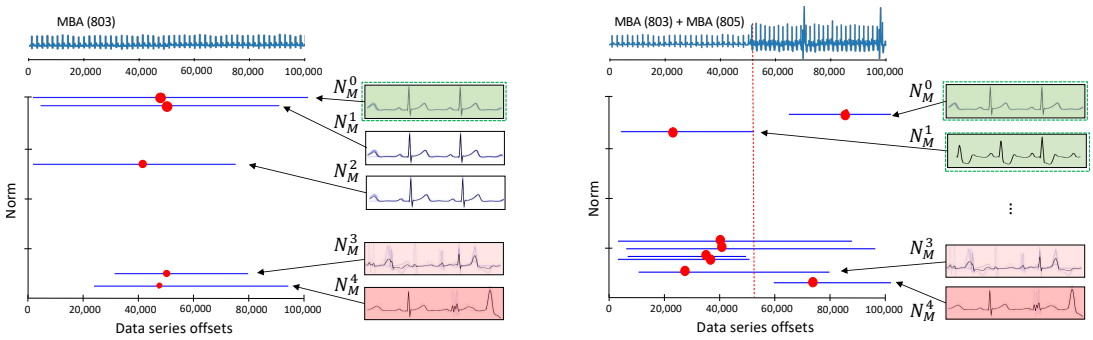
$$Norm(\mathcal{C}, \mathbb{C}) = Frequency(\mathcal{C})^2 \times Coverage(\mathcal{C}) \times Centrality(\mathcal{C}, \mathbb{C}) \tag{3.8}$$

The $Norm$ function provides an index with regards to the Normal Model properties we take into consideration. Since high *coverage* values might erroneously be assigned to clusters with low *frequency*, we favor clusters that have high *frequency*. For this reason, it appears squared in Equation 3.8.

3.3.2.4 Normal Model Extraction

In Figure 3.3(a), we report the cluster scores we obtain for the MBA ECG recordings (patient 803). In the plot, we report each cluster $Norm$ score (the size of the red point is proportional to $Frequency(\mathcal{C})$) coupled with their *coverage* (blue line), which starts and ends respectively at the smallest and largest offset of the cluster subsequences. In the right part of Figure 3.3, we depict the subsequences in each cluster. The x-axis value assigned to each red point is the arithmetic mean of its subsequences offsets in the corresponding cluster. This set of clusters $\mathbb{C} = \{\mathcal{C}_0, ..., \mathcal{C}_n\}$ will be used in the Normal Model $N_M = \{(N_M^0, w^0), ...(N_M^n, w^n)\}$, with $N_M^i = Center(\mathcal{C}_i)$ and $w^i = Norm(\mathcal{C}_i, \mathbb{C})$.

In this example, the subsequences in the cluster with the highest $Norm$ score represent correct Heartbeat Ventricles contracts. The centroid of this cluster will be the most influential in N_M. On the other hand, clusters with low scores contain subsequences that do not represent any known features (they may be noise or even repeated anomalies) and, therefore, will not have a real influence in N_M.



(a) Data series containing one single type of normal subsequences

(b) Data series containing two types of normal subsequences

Figure 3.3: (a) $Norm$ cluster scoring of MBA ECG recordings (patient 803); (b) $Norm$ cluster scoring of the concatenation of two MBA ECG recordings (patient 803 and 805).

3.3.2.5 Anomaly Score Computation

We now discuss the problem of how to identify the anomalous subsequences in a series T, assuming that we have computed the Normal Model $N_M = \{(N_M^0, w^0), (N_M^1, w^1)..., (N_M^n, w^n)\}$. Remember that N_M (ideally) represents the expected, normal behavior of the data. Intuitively, the anomalous subsequences are the ones that are far away from most of the subsequences in N_M.

Our technique starts by considering the pairwise distances between each subsequence of length ℓ in T to subsequences of the same length in each of N_M^i in N_M. For each subsequence N_M^i in N_M, this operation results in a meta-sequence, $N_M^i \bowtie_\ell T$ (the *join* sequence defined in Definition 10), that contains at position j the nearest neighbor distance between subsequence $T_{j,\ell}$ and any subsequence of the same length, ℓ, in N_M.

We compute all the join sequences $N_M^i \bowtie_\ell T$ (with $(N_M^i, w^i) \in N_M$), which contain the distances between each subsequence of T and their nearest neighbor in N_M^i. As described in Definition 15, we then compute the anomaly score for each subsequence. This score corresponds to the nearest neighbor distance between the subsequence to score and all the subsequences in each N_M^i in N_M. Given a subsequence $T_{j,\ell}$, we retrieve the nearest neighbor distance between $T_{j,\ell}$ and every $N_M^i \in N_M$, $(N_M^i \bowtie_\ell T)_j$. We then weigh these distances with w_i and sum them. Formally, for each subsequence in $T_{i,j}$, the anomaly score is computed as:

$$d_{N_M}(T_{j,\ell}) = \sum_{(N_M^i, w^i) \in N_M} w^i (N_M^i \bowtie_\ell T)_j \tag{3.9}$$

These scores represent the degree of abnormality: the larger the score is, the more abnormal the subsequence is. We then have to extract the k subsequences of length ℓ, which have the highest scores and rank them.

Algorithm 2: NormA

input : data series T, Normal Model length ℓ_{N_M}, subsequence length ℓ
output: Normal Model N_M and list for $Anomalies$

// Compute the Normal Model N_M
1 compute $\mathbb{S}^{selfjoin}$ (or $\mathbb{S}^{sample}$) from T;
// compute the set of subsequences clusters in T ($\mathbb{C}$)
2 $\mathbb{C} \leftarrow SubsequencesClustering(\mathbb{S}, \ell_{N_M})$;
3 $N_M \leftarrow \{\}$;
4 **for** $\mathcal{C}$ *in* $\mathbb{C}$ **do**
5 | add $(centroid(\mathcal{C}), Norm(\mathcal{C}, \mathbb{C}))$ in N_M;
6 **end**
// Compute the anomaly score for each subsequence in T
7 $allJoin \leftarrow []$;
8 **foreach** (N_M^i, w^i) *in* N_M **do**
9 | $allJoin \leftarrow N_M^i \bowtie_\ell T$;
10 **end**
11 $AnomalyScore\ d \leftarrow []$;
12 **foreach** $j \in [0, |T| - \ell]$ **do**
13 | $d \leftarrow \sum_{(N_M^i, w^i) \in N_M} w^i join[j]$;
14 **end**
15 $Anomalies \leftarrow$ subsequences with *top-k* d values;
16 $Anomalies \leftarrow$ sort subsequences in $Anomalies$ in order of decreasing values in d

3.3.3 Overall Algorithm and Complexity Analysis

The overall procedure for computing the Normal Model is then structured as shown in Algorithm 2. In Line 1, we select a subset of subsequences, $\mathbb{S}$, applying one of the two strategies we discussed earlier (i.e., $\mathbb{S}^{selfjoin}$, or $\mathbb{S}^{sample}$), which take into consideration several desired characteristics of the correct (non-anomalous) part of the data. Subsequently, we cluster them in Line 2. In Line 4, we iterate each cluster that is assigned to the $Norm$ score (Line 5) and then added to the Normal Model as a tuple composed of its centroid and its score. The assigned score quantifies how much a group of similar subsequences (cluster) supports the properties we define over correct data. We use NormA-SJ to refer to the algorithm that uses $\mathbb{S}^{selfjoin}$, and NormA-smpl for the variation with $\mathbb{S}^{sample}$.

3.3.3.1 Normal Model construction complexity

The complexity of the first section of Algorithm 2 depends on the choice of the subsequence selection strategy performed in the initial part. We can compute $\mathbb{S}^{selfjoin}$, using the state-of-the art algorithm *Stomp* [129] in $O(|T|^2)$ time. On the other hand, computing $\mathbb{S}^{sample}$ takes linear time in the worst case ($O(|T|)$). In the experimental evaluation, we test the

two selection strategies in isolation to assess their accuracy separately. Subsequently, the subsequences linkage computation takes $O(\ell_{N_M} * |\mathbb{S}|^2)$.

It is important to note that the space of $|\mathbb{S}|$ is in general two orders of magnitude smaller than the original space of T. In turn, selecting a dendrogram cut has a worst-case time complexity of $O(\ell_{N_M} * |\mathbb{S}|^2)$, when all the cuts need to be evaluated. As we show in the experimental evaluation, the number of cuts considered in Algorithm 2 is very small in practice.

3.3.3.2 Score computation complexity

The complexity of the second section of Algorithm 2 is defined by the computation of $N_M^i \bowtie_\ell T$, which is bounded by $O(|T| * \ell_{N_M} * |N_M|)$, where $|N_M|$ is the number of subsequences in N_M (remember that $|N_M| << |T|$). Therefore, the anomalies extraction step is negligible, and the complexity is $O(|T| * \ell_{N_M} * |N_M|)$.

3.3.3.3 Overall complexity

To conclude the overall complexity in the worst case is $O(\ell_{N_M}(|\mathbb{S}|^2 + |T| * |N_M|))$ for NormA-smpl and $O(|T| * 2)$ for NormA-SJ. As one can notice, the complexity of NormA-SJ is significantly worse than NormA-smpl. We will demonstrate the advantage in execution time of NormA-smpl in Section 3.5.8.

3.4 Series2Graph: Graph-based modeling of the data series

We now present an alternative data structure to represent the subsequences normal behaviors of the data series. The previous normal model was a set of subsequences that aimed to store both normal and abnormal subsequences in the same set, associated with weights that rank them based on their normality. One can argue that ordering information is missing from this data structure representation. We thus formulate an approach for subsequence anomaly detection based on the data series representation into a Graph, in which edges encode the ordering information [12]. Figure 3.1(b) illustrates the Graph data structure.

3.4.1 Subsequence Graph Definition

We now provide a new formulation for subsequence anomaly detection. The idea is that a data series is transformed into a sequence of abstract states (corresponding to different subsequence patterns), represented by nodes $\mathcal{N}$ in a directed graph, $G(\mathcal{N}, \mathcal{E})$, where the edges $\mathcal{E}$ encode the number of times one state occurred after another. Thus, normality can be characterized by (i) the edge weight, which indicates the number of times two subsequences occurred one after the other in the original sequence, and (ii) the node degree, the

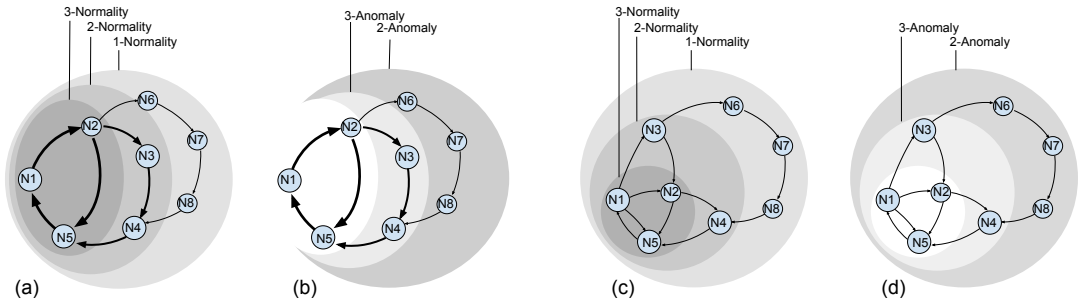


Figure 3.4: 3-Normality, 2-Normality, 1-Normality, and 3-Anomaly, 2-Anomaly for two given graphs ((a),(b) and (c),(d)) representing the simplified model of two data series. Edge weights and node degrees are used to define the θ-Normality and θ-Anomaly subgraphs.

number of edges adjacent to the node, which indicates the proximity of the subsequences in that node to other subsequences. Note that G is a connected graph (there exists a path between every pair of nodes), and thus, the degree of each node is at least equal to 1.

Under this formulation, paths in the graph composed of high-weight edges and high-degree nodes correspond to normal behavior. As a consequence, the normality of a data series can be defined as follows.

Definition 18 (θ-Normality). *Let a node set be defined as $\mathcal{N} = \{N_1, N_2, ..., N_m\}$. Let also a data series T be represented as a sequence of nodes $\langle N^{(1)}, N^{(2)}, ..., N^{(n)} \rangle$ with $\forall i \in [0, n], N^{(i)} \in \mathcal{N}$ and $m \leq n$. The θ-$Normality$ of T is the subgraph $G^{\nu}_{\theta}(\mathcal{N}_{\nu}, \mathcal{E}_{\nu})$ of $G(\mathcal{N}, \mathcal{E})$ with $\mathcal{E} = \{(N^{(i)}, N^{(i+1)})\}_{i \in [0,n-1]}$, such that: $\mathcal{N}_{\nu} \subset \mathcal{N}$ and:*

$$\forall (N^{(i)}, N^{(i+1)}) \in \mathcal{E}_{\nu}, w((N^{(i)}, N^{(i+1)})).(deg(N^{(i)}) - 1) \geq \theta \tag{3.10}$$

An example of θ-Normality subgraph is shown in Figures 3.4(a) and (c). In Figure 3.4(a), the subgraph composed of nodes N_1, N_2, N_5, has edges with weights larger than 3, and a minimum node degree of 2. Therefore, it is a 3-$Normality$ subgraph. In Figure 3.4(c), the subgraph composed of nodes N_1, N_2, N_5, has edges of weight 1, but does not have any node with a degree under 4. Therefore, it is a 3-$Normality$ subgraph. Similarly, we define an anomaly as follows.

Definition 19 (θ-Anomaly). *Let a node set be defined as $\mathcal{N} = \{N_1, N_2, ..., N_m\}$. Let a data series T be represented as a sequence of nodes $\langle N^{(1)}, N^{(2)}, ..., N^{(n)} \rangle$ with $\forall i \in [0, n], N^{(i)} \in \mathcal{N}$ and $m \leq n$. The θ-$Anomaly$ of T is the subgraph $G^{\alpha}_{\theta}(\mathcal{N}_{\alpha}, \mathcal{E}_{\alpha})$ of $G(\mathcal{N}, \mathcal{E})$ with $\mathcal{E} = \{(N^{(i)}, N^{(i+1)})\}_{i \in [0,n-1]}$, such that:*

$$G^{\nu}_{\theta}(\mathcal{N}_{\nu}, \mathcal{E}_{\nu}) \cap G^{\alpha}_{\theta}(\mathcal{N}_{\alpha}, \mathcal{E}_{\alpha}) = \emptyset \tag{3.11}$$

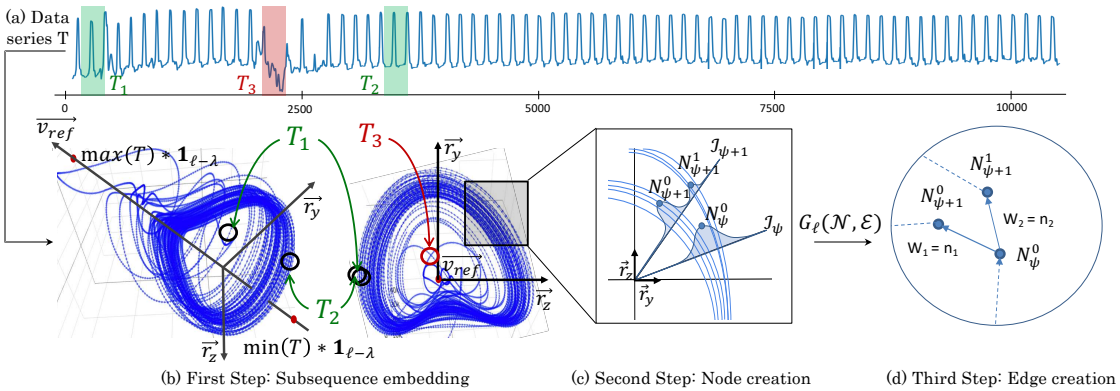


Figure 3.5: Series2Graph steps in order to build the graph from a data series (a): embed the subsequences (b), create the nodes (c), and extract the edges (d).

An example of θ-$Anomaly$ subgraph is outlined in Figures 3.4(b) and (d). In Figure 3.4(b), the nodes that do not belong to the 3-$Normality$ subgraph constitute the 3-$Anomaly$ subgraph. The 2-$Anomaly$ subgraph is included in the 3-$Anomaly$ subgraph, and the intersection of the 2-$Anomaly$ and 2-$Normality$ subgraphs is empty. Similar observations hold for Figure 3.4(d). We now define the membership criteria of a subsequence to a θ-Normality subgraph.

Definition 20 (θ-Normality Membership)**.** *Given a data series T represented as a sequence of abstract states $\langle N^{(1)}, N^{(2)}, ..., N^{(n)} \rangle$, a subsequence $T_{i,\ell}$, represented by $\langle N^{(i)}, N^{(i+1)}, ..., N^{(i+\ell)} \rangle$, belongs to the θ-$Normality$ of T if and only if $\forall j \in [i, i+\ell], (N^{(j)}, N^{(j+1)}) \in \theta\text{-}Normality(T)$. On the contrary, $T_{i,\ell}$ belongs to the θ-$Anomaly$ of T if and only if:*

$$\exists j \in [i, i+\ell], (N^{(j)}, N^{(j+1)}) \notin \theta\text{-}Normality(T) \tag{3.12}$$

Based on the above definitions, using θ-Normality subgraphs naturally leads to a ranking of subsequences based on their "normality". For practical reasons, this ranking can be transformed into a score, where each rank can be seen as a threshold in that score. We elaborate on this equivalence in the following section. Observe also that the subsequence length is not involved in the definition of normal/abnormal, which renders this approach more general and flexible. Note that given the existence of graph G, the above definitions imply a way for identifying the anomalous subsequences. The problem is now how to construct this graph. Table 3.2 summarizes the symbols we use in this paper.

3.4.2 Computational Steps

In this section, we describe Series2Graph, one of our two unsupervised solution to the subsequence anomaly detection problem. For a given data series T, the overall Series2Graph process is divided into four main steps as follows:

Symbol	Description
ℓ_G	Graph subsequence length
$\mathcal{N}, \mathcal{E}$	set of nodes and edges
$G(\mathcal{N}, \mathcal{E})$	directed graph corresponding to T
θ	density layer (for normality/anomaly)
θ-$Normality$	subgraph of G (also called G_θ^ν)
θ-$Anomaly$	subgraph of G (also called G_θ^α)
$\mathcal{N}_\nu, \mathcal{E}_\nu$	set of nodes and edges of θ-$Normality$
$\mathcal{N}_\alpha, \mathcal{E}_\alpha$	set of nodes and edges of θ-$Anomaly$
$w(e)$	weight of edge $e \in \mathcal{E}$
$deg(N_i)$	degree of node $N_i \in \mathcal{N}$
$Proj$	set of all embedded subsequences
$Proj_r$	reduced set $Proj$ of three dimensions
$SProj$	rotated $Proj_r$
ψ	angle
$\mathbf{\Psi}$	angle set
$\mathcal{I}_\psi$	radius set of angle ψ
$\mathcal{N}_\psi$	node set in $\mathcal{I}_\psi$

Table 3.2: Series2Graph table of symbols.

1. **Subsequence Embedding**: Project all the subsequences (of a given length ℓ_G) of T in a two-dimensional space, where shape similarity is preserved (as illustrated in Figure 3.5(a)).

2. **Node Creation**: Create a node for each one of the densest parts of the above two-dimensional space. These nodes can be seen as a summarization of all the major patterns of length ℓ that occurred in T (as illustrated in Figure 3.5(b)).

3. **Edge Creation**: Retrieve all transitions between pairs of subsequences represented by two different nodes: each transition corresponds to a pair of subsequences, where one occurs immediately after the other in the input data series T. We represent transitions with an edge between the corresponding nodes. The weights of the edges are set to the number of times the corresponding pair of subsequences was observed in T (as illustrated in Figure 3.5(c)).

4. **Subsequence Scoring**: Compute the normality (or anomaly) score of a subsequence of length $\ell \geq \ell_G$ (within or outside of T), based on the previously computed edges/nodes and their weights/degrees.

We note that the length ℓ_G required in the first step of the method is user-defined but is independent of the length of the subsequences that we want to detect as anomalies, which can have different lengths. For a targeted anomaly length ℓ, we set by default $\ell_G = 2/3 * \ell$ (we evaluate this choice in Section 3.5.6). Below, we describe in detail each one of the above steps.

3.4.2.1 Subsequences Embedding

We first describe our approach for projecting a data series into a two-dimensional space. We propose a new shape-based embedding, such that two subsequences similar in shape will be geometrically close in the transformed space after the embedding. In order to achieve this, we (i) extract all the subsequences and represent them as vectors, (ii) reduce the dimensionality of these vectors to three dimensions (that we can visualize in a three-dimensional space), (iii) rotate the space of these vectors (i.e., subsequences) such that two of the components contain the shape related characteristic, and the last one the average value. As a result, two subsequences with similar shapes but a very different mean value (i.e., small Z-normalized Euclidean distance, but large Euclidean distance) will have very close values for the first two components but very different for the third one.

We start by extracting subsequences using a sliding window that we slide by one point at a time. Note that this is equivalent to using a *time delay embedding* [55] with a delay $\tau = 1$. We then apply a local convolution (of size $\lambda = \ell_G/3$) to each subsequence to reduce noise and highlight the important shape information. Formally, for a given subsequence length ℓ_G and local convolution size $\lambda = \ell_G/3$, we transform subsequence T_{i,ℓ_G} into a vector (of size $\ell_G - \lambda$):

$$\left[\sum_{k=i}^{i+\lambda} T_k, \sum_{k=i+1}^{i+1+\lambda} T_k, ..., \sum_{k=i+\ell_G-\lambda}^{i+\ell_G} T_k\right] \tag{3.13}$$

We insert the vectors corresponding to all subsequences T_{i,ℓ_G} in matrix $Proj(T, \ell_G, \lambda) \in \mathbb{M}_{|T|,\ell_G-\lambda}(\mathbb{R})$, where $\mathbb{M}$ is the set of real-valued matrices with $|T|$ rows and $\ell_G - \lambda$ columns.

In order to reduce the dimensionality of matrix $Proj(T, \ell_G, \lambda)$, we apply a Principal Component Analysis (PCA) transform. For the sake of simplicity, we keep only the first three components (PCA_3), and denote the reduced three-column matrix as $Proj_r(T, \ell_G, \lambda)$.

We note that using the first three components was sufficient for our analysis. Consider that for the 25 datasets used in our experimental evaluation (See Section 3.5), the three most important components explain on average 95% of the total variance. Generalizing our solution to a larger number of important components is considered as future works.

Since we are interested in subsequence anomalies, which correspond to anomalous shapes (trends), we need to emphasize (out of the three components obtained by the aforementioned reduced projection) the components that explain the most the shape of the subsequences. Let $min(T)$ and $max(T)$ be the minimum and maximum values of the data series T. We extract the vector $\vec{v}_{ref} = \overrightarrow{O_{mn}O_{mx}}$, where $O_{mn} = PCA_3(min(T) * \lambda * \mathbf{1}_{\ell-\lambda})$ and $O_{mx} = PCA_3(max(T) * \lambda * \mathbf{1}_{\ell-\lambda})$ (PCA_3 returns the three most important components using the trained PCA applied on $Proj(T, \ell, \lambda)$). Intuitively, the vector $\vec{v}_{ref}$ describes the time dimension along which the values change (bounded by $\lambda * min(T)$ and $\lambda * max(T)$, where the multiplication with λ corresponds to a local convolution). The other dimensions (orthogonal vectors of $\vec{v}_{ref}$) describe how the values change. Thus, overlapping points/sequences in these other dimensions indicate recurrent behaviors, and isolated

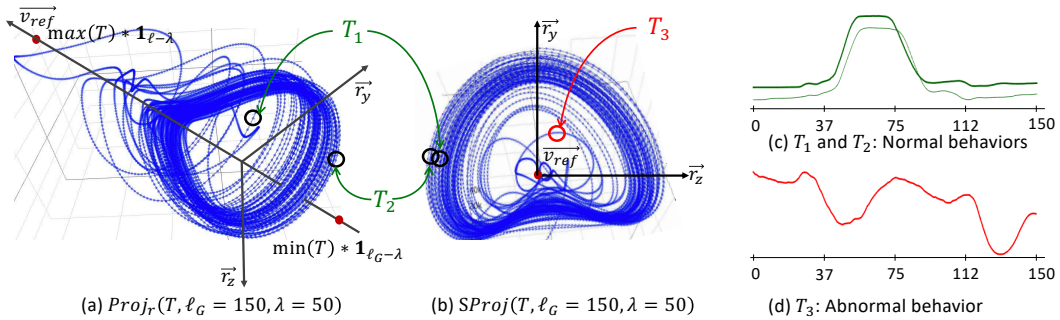


Figure 3.6: (a) $Proj_r(T, \ell_G, \lambda)$ and (b) $SProj(T, \ell_G, \lambda)$ of a data series T corresponding to the movement of an actor's hand that (c) takes a gun out of the holster and points to a target (normal behavior); (d) the anomaly (red subsequence) corresponds to a moment when the actor missed the holster [59]. We rotate (a) into (b) such that $\vec{v}_{ref}$ is invariant in two dimensions.

points/sequences indicate possible anomalies. Given the unit vectors $(\vec{u}_x, \vec{u}_y, \vec{u}_z)$ that represent the axes of the cartesian coordinate system of the PCA, the angle $\phi_x = \angle\vec{u}_x\vec{v}_{ref}$, $\phi_y = \angle\vec{u}_y\vec{v}_{ref}$, $\phi_z = \angle\vec{u}_z\vec{v}_{ref}$ and their corresponding rotation matrices $R_{u_x}(\phi_x)$,$R_{u_y}(\phi_y)$ and $R_{u_z}(\phi_z)$, we define $SProj(T, \ell, \lambda)$ as follows:

$$SProj(T, \ell_G, \lambda) = R_{u_x}(\phi_x)R_{u_y}(\phi_y)R_{u_z}(\phi_z)Proj_r(T, \ell_G, \lambda)^T \tag{3.14}$$

The matrix $SProj(T, \ell_G, \lambda)$ is the reduced projection $Proj_r(T, \ell_G, \lambda)$ rotated in order to have the unit vector $\vec{u}_x$ aligned with the offset vector $\vec{v}_{ref}$.

Figure 3.6 depicts the rotation procedure to transform $Proj_r$ into $SProj$ for an example data series T that corresponds to the movement of an actor's hand that takes a gun out of the holster and points to a target (normal behavior). This rotation is using vector $\vec{v}_{ref}$, defined by the minimal and maximal constant sequences mentioned earlier (marked with the red dots in Figure 3.6(a)). The unit vectors of the rotated space are $(\frac{\vec{v}_{ref}}{||\vec{v}_{ref}||}, \vec{r}_y, \vec{r}_z)$, where $\vec{r}_y$ and $\vec{r}_z$ are the rotated vectors $\vec{u}_y$ and $\vec{u}_z$.

What this rotation achieves is that (similarly to Z-normalization) subsequences with a different mean but the same shape in the space before the rotation (e.g., subsequences T_1 and T_2 in Figure 3.6(c)) will have very close $\vec{r}_y$ and $\vec{r}_z$ components in the new coordinate system (as shown in Figures 3.6(a) and (b)). Therefore, subsequences with similar shapes will appear close together, shapes that often repeat in the dataset will form dense clusters in the space (like subsequences T_1 and T_2), and rare shapes (anomalies) will appear relatively isolated (like subsequence T_3). Figures 3.6(c) and (d) depict the normal (T_1 and T_2) and abnormal (T_3) subsequences. The anomaly (T_3) corresponds to a case when the actor missed the holster [59].

We observe that in the rotated space (see Figure 3.6(b)), the shape differences are easy to distinguish, and the normal behavior (dense clusters of repeated patterns) and anomalies (isolated patterns) are clearly separated.

Algorithm 3: Pattern Embedding

input : Data series T, input length ℓ_G, λ
output: 3-dimensional points sequence $SProj$

// Transform first subsequence
$P \leftarrow \left(\sum_{k=j}^{j+\lambda} T_k \right)_{j \in [0, \ell_G - \lambda]}$;
add P in $Proj$;
// Transform every other subsequences in T
foreach $i \in [1, |T| - \ell_G]$ **do**
  $P[0 : \ell_G - \lambda - 1] \leftarrow P[1 : \ell_G - \lambda]$;
  $P[\ell_G - \lambda] \leftarrow \sum_{k=i+\ell_G-\lambda}^{i+\ell_G} T_k$;
  add P in $Proj$;
end
// Reduce to three dimensions
$pca \leftarrow PCA_3.fit(Proj)$;
$Proj \leftarrow pca.transform(Proj)$;
// Get rotation characteristics
$v_{ref} \leftarrow pca.transform((max(T) - min(T)) * \lambda * \mathbf{1}_{\ell_G - \lambda})$;
$\phi_x, \phi_y, \phi_z \leftarrow getAngle((u_x, u_y, u_z), v_{ref})$;
$R_{u_x}, R_{u_y}, R_{u_z} \leftarrow GetRotationMatrices(\phi_x, \phi_y, \phi_z)$;
// Rotate SProj
$SProj \leftarrow R_{u_x}.R_{u_y}.R_{u_z}.Proj^T$

In the rest of this section, $SProj(T, \ell_G, \lambda)$ will refer to the 2-dimensional matrix keeping only the r_y and r_z components. Algorithm 3 describes the computation of the pattern embeddings. Finally, it is important to note that this embedding space is useful to detect abnormal subsequences because of their shapes (which is due to the dropped component $\vec{v}_{ref}$ that contained the information in the mean values of the subsequences).

3.4.2.2 Node Creation

At this point, we are ready to extract shape-related information, as in Figure 3.6, where recurrent and isolated trajectories can be distinguished. The idea is to extract the most *crossed* sections of the 2-dimensional space defined by the unit vector $(\vec{r}_y, \vec{r}_z)$. These sections will be the nodes in the graph we want to construct. First, we define the *radius subset*.

Definition 21 (Radius Set)**.** *Given a data series T and its projection matrix $P = SProj(T, \ell_G, \lambda)$, the radius set $\mathcal{I}_\psi$ is the set of intersection points between the vector $\vec{u}_\psi = cos(\psi)\vec{r}_y + sin(\psi)\vec{r}_z$ and every segment $[x_{i-1}, x_i]$, where x_{i-1}, x_i are two consecutive rows of P:*

$$\mathcal{I}_\psi = \left\{ x \middle| (\vec{u}_\psi \times \vec{x} = \vec{0}) \wedge (\overrightarrow{x_{i-1}x} \times \overrightarrow{x_{i-1}x_i} = \vec{0}) \right\}$$

where $\times$ operator is the cross product.

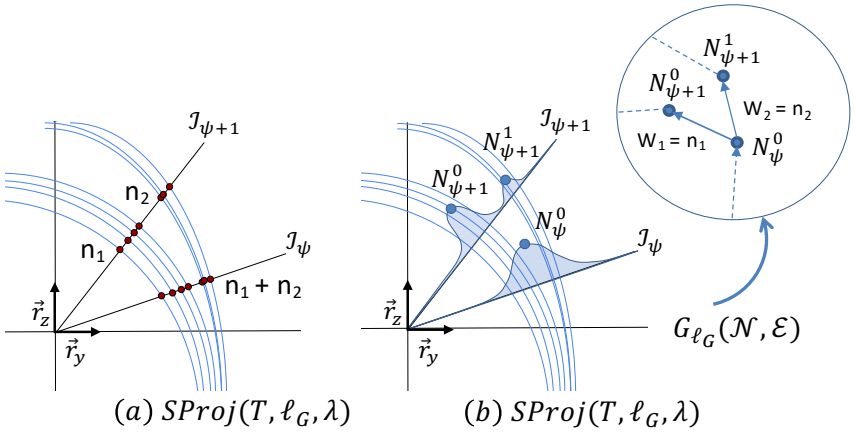


Figure 3.7: Node extraction from $SProj(T, \ell_G, \lambda)$ by measuring the density of the intersected trajectories to a given vector. The densest points are added to the Node Set $\mathcal{N}$.

Figure 3.7 (a) displays two radius subsets (marked with the red points). We can now define the *Pattern Node Set* as follows.

Definition 22 (Pattern Node Set). *Given a data series T, its projection $P = SProj(T, \ell_G, \lambda)$ and a set of $\mathcal{I}_\psi$ ($\psi \in \mathbf{\Psi}$), the Pattern Node Set of T is:*

$$\begin{aligned} &\mathcal{N} = \cup_{\psi \in \mathbf{\Psi}} \mathcal{N}_\psi \\ &\mathcal{N}_\psi = \left\{ x \middle| \exists \epsilon, \forall y, |x - y| > \epsilon \implies f_h(x, \mathcal{I}_\psi) > f_h(y, \mathcal{I}_\psi) \right\} \\ &\textit{with } f_h(x, \mathcal{I}_\psi) = \frac{1}{nh\sqrt{2\pi\sigma(\mathcal{I}_\psi)^2}} \sum_{x_i \in \mathcal{I}_\psi} e^{\frac{(x - x_i - h\mu(\mathcal{I}_\psi))^2}{2h\sigma(\mathcal{I}_\psi)^2}} \end{aligned} \tag{3.15}$$

In the above definition, f is a kernel density estimation function applied on a radius subset using a Gaussian distribution. Then, nodes become the areas in the 2-dimensional space, where the trajectories of the patterns are the most likely to pass through. In other words, each node corresponds to a set of very similar patterns. The bandwidth parameter h affects the granularity of the extraction. The smaller the h value is, the more local maxima, and therefore the more nodes we will end up with. Moreover, the larger the h value is, the fewer nodes the graph will have, and therefore the more general it will be. We define parameter $r = |\mathbf{\Psi}|$ as the number of angles ψ that we use in order to extract the pattern node set. In other words, this parameter is sampling the space (refer to Algorithm 4, Line 1). Once again, many angles will lead to high precision, but at the cost of increased computation time.

In practice, we observed that parameter r is not critical, and we thus set $r = 50$. We demonstrate the latter statement in Section 3.5.6. Regarding the bandwidth parameter of the density estimation, we set it following the Scott's rule [102]: $h_{scott} = \sigma(\mathcal{I}_\psi).|\mathcal{I}_\psi|^{-\frac{1}{5}}$. In accordance with the use case, a better parametrization of h and r might differ. However, we observe that this fixed bandwidth returned strong results, and we empirically confirm its pseudo optimality in Section 3.5.6.

Algorithm 4: Node Extraction

input : 2-dimensional point sequence $SProj$, rate r, bandwidth h
output: Node Set $\mathcal{N}$

```
   // Set the number of radius
1  Ψ ← (i 2π/r)_{i∈[0,r]};
2  N ← {};
3  foreach ψ ∈ Ψ do
4      I_ψ ← [];
5      foreach i ∈ [0, |SProj| − 1] do
           // Find intersected points
6          radius ← max_{x,y}(SProj_i, SProj_{i+1});
7          P_ψ ← (radius_x.cos(ψ), radius_y.sin(ψ));
8          add Intersect((Ω, P_ψ), (SProj_i, SProj_{i+1})) in I_ψ;
9      end
       // Extract Nodes
10     N_ψ ← argmax_{x∈I_ψ} f_h(x, I_ψ);
11     add N_ψ in N;
12 end
```

Algorithm 4 outlines the above process for extracting the node set from $SProj$. For example, in Lines 6-8, we compute for each radius subset $\mathcal{I}_\psi$ all intersection points between a radius vector and the possible segments composed of two consecutive points in $SProj$.

3.4.2.3 Edge Creation

Once we identify the nodes, we need to compute the edges among them. Recall that the set of extracted nodes corresponds to all the possible states, where subsequences of the data series T can be. In order to compute the transitions among these nodes, we loop through the entire projection $SProj(T, \ell_G, \lambda)$ and we extract the sequence $\langle N^{(0)}, N^{(1)}, ..., N^{(n)} \rangle$ of the nodes N_i in $\mathcal{N}$ that the embedded subsequences $(SProj(T, \ell_G, \lambda)_0, ..., SProj(T, \ell_G, \lambda)_n)$ belong to. Intuitively, the above node sequence involves all the nodes in $\mathcal{N}$ (some of them more than once) and represents the entire input data series. We use this sequence to identify the edges of the graph G_{ℓ_G} we want to construct. In practice, we extract the edges (all the pairs of successive nodes in the above sequence) and set their weights to the number of times the edge is observed in the sequence. Formally, the edges set $\mathcal{E}$ is defined as follows.

Definition 23 (Pattern Edges Set). *Given a data series T, its projection $P = SProj(T, \ell_G, \lambda)$ and its Pattern Node Set $\mathcal{N}$, the edges set $\mathcal{E}$ is equal to:*

$$\mathcal{E} = \left\{ \left(S(P_i), S(P_{i+1}) \right) \right\}_{i \in [1, |P|-1]}, \tag{3.16}$$

where function S finds for a given projection point, the closest node in $\mathcal{N}$. Formally:

$$S(x) = argmin_{n \in \mathcal{N}} d(x, n) \tag{3.17}$$

Algorithm 5: Edge Extraction

input : 2-dimensional points sequence $SProj$, and a node set $\mathcal{N}$
output: a Edge Set $\mathcal{E}$

$\mathbf{\Psi} \leftarrow \left(i\frac{2\pi}{r}\right)_{i\in[0,r]}$;
$NodeSeq \leftarrow []$;
foreach $i \in [0, |SProj| - 1]$ **do**
    // Get the two radius that bound SProj$_i$ and SProj$_{i+1}$
    $\psi_i \leftarrow getAngle(\vec{u_x}, SProj_i)$;
    $\psi_{i+1} \leftarrow getAngle(\vec{u_x}, SProj_{i+1})$;
    foreach $(\psi \in \mathbf{\Psi}) \wedge (\psi \in [\psi_i, \psi_{i+1}])$ **do**
        // Fill the sequence of node NodeSeq
        $radius \leftarrow max_{x,y}(SProj_i, SProj_{i+1})$;
        $P_\psi \leftarrow (radius_x.cos(\psi), radius_y.sin(\psi))$;
        $x_{int} \leftarrow Intersect((\Omega, P_\psi), (SProj_i, SProj_{i+1}))$;
        $n_{int} \leftarrow argmin_{n\in\mathcal{N}_\psi}(|x_{int} - n|)$;
        add n_{int} in $NodeSeq$;
    end
    // Extract edges from NodeSeq
    $\mathcal{E} \leftarrow \left\{(NodeSeq_i, NodeSeq_{i+1})\right\}_{i\in[0,|fullPath|]}$;
end

where $x \in P$ and d is the Euclidean distance.

Since the weight of each edge is equal to the cardinality of this edge in the edge set $\mathcal{E}$, this weight is proportional to the number of times two subsequences follow each other in the input data series. For efficiency, $S(x)$ is computed as follows: for a given projection point, we first find the node subset N_ψ of $\mathcal{N}$ (with $\psi \in \mathbf{\Psi}$), such that $|\angle\vec{x}\vec{u_\psi}|$ is minimal. We then compute $S(x)$ such as $S(x) = argmin_{n\in\mathcal{N}_\psi}|x.\vec{u_\psi} - n|$, where $\vec{x}.\vec{u_\psi}$ is the scalar product between $\vec{x}$ and $\vec{u_\psi}$. As depicted in Figures 3.7(a) and (b), a total of $n_1 + n_2$ subsequences are intersected by $\mathcal{I}_\psi$ and represented by node N^0_ψ. At $\mathcal{I}_{\psi+1}$, these subsequences are divided between nodes $N^0_{\psi+1}$ (n_1 subsequences) and $N^1_{\psi+1}$ (n_2 subsequences). Therefore, we have $w(N^0_\psi, N^0_{\psi+1}) = n_1$ and $w(N^0_\psi, N^1_{\psi+1}) = n_2$. Algorithm 5 outlines the steps we follow to extract the edges among the nodes in $\mathcal{N}$.

3.4.2.4 Anomaly Score Computation

We now describe how we can use the information in the graph to identify the normal and anomalous behaviors.

We start with the conversion of a subsequence to a path in a given graph. For an already computed graph $G_{\ell_G}(\mathcal{N}, \mathcal{E})$, we define function $Time2Path(G_{\ell_G}, T_{i,\ell})$ that converts a subsequence $T_{i,\ell}$ into a path (i.e., a sequence of nodes) in G_{ℓ_G}, by (i) computing the pattern embedding SP of $T_{i,\ell}$ (using the PCA transformation and rotation matrices, com-

puted in Lines 8 and 12, respectively, of Algorithm 3), and (ii) extracting the edges using $EdgeExtraction(SP, \mathcal{N})$ (output of Algorithm 5) on the Node Set $\mathcal{N}$ of graph G_{ℓ_G}.

We are now ready to measure normality. As mentioned earlier, modeling a data series using a cyclic graph results in the graph encoding information based on the recurrence of subsequences. Then, the path normality score function can be defined as follows.

Definition 24 (Path Normality Score). *Given a data series T and its graph $G_{\ell_G}(\mathcal{N}, \mathcal{E})$, and a subsequence $T_{i,\ell}$ of length $\ell \geq \ell_G$, the normality of a path $P_{th} = Time2Path(G_{\ell_G}(\mathcal{N}, \mathcal{E}), T_{i,\ell}) = \langle N^{(i)}, N^{(i+1)}, ..., N^{(i+\ell)} \rangle$ is equal to:*

$$Norm(P_{th}) = \sum_{j=i}^{i+\ell-1} \frac{w(N^{(j)}, N^{(j+1)})(deg(N^{(j)}) - 1)}{\ell} \tag{3.18}$$

We can thus infer a normality score for subsequences in T using the $Time2Path$ function defined earlier (the opposite of this normality score is the anomaly score). Formally, the normality score is defined as follows.

Definition 25 (Subsequence Normality Score). *Given a data series T, its graph $G_{\ell_G}(\mathcal{N}, \mathcal{E})$ and a subsequence $T_{i,\ell}$ of length $\ell \geq \ell_G$, the $Normality$ score $T_{i,\ell}$ is equal to:*

$$Normality(T_{i,\ell}) = Norm(Time2Path(G_{\ell_G}(\mathcal{N}, \mathcal{E}), T_{i,\ell})) \tag{3.19}$$

Observe that the two previous definitions are consistent with the definition of θ-Normality, such that every P_{th} in θ-Normal subgraph will have $N(P_{th}) \geq \theta$, and every P_{th} that is exclusively in a lower normality level will have $N(P_{th}) \leq \theta$. As a matter of fact, the rank generated by the normality score is similar to the θ-$Normality$ rank, and in both rankings, the anomalies are found at the bottom of the ranking. The following lemma formalizes this statement.

Lemma 1. *Given a data series T, its graph $G_{\ell_G}(\mathcal{N}, \mathcal{E})$, a subsequence $T_{i,\ell}$, and its path $P_{th} = Time2Path(G_{\ell_G}(\mathcal{N}, \mathcal{E}), T_{i,\ell})$, we have: $\forall \theta \in \mathbb{N}_{>0}, N(P_{th}) < \theta \implies P_{th} \in \theta$-Anomaly$(T)$*

Proof. Consider a subsequence $T_{i,\ell}$ corresponding to $P_{th} = \langle N^{(i)}, N^{(i+1)}, ..., N^{(i+\ell)} \rangle$. If $P_{th} \in \theta$-Normality(T), then according to Definition 20, we have:

$$\begin{aligned} &\forall j \in [i, i+\ell-1], (N^{(j)}, N^{(j+1)}) \in \theta\text{-Normality}(T) \\ \implies &\forall j \in [i, i+\ell-1], w(N^{(j)}, N^{(j+1)}).(deg(N^{(j)}) - 1) \geq \theta \\ \implies &\sum_{j=i}^{i+\ell-1} \frac{w(N^{(j)}, N^{(j+1)}).(deg(N^{(j)}) - 1)}{\ell} \geq \theta \end{aligned}$$

As a consequence, $P_{th} \notin \theta$-Normality(T) and according to Definition 20, $P_{th} \in \theta$-Anomaly(T). □

Algorithm 6: Series2Graph

input : data series T, input length ℓ_G, subsequence length to analyze ℓ
output: a data series $NormaliltyScore$

```
// Embedding step described in Alg. 3
1 SProj ← PatternEmbedding(T, ℓ_G, λ);
// Node creation step described in Alg. 4
2 N ← NodeExtraction(SProj, r = 50, h = h_opt);
// Edge creation step described in Alg. 5
3 E ← EdgeExtraction(SProj, N);
4 G_ℓG ← Graph(N, E);
// Normality scores computation
5 NormalityScore ← [0]_{0,|T|−ℓ};
// compute Normality score for all subsequences of length ℓ in T
6 foreach i ∈ [1, |T| − ℓ] do
7 |   NormalityScore[i] ← Norm(Time2Path(G_ℓG, T_{i,ℓ}));
8 end
9 NormalityScore ← movingAverage(NormalityScore, ℓ_G)
```

Therefore, the subsequences of T with a low score are those that compose the θ-*Anomaly* subgraph, where the value of θ is low (close to one for the discords). We note that this process identifies both single anomalies (discords) and recurrent anomalies.

3.4.3 Overall Algorithm and Complexity Analysis

Algorithm 6 summarizes all the steps of our approach. In Lines 1-3, we compute the subsequence embedding, the Node Set $\mathcal{N}$ and then the Edge Set $\mathcal{E}$ in order to build the graph G_{ℓ_G}. Line 7 computes the $NormalityScore$ for all subsequences of the input data series: we use a sliding window over the input data series to extract all subsequences, we score each one of them and store the result in the vector $NormalityScore$, initialized in Line 5. Finally, we apply a moving average filter (of window length ℓ_G) on the $NormalityScore$ vector (Line 9). This filter tries to rectify possible small inaccuracies of the scoring function by ensuring that two highly overlapping subsequences will have similar $NormalityScores$ (as we would normally expect).

We now describe the time complexity of the different computational steps. We refer in this section to Algorithms 3, 4, and 5.

3.4.3.1 Embedding Step Complexity

Algorithm 3 describes the computation of the pattern embeddings. A naive solution is to compute all the convolutions for all the subsequences of T, which leads to a complexity of magnitude $O(|T| * \ell_G * \lambda)$. Nevertheless, by using the previously computed convolu-

tions (Line 4), the complexity is reduced to $O(|T| * \lambda)$. The PCA step of Algorithm 3 is implemented with a randomized truncated Singular Value Decomposition (SVD), using the method of Halko et al. [45] with time complexity of $O(|T|*(\ell_G-\lambda)*|component|)$. The last step consists of matrix multiplications, and therefore has a complexity of $O(|T|*|R_{u_x}|^2)$. In our case, the size of the rotation matrices are much smaller than λ, which leads to a global complexity of $O(|T| * (3 * (\ell_G - \lambda)))$.

3.4.3.2 Node Creation step complexity

Algorithm 4 outlines the above process for extracting the node set from $SProj$. In Lines 6-8, we compute for each radius subset $\mathcal{I}_\psi$ all intersection points between a radius vector and the possible segments composed of two consecutive points in $SProj$. The complexity of this operation is bounded by $O(|SProj|*r) \simeq O(|T|*r)$. The time complexity of the kernel density estimation is $O(|\mathcal{I}_\psi|)$ (since $|\mathcal{I}_\psi| \leq |SProj|$). Actually, we experimentally observed that $|\mathcal{I}_\psi| << |SProj|$. Therefore, the overall time complexity is bounded by $O(|T| * r)$. We can improve this complexity using the following observation. Instead of checking the intersection with every possible radius, we can select those that bound the position of the points i and $i+1$ in $SProj$ (only the radius with ψ between $\psi_i = \angle\vec{u_x}.\vec{SProj_i}$ and $\psi_{i+1} = \angle\vec{u_x}.\vec{SProj_{i+1}}$). Therefore, the worst case complexity becomes $O(|T| * r)$, and the best case complexity is reduced to $O(|T|)$. We thus denote the complexity as $O(|T| * \alpha_\mathcal{N})$ with $\alpha_\mathcal{N} \in [1, r]$.

3.4.3.3 Edge Creation step complexity

Algorithm 5 outlines the steps we follow to extract the edges among the nodes in $\mathcal{N}$. For each point in the input data series T, we identify the radius it belongs to, and we choose the closest node. Therefore, the complexity is bounded by $O(|T|)$ and varies based on the number of radius we have to check and the number of nodes in each $\mathcal{N}_\psi$. The former is bounded by parameter r: on average, we have no more than $|T|/r$ points per $\mathcal{N}_\psi$. The overall complexity is in the worst case $O(|T|^2)$, and in the best case, $O(|T|)$. We note that this worst-case corresponds to the situation where each subsequence in T belongs to a different node. It is not what we observe in practice: the overall complexity is close to the best case for all our datasets. We thus denote the complexity as $O(|T| * \alpha_\mathcal{E})$ with $\alpha_\mathcal{E} \in [1, |T|]$.

3.4.3.4 Score computation complexity

Two elements are necessary to compute the score for a given subsequence of length ℓ. We first need to compute the degree of each node. The latter can be achieved in $O(|\mathcal{N}| + |\mathcal{E}|)$ and needs to be computed only once. As we observe in practice that both $|\mathcal{N}|, |\mathcal{E}|$ are significantly smaller than $|T|$, this operation is negligible compared to the other computational steps. Then we have two cases: (i) the subsequence to score is inside the data series used to build the graph, or (ii) the subsequence is not in the data series.

For the first case (i), we just need to find the sequence of nodes that corresponds to the subsequence (we can store this sequence in-memory) and the computation of the score is linear to the number of nodes in the corresponding path. For a given subsequence $T_{i,\ell}$ of T, we have $1 < |Time2Path(G_{\ell_G}, T_{i,\ell})| < \ell - \ell_G$. Thus the complexity to score every subsequence in T is bounded by $O(|T| * (\ell - \ell_G))$.

For the second case (ii), we need to match first the new subsequence in the embedding space. As we first need to compute the convolutions over the subsequence, this can be achieved in $O(\ell * \lambda)$ (the PCA projection and the rotation is negligible compared to this operation). We then need to match the points in the embedded space to existing nodes. It can be achieved in $O(\ell)$. Then the scoring of the corresponding path is bounded by $O(|T| * \ell)$, which is negligible compare to the previous operations. Thus the complexity to score a new subsequence is $O(\ell * \lambda)$.

3.4.3.5 Overall complexity

In total, the construction of G_{ℓ_G} and the subsequences scoring for a data series T can be achieved in $O(|T| * (3(\ell_G - \lambda) + \alpha_{\mathcal{N}} + \alpha_{\mathcal{E}} + (\ell - \ell_G)))$. As mentioned earlier, $\alpha_{\mathcal{E}}$ is closer from 1 than $|T|$ and $\alpha_{\mathcal{N}}$ varies between 1 and r, the overall complexity depends mostly on the size of the data series.

3.5 Experimental Evaluation

In this section, we empirically evaluate our proposed approaches compared to state-of-the-art techniques on a benchmark composed of real and synthetic data series. We first describe the details of the implementations of our proposed approaches. We then enumerate the datasets used in this section (and also used in the following chapters). We also define the accuracy and execution time measure used to compare different methods. We then compare NormA and Series2Graph with state-of-the-art approaches for anomaly detection accuracy. We demonstrate the shortcomings of discord-based approaches on data series containing similar anomalies. Finally, we measure the execution time of NormA and Series2Graph in comparison with state-of-the-art algorithms. Overall, we demonstrate that NormA and Series2Graph outperform both execution time and accuracy of current state-of-the-art approaches.

3.5.1 Implementation

3.5.1.1 Technical Details

We implemented our algorithms in C (for NormA), compiled with gcc 5.4.0, and Python 3.6 (for NormA and Series2Graph). The evaluation was conducted on a server with Intel Xeon CPU E5-2650 2.20GHz and 250GB RAM. We implemented our two methods as pip packages.

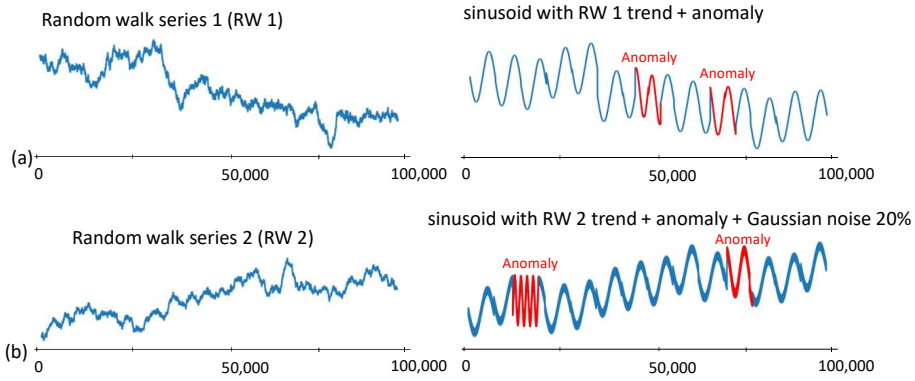


Figure 3.8: Synthetic datasets. (a) Random walk sequence (left), and sinusoid signal following the same trend (right) with injected anomalies (red/bold subsequences). (b) A second example, with 20% of Gaussian noise added on top.

3.5.2 Description of the datasets

We benchmark our approaches using real and synthetic datasets, for all of which a ground truth of annotated anomalies is available (Table 3.3). Following previous work [103], we use several synthetic datasets that contain sinusoid patterns at fixed frequency following a random walk trend (Figure 3.8). We then randomly inject different numbers of anomalies, in the form of sinusoid waveforms with different phases and higher than normal frequencies (Figure 3.8(a)), and add various levels of Gaussian noise on top (Figure 3.8(b)). We refer to those datasets using the label SRW-[# of anomalies]-[% of noise]-[length of anomaly] and use them in order to test the performance of the algorithms under different, controlled conditions.

Our real datasets are the following. Simulated engine disks data (SED) from the NASA Rotary Dynamics Laboratory [2] representing disk revolutions recorded over several runs (3K rpm speed). MIT-BIH Supraventricular Arrhythmia Database (MBA) [40, 83], which are electrocardiogram recordings from 5 patients, containing multiple instances of two different kinds of anomalies. Important information related to the aforementioned datasets are listed in Table 3.3.

3.5.3 Description of the evaluation metrics

We use the precision-at-k (Precision@k or P@k) accuracy measure to evaluate the effectiveness of the methods. The latter is the ratio of correctly identified anomalies in the η subsequences corresponding to the η highest anomaly score. (This corresponds to precision on the anomaly class $TP_A/(TP_A + FP_A)$, where TP_A is the number of detected true anomalies, and FP_A the number of false positives.) Note that this parameter η is only used for evaluation purposes and is not required for practical usage. In our accuracy evaluation, we set η to the number of anomalies in the sequence ($\eta = N_A$ of Table 3.3). Recall that

Datasets	Length	ℓ	$\mathbf{N_A}$	Field
Annotated				
SED	100K	75	50	Electronic
MBA (803)	100K	75	62	Cardiology
MBA (805)	100K	75	66	Cardiology
MBA (806)	100K	75	27	Cardiology
MBA (820)	100K	75	76	Cardiology
MBA (14046)	100K	75	142	Cardiology
SRW-[20-100]-[0%]-[200]	100K	200	var.	Synthetic
SRW-[60]-[5%-25%]-[200]	100K	200	60	Synthetic
SRW-[60]-[0%]-[100-1600]	100K	var.	60	Synthetic

Table 3.3: List of dataset characteristics: series length, anomaly length (ℓ), number of annotated anomalies ($\mathbf{N_A}$), field.

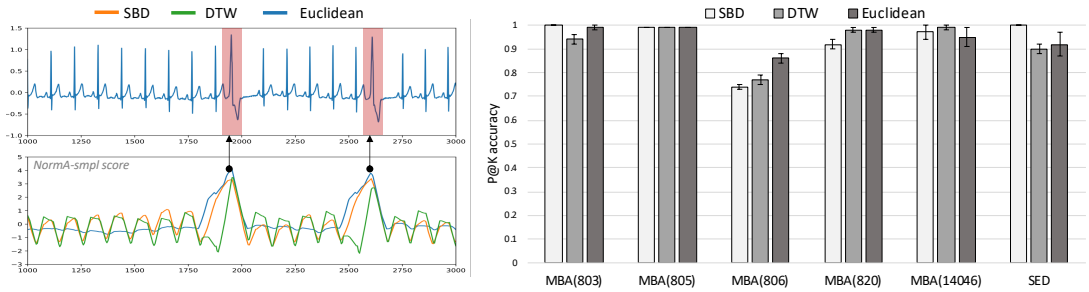
the annotated datasets we use in this work have *all* their anomalies annotated. We also measure time in order to evaluate the efficiency and scalability of the methods.

3.5.4 Description of the baselines

We compare NormA and Series2Graph to the current state-of-the-art algorithms for anomaly detection in data series. We consider two techniques that enumerate $Top\text{-}k$ 1^{st} discords, GrammarViz (named GV and described in Section 2.5.4.3) and STOMP (described in Section 2.5.4.4). Moreover, we compare NormA against the Disk Aware Discord Discovery algorithm (named DAD and described in Section 2.5.4.2), which finds m^{th} discords. We also compare to Local Outlier Factor (named LOF and described in Section 2.5.3.1) and Isolation Forest (named IF and described in Section 2.5.3.2). These two methods are not specific to subsequence anomaly detection but constitute strong baselines from the literature on multi-dimensional data outlier detection. Finally, we include in our comparison LSTM-AD (described in Section 2.6.1.4), a *semi-supervised* deep learning technique. Note that the comparison to LSTM-AD is not fair to all the other techniques: LSTM-AD has to first train on labeled normal data, which gives it an unfair advantage; all the other techniques are *unsupervised*. We include it to indicate how the unsupervised techniques compare to a state-of-the-art semi-supervised anomaly detection algorithm. In practice, we train LSTM-AD on the longest subsequence without anomalies: 4109-10846 points (7000 on average).

3.5.5 NormA Parameters influences

In this section, we evaluate the sensitivity of the Normal Model N_M, as a function of its length ℓ_{N_M} and distance function d (relevant for NormA-SJ and NormA-smpl), and of the sampling rate r (relevant for NormA-smpl).



(a) NormA-smpl result for MBA(803) using SBD, DTW and Euclidean distances

(b) NormA-smpl accuracy on MBA datasets using SBD, DTW and Euclidean distances

Figure 3.9: Distance measure impact experiment. (a) NormA-smpl accuracy score for MBA(803) for sbd, DTW and $Euclidean$ distances. (b) Overall accuracy for all the MBA datasets.

3.5.5.1 Distance Functions d

We now evaluate the impact of the distance measure used in the NormA framework (we use NormA-smpl as our baseline). As explained in Section 2.5.1, different distance measures can be used. For this purpose, we use as the d function of Definition 15 the $Euclidean$ distance (i.e., the default distance measure for our proposed method), the *Shape-Based Distance* (SBD), and the *Dynamic Time Warping* (DTW) distance (all defined in Section 2.5.1).

Figure 3.9(a) depicts the NormA-smpl score for the three distance measures for a 6000 points snippet of the MBA(803). In Figure 3.9(b), we depict the averaged accuracy (we use Precision@k accuracy and define it in Section 3.5.3) results from over ten different runs for the SED and all the MBA datasets (more details on these datasets are provided in Section 3.5.2). The results show that the SBD, DTW, and $Euclidean$ distances lead to similar results (with no clear winner).

Overall, $Euclidean$ distance provides accurate results. Moreover, through the use of the MASS algorithm [124] it is significantly faster than the other two distance measures. For the remainder of our work, we thus use this distance as function d in Definition 15.

3.5.5.2 Normal Model subsequences length ℓ_{N_M}

We now measure the performance for Precision@k anomaly detection, setting k equal to the number of anomalies contained in each one of our six real annotated datasets with multiple anomalies, and we vary the length of the Normal Model (ℓ_{N_M}), using a multiplicative factor ranging between 1.1-10 times the anomalous pattern length ℓ. Figure 3.10(b,c) shows Precision@k for each Normal Model length we tested using both NormA-smpl and NormA-SJ (the results for NormA-smpl are averages over 10 runs).

We observe that the accuracy values become stable once the Normal Model length is at least 2.5x larger than the anomaly length. Before $a = 2$, we observe, as intuitively explained in Section 3.3.1, a significant accuracy drop. We also note that this behavior is the same for both NormA-SJ and NormA-smpl, and absolute accuracy values are in both cases almost

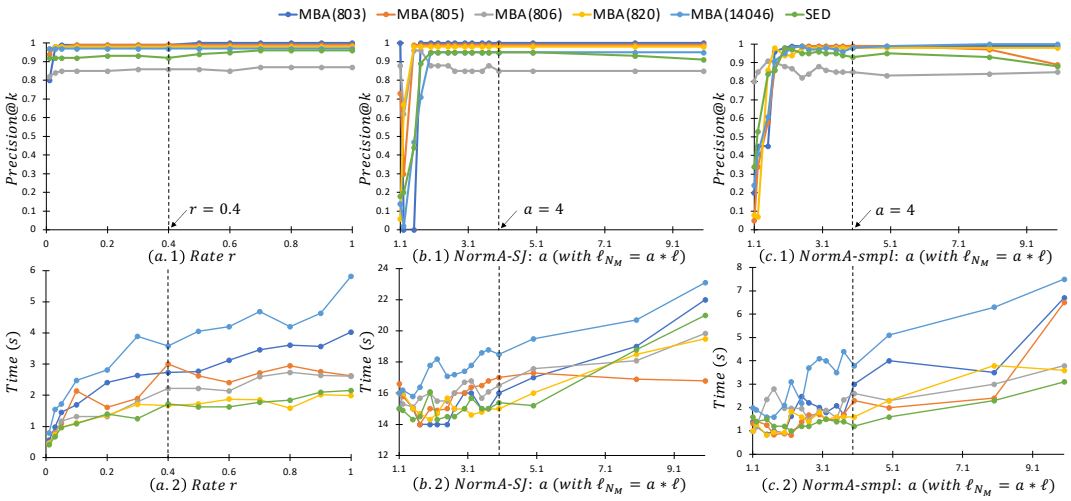


Figure 3.10: Precision@k and execution time (in seconds) over MBA and NASA datasets (a) when we vary r for NormA-smpl, when we vary l_{N_M} (b) for NormA-SJ and (c) for NormA-smpl.

the same. We then observe in Figures 3.10(b.2) and (c.2) that the execution time increases slightly when ℓ_{N_M} increases. This is expected because the overall complexity of NormA (both NormA-smpl and NormA-SJ) depends on ℓ_{N_M} (Section 3.3.3). We set the Normal Model length to the default value of 4x the anomaly length in all following experiments in Section 3.5.

3.5.5.3 Sampling rate r

We then compute accuracy as we vary the sampling ratio r (see Definition 17) for computing the Normal Model for NormA-smpl. Figures 3.10(a.1) and (a.2) depict Precision@k and execution time (in seconds) for MBA and NASA datasets when we vary the sampling ratio r between 0.01 (1 percent of non-overlapping subsequences selected) and 1 (100 percent of non-overlapping subsequences selected). Even though a small accuracy drop can be spotted for r smaller than 0.05, we observe that this parameter does not have a strong influence on accuracy. As r has a direct impact on $|\mathbb{S}|$ (which is, as explained in Section 3.3.1, an important parameter in NormA complexity), the execution time increases when r increases. In all following experiments in Section 3.5, we use the default value $r = 0.4$.

3.5.6 Series2Graph Parameters Influences

In this section, we evaluate the sensitivity of the Series2Graph parameters. As we build the graph on the entire data series (including the anomalies), we first evaluate the influence of the cardinality of the anomalies. We then evaluate the influence of the local convolution window λ, the number of radius subset r, the graph subsequence length ℓ_G, and the bandwidth h. For that purpose, we vary one of the parameter and set all the others to their

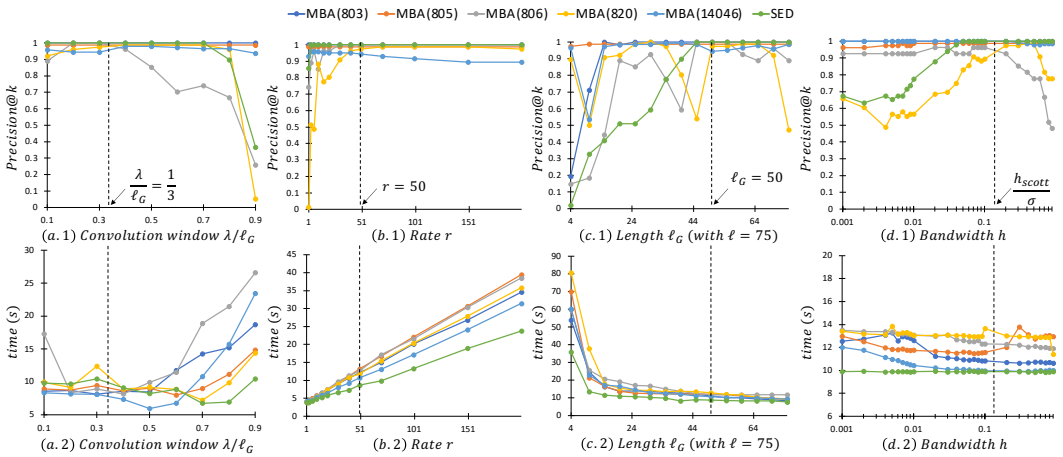


Figure 3.11: Precision@k and execution time (in seconds) over MBA and NASA datasets (a) when we vary λ/ℓ_G ratio, (b) when we vary the number of radius subset r (c) when we vary the graph subsequence length ℓ_G (while keeping the same targeted subsequence length ℓ to score) (d) when we vary the bandwidth ratio $h/\sigma((I)_\psi)$.

default values (i.e. $h = \bar{h}_{scott}$, $\lambda = \ell_G/3$, $\ell_G = 2/3 * \ell$ and $r = 50$). Figure 3.11 depicts the influences of these parameters on both accuracy (Precision@k) and execution time for all MBA and NASA datasets. We discuss in detail the depicted results in the following sections.

3.5.6.1 Anomaly Cardinality influence

In this section, we evaluate our method the detect rare and unusual subsequences in data series. Note that this represents the core hypothesis of our problem definition. We thus build synthetic data series composed of two or three distinct patterns. We randomly place them in the data series, and we vary their cardinality. We then compute the averaged anomaly score for each patterns type. We expect our method to detect as anomalies the patterns with small cardinalities. Figure 3.12 depicts the effect of the cardinality mentioned above on Series2Graph anomaly score. Figure 3.12(a) corresponds to time series that have one normal pattern that repeats over time (Pattern 1) and one anomaly that we inject (Pattern 2). We first measure the average anomaly score of these two patterns when we vary the cardinality of Pattern 2 (bottom plots of Figure 3.12(a)). We then change the cardinality of Pattern 2 that we inject (x-axis) by keeping the total sequence length constant. For each cardinality of Pattern 2, we repeat the process ten times: we compute the anomaly scores for ten different data series, composed by concatenating Patterns 1 and 2 at a random order (the cardinality of Patterns 1 and 2 remain constant). Figures 3.12(a.1) and 3.12(a.2) show that, on average, the anomaly score of Pattern 2 (red line) reduces as its cardinality in the data series increases. Similarly, the anomaly score of Pattern 1 increases as its cardinality in the data series decreases. We observe the same when we have more than two patterns (in Figure A(b), we added a third pattern with fixed cardinality of 5, which has its anomaly score represented by the black line in the bottom graph). In both Figures, 3.12(a) and 3.12(b), the crossing point between the red and blue line is approximatively when Pattern 2 becomes

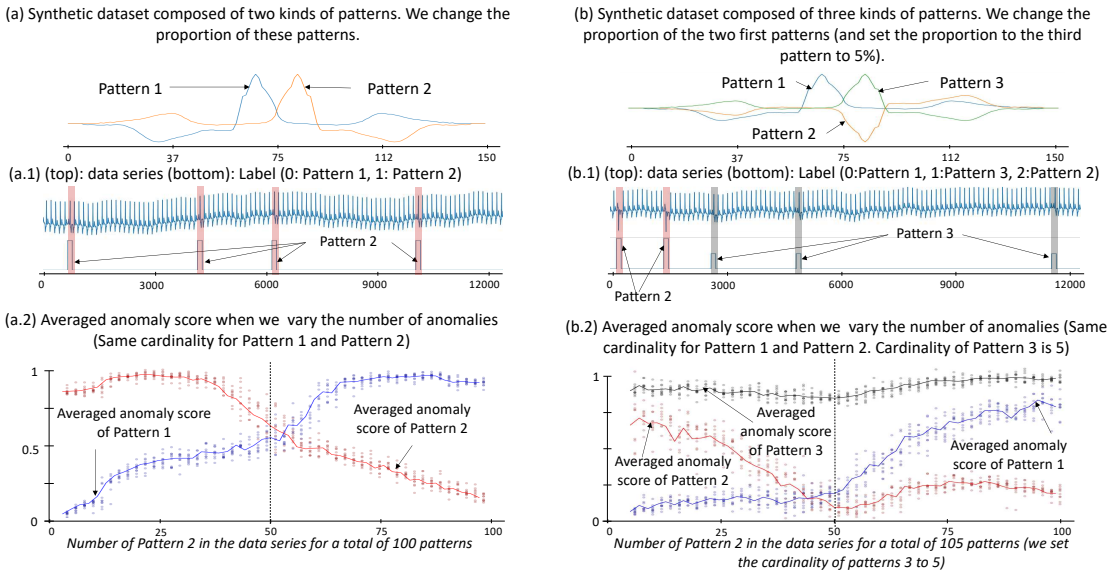


Figure 3.12: Effect of anomaly cardinality on Series2Graph detection accuracy

more frequent than Pattern 1. Assuming that, for long data series, anomalies are rare events, this empirically demonstrates the ability of Series2Graph to detect abnormalities.

3.5.6.2 Convolution window λ

We first analyze the influence of the convolution window λ on the anomaly detection accuracy and the execution time. Figures 3.11(a.1) and (a.2) depict Precision@k and execution time in seconds for all MBA and NASA datasets when we vary the ratio λ/ℓ_G. We observe a drop in accuracy for three of the six datasets when the ratio is greater than 0.5. It can be explained by the fact that the convolutions hide some critical part of the anomalous subsequences. We also observe an increase in the execution time when λ increases. In theory, as the complexity of the embedding step is $O(3 * |T|(\ell_G - \lambda))$, the execution time should reduce when λ increases. Nevertheless, the implementation of the first step (of complexity $O(|T| * \lambda)$ which is in theory negligible) is less efficient (because coded fully in Python) than the second step (of higher complexity $O(3 * |T|(\ell_G - \lambda))$ but coded in C/C++). Thus the increase in execution time is caused by the first step. We set by default $\lambda = \ell_G/3$.

3.5.6.3 Number of radius subsets r

We then analyze the influence of the number of radius subsets r on the anomaly detection accuracy and the execution time. Figures 3.11(b.1) and (b.2) depict Precision@k and execution time in seconds for all MBA and NASA datasets when we vary r. For three datasets, we notice a drop in accuracy for very small r. Nevertheless, the accuracy is stable for $r > 10$. We also observe that the execution time increases with r. It is explained by the parameter $\alpha_{\mathcal{N}}$ which varies between 1 and r. We thus set by default $r = 50$. However, this parameter

can be fine-tuned to better suit a specific use case.

3.5.6.4 Graph Subsequence Length ℓ_G

We now evaluate the influence of the graph subsequence length ℓ_G. For that purpose, we set as constant the anomaly subsequence length (ℓ), and we vary the graph subsequence length. Figures 3.11(c.1) and (c.2) depict Precision@k and execution time in seconds for all MBA and NASA datasets. We observe that the accuracy increases when ℓ_G gets closer to ℓ. It is expected as too small subsequences to build the graph would lead to miss unusual transitions. Moreover, the execution time is significantly higher for small ℓ_G. As the anomaly score computation is of complexity $O(|T| * (\ell - \ell_G))$, a small ℓ_G compared to ℓ increases the computational time. For our benchmark datasets, we set by default $\ell_G = 50$, which corresponds to approximately two-thirds of the anomaly length.

3.5.6.5 Gaussian Kernel Bandwidth h

We finally evaluate the impact of the kernel bandwidth h_{scott} in $f_h(x, \mathcal{I})$ in the node extraction step. We measure the accuracy for different bandwidths. Figures 3.11(d.1)and (d.2) display the Precision@k on all the MBA and NASA datasets as a function of $h/\sigma(\mathcal{I}_\psi)$ (logarithmic scale). As expected, a small bandwidth ratio breaks down too much the normal pattern, and therefore reduces its Normality score, while a large bandwidth ratio (above 0.7) hinders some key nodes from detecting anomalies in two of the six datasets, namely MBA(806) and MBA(820). The anomalies in these two datasets are close to the normal behavior. Thus the abnormal trajectories can be easily missed. In contrast, using the Scott bandwidth ratio h_{scott} (marked with the dotted line) leads to very good accuracy for all the datasets we tested (we used the datasets with the same anomaly and pattern lengths so that we can compare Scott bandwidth ratios). Moreover, we observe that the ratio $h/\sigma(\mathcal{I}_\psi)$ does not have any impact on the execution time.

3.5.6.6 Anomaly Length flexibility

We now evaluate the influence of the anomaly subsequence length. For NormA, the subsequences of length ℓ that the user want to evaluate can only be between $0 < \ell < \ell_{N_M}$. On the contrary, for Series2Graph, one user can evaluate subsequences of length ℓ between $\ell_G < \ell$. Thus, in theory, Series2Graph is more flexible than NormA. We already demonstrated that Series2Graph is not strongly affected by the parameter ℓ_G, we now illustrate and give an example on the flexibility of Series2Graph on the subsequence anomaly length.

Figure 3.13 depicts the $G_\ell(\mathcal{N}, \mathcal{E})$ graphs for ℓ_G equals to $80, 100, 120$, while the anomalies length is 75 for Type S anomalies, and 120 for type V anomalies. The results show that in all cases, irrespective of the length ℓ used to construct the graph, the anomaly trajectories (Type V highlighted in red and Type S highlighted in blue) are distinct from the highly-weighted trajectories (thick black) that correspond to normal behavior.

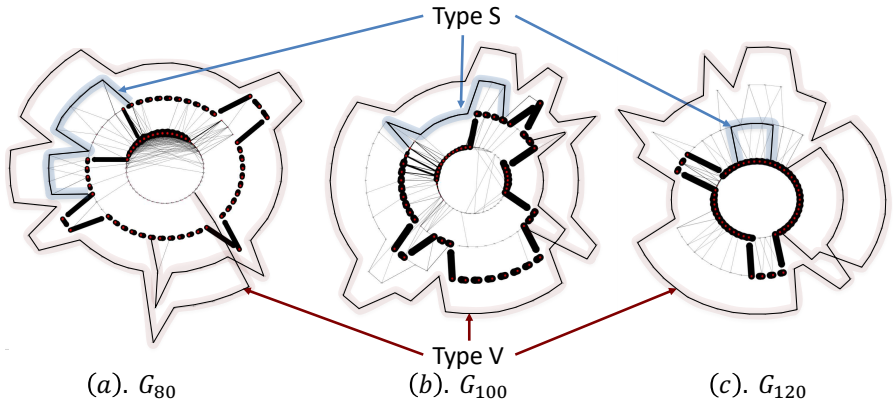


Figure 3.13: $G_\ell(\mathcal{N}, \mathcal{E})$ of the MBA(820) electrocardiogram data series for ℓ of $80, 100$ and 120. In the three cases, the different kinds of anomalies (S: blue, V: red) are well separable with lower edges weights.

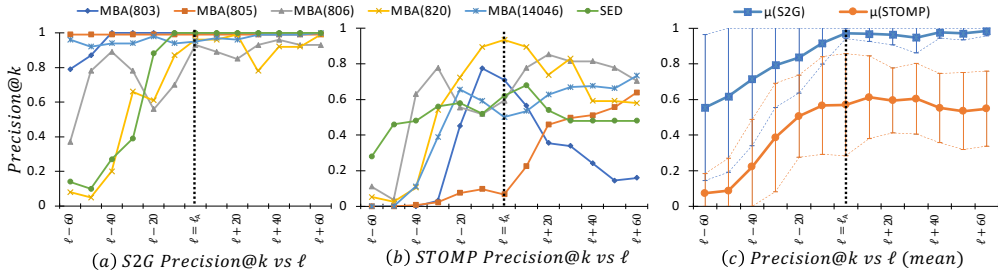


Figure 3.14: On the MBA and SED datasets: (a) Precision@k of Series2Graph varying the input and query length ($2\ell/3 = \ell_G$) to build the graph G_{ℓ_G}. (b) STOMP Precision@k varying the input length ℓ_G, (c) STOMP and Series2Graph in Precision@k average compared to the input length ℓ_G.

In order to complement this observation, we conduct the following experiment. First, we measure the Precision@k as the input length ℓ_G and the query length ℓ vary (using a query length $2\ell/3 = \ell_G$, with the anomaly length $\ell_A = 80$). Figure 3.14(a) demonstrates the stable behavior of Series2Graph. Even though the Top-k accuracy varies for small lengths, the performance remains relatively stable when the input lengths ℓ_G we use to construct the graph are larger than the anomaly length ℓ_A. This means that simply selecting an ℓ_G value larger than the expected anomaly length ℓ_A will lead to good performance.

In contrast, as Figure 3.14(b) demonstrates, the performance of STOMP (a discord-based approach) varies widely. Thus, such approaches need careful tuning, requiring domain expertise and good knowledge of the possible anomaly lengths. Furthermore, even though STOMP accuracy seems to converge to a stable value as the length is increasing, the Series2Graph accuracy stays significantly higher and much more stable in average, as shown in Figure 3.14(c).

Datasets	GV	STOMP	DAD	LSTM-AD	LOF	IF	NormA-smpl	NormA-SJ	Series2Graph
SED	0.46	0.57	0.44	0.10	0.65	0.65 (0.02)	0.92 (0.05)	0.91	**1.00**
MBA (803)	0.15	0.72	0.01	0.35	0.08	**1.00 (0.00)**	0.99 (0.01)	**1.00**	**1.00**
MBA (805)	0.09	0.10	0.03	0.85	0.42	0.99 (0.01)	**0.99 (0.00)**	**0.99**	**0.99**
MBA (806)	0.01	0.59	0.66	0.10	0.92	0.75 (0.06)	0.86 (0.02)	0.85	**1.00**
MBA (820)	0.05	0.90	0.04	0.09	0.42	0.92 (0.03)	**0.98 (0.01)**	**0.98**	0.91
MBA (14046)	0.09	0.54	0.71	**1.00**	0.64	0.99 (0.01)	0.95 (0.04)	0.93	0.95
SRW-[20]-[0%]-[200]	**1.00**	0.77	0.55	0.94	0.74	0.75 (0.05)	**1.00 (0.00)**	**1.00**	**1.00**
SRW-[40]-[0%]-[200]	0.97	**1.0**	0.05	**1.00**	0.89	0.92 (0.02)	0.97 (0.01)	0.97	**1.00**
SRW-[60]-[0%]-[200]	0.96	0.88	0.10	0.92	0.76	0.87 (0.02)	0.99 (0.01)	**1.00**	**1.00**
SRW-[80]-[0%]-[200]	0.96	0.43	0.14	0.95	0.82	0.86 (0.01)	**0.98 (0.00)**	0.98	**1.00**
SRW-[100]-[0%]-[200]	0.95	0.99	0.11	**1.00**	0.75	0.92 (0.02)	**1.00 (0.00)**	**1.00**	**1.00**
SRW-[60]-[5%]-[200]	**1.0**	0.73	0.21	0.96	0.88	0.89 (0.01)	**1.00 (0.00)**	**1.00**	**1.00**
SRW-[60]-[10%]-[200]	0.83	**0.98**	0.01	0.94	0.70	0.80 (0.01)	**0.98 (0.00)**	**0.98**	**0.98**
SRW-[60]-[15%]-[200]	0.76	0.62	0.17	0.94	0.66	0.82 (0.01)	0.99 (0.01)	**1.00**	0.98
SRW-[60]-[20%]-[200]	0.73	**1.0**	0.01	0.96	0.73	0.85 (0.02)	**1.00 (0.00)**	**1.00**	**1.00**
SRW-[60]-[25%]-[200]	0.63	0.64	0.09	0.83	0.67	0.80 (0.01)	**0.99 (0.01)**	0.94	**0.98**
SRW-[60]-[0%]-[100]	0.98	**1.0**	0.23	**1.00**	0.74	0.88 (0.02)	**1.00 (0.00)**	**1.00**	0.96
SRW-[60]-[0%]-[200]	0.96	0.60	0.19	**1.00**	0.85	0.83 (0.01)	**1.00 (0.00)**	**1.00**	0.98
SRW-[60]-[0%]-[400]	0.98	**1.0**	0.63	0.88	0.76	0.88 (0.01)	0.98 (0.01)	**1.00**	0.96
SRW-[60]-[0%]-[800]	0.91	0.86	-	0.76	0.69	0.87 (0.01)	0.97 (0.02)	**0.98**	**0.98**
SRW-[60]-[0%]-[1600]	**1.0**	**1.0**	-	0.90	0.52	0.64 (0.02)	0.92 (0.04)	0.97	0.94
average	0.62	0.73	0.24	0.78	0.68	0.85	0.97	**0.98**	**0.98**

Table 3.4: P@k accuracy for LOF, IF, DAD, STOMP, GV, LSTM-AD, NormA-smpl (standard deviation over 100 runs shown in parenthesis), and NormA-SJ and Series2Graph. We set k equal to the number of anomalies and ℓ to the length of the (annotated) anomalies.

3.5.7 Accuracy Evaluation

In this section, we report the anomaly detection accuracy results. In Table 3.4, we show the P@k accuracy (correctly identified anomalies among the k retrieved divided by k), with k equal to the number of anomalies. These experiments test the capability of each method to retrieve the k anomalous subsequences in each dataset correctly. For NormA, we simply have to report the P@k anomalies that the algorithm produces. Similarly, we compute accuracy for Isolation Forest and LOF, considering the k subsequences assigned with the highest scores by these two approaches. For the discord based techniques, we have to consider the $Top\text{-}k$ 1^{st} discord and the m^{th} discord (with $m = k$). Finally, LSTM-AD marks as anomalies the subsequences that have the largest errors (distances) to the sequences that the LSTM-AD algorithm predicts; we compute accuracy considering the subsequences with the k largest errors.

In the first section of Table 3.4, we report the results of all techniques on the annotated real datasets with multiple (diverse and similar) anomalies. We observe that both NormA (NormA-SJ and NormA-smpl) and Series2Graph are outperforming the other state-of-the-art approaches, except for MBA(14046), for which their performance is still very close to the best performer. As expected, $Top\text{-}k$ 1^{st} discord techniques (GV and STOMP) achieve low accuracy since anomalies do not correspond to rare subsequences (i.e., isolated discords). We also observe that the m^{th} discord technique (DAD), which can detect groups of m similar anomalous subsequences, does not perform well, either. It is due to the many false positives produced by the algorithm.

In the other three sections of Table 3.4, we report the accuracy of the evaluated methods

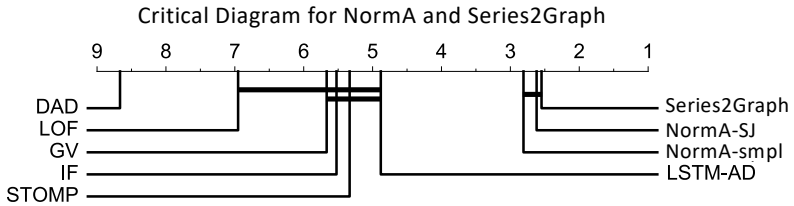


Figure 3.15: Critical difference diagram ($\alpha = 0.05$) for the data series of Table 3.4.

on all the synthetic datasets (where we vary the number of anomalies, the % of Gaussian noise, and the anomaly subsequence length ℓ). We note that the accuracy of the discord discovery techniques substantially improves since, in this case, most anomalies correspond to rare and isolated subsequences (i.e., different from one another). Even in these cases, NormA and Series2Graph are more accurate than other state-of-the-art approaches.

Regarding LSTM-AD, we note that, in general, it is more accurate than the discord-based algorithms. Nevertheless, we stress that LSTM-AD only achieves this performance because (contrary to the rest of the techniques) it benefits from a training phase on labeled data. However, in order to train the LSTM model, we select the longest continuous sequence without anomalies. As some of the datasets contain many anomalies, the longest sequence without anomalies may not be long enough to fit the model correctly. As a matter of fact, LSTM-AD cannot match the performance of NormA and Series2Graph. Since we would expect a semi-supervised algorithm to perform at least as good as an unsupervised algorithm, these results suggest that supervised methods still have lots of potential for improvement.

Regarding LOF, we observe that it does not perform well in our context. Isolation Forest achieves better performance but not as good as NormA and Series2Graph.

Overall, we observe that NormA and Series2Graph are more accurate than all competitors (with very few exceptions, for which their performance is still very close to the best one) in all the settings we used in our evaluation. Moreover, we observe that Series2Graph and NormA-SJ have similar performances. Furthermore, we note that the performance of NormA-smpl is in almost all cases equal to that of NormA-SJ and Series2Graph, or very close to it. However, on average, it is slightly less accurate on our dataset corpus.

Finally, after rejecting the null hypothesis using the Friedman test, we use the pairwise Post-Hoc Analysis using a Wilcoxon signed-rank test [116] to test and produce the critical difference diagram for the algorithms and datasets of Table 3.4. The critical difference diagram with $\alpha = 0.05$ (Figure 3.15) shows that Series2Graph, NormA-SJ, and NormA-smpl are the overall winners, with Series2Graph, NormA-SJ, and NormA-smpl being significantly better than all previous algorithms.

3.5.8 Execution Time Evaluation

We now present scalability tests (we do not consider LSTM-AD since supervised methods have a completely different way of operation and associated costs, e.g., data labeling and model training)

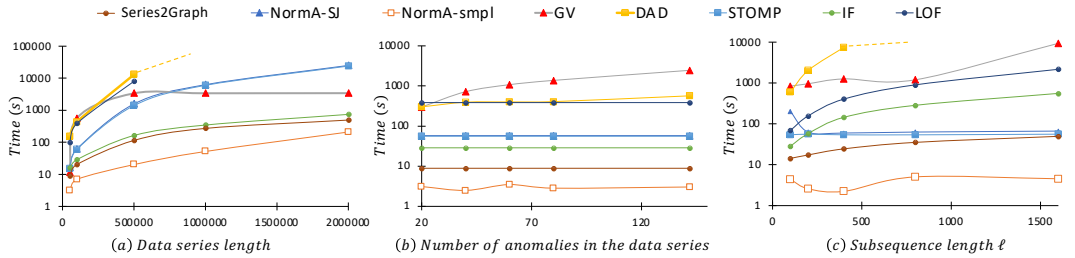


Figure 3.16: Execution time (in seconds) evaluation for the baselines and our two proposed approaches NormA and Series2Graph when we vary (a) the data series length, (b) the number of anomalies, (c) the subsequence length ℓ.

In Figure 3.16(a), we report the execution time (seconds in log scale) of NormA, Series2graph, and all the state-of-the-art approaches, when varying the data series length ($|T|$). We use several prefix snippets (50K, 100K, 500K, 1M, 2M points) of the real dataset MBA(14406), and we set k equal to the number of anomalies that are annotated in each snippet. We observe that NormA-smpl is 1-2 orders of magnitude faster than all the other approaches and gracefully scales with the dataset size. It is because the number of distance calculations performed by NormA-smpl in Algorithm 9 for each subsequence in the data (computation of join sequence) is limited to the subsequences contained in N_M. NormA performs a limited number of distance calculations during subsequence clustering (Algorithm 1), since only a small part of subsequences in the input data series are selected to be clustered ($\mathbb{S}^{selfjoin}$, or $\mathbb{S}^{sample}$). Thus, NormA-SJ that uses the STOMP algorithm for the Normal Model computation stage has a small additional time overhead (compared to STOMP). GV, DAD, and LOF adopt different pruning strategies to reduce the number of Euclidean distance computations, proving to be less effective. DAD and LOF, in particular, reach the time-out point (8 hours in our experiments) for datasets larger than one million points. Moreover, we also notice that Series2Graph is the second-fastest approach. As NormA-smpl is performing random sampling as preprocessing step, Series2Graph is slower than NormA-smpl. However, Series2Graph still scales with the dataset size.

In the next set of experiments, we measure the execution time (seconds in log scale) of the algorithms as we vary the number of anomalies; we use the MBA(14406) and instruct the algorithms to find 20, 40, 60, 80, and 142 anomalies (Figures 3.16(b)). We observe that the time performance of NormA and Series2Graph is not influenced by the number of anomalies, since for every subsequence in the dataset, we compute the distance anyway to its nearest neighbor in the Normal Model (for NormA) and the corresponding path in the graph (for Series2Graph). Similarly, STOMP, IF and LOF enumerate in quadratic time all the Top-k 1^{st} *discords*, always consuming the same amount of time. In contrast, the performance of GV and DAD are negatively influenced by the number of anomalies. It confirms that the pruning strategies they use are influenced by the number of anomalies to discover.

Figure 3.16(d) depicts the time performance results as we vary the length of the anomalies between 100-1600 points (SRW-[60]-[0%]-[100-1600] data series). The performance of STOMP is constant because its complexity is not affected by the (anomaly) subsequence length. NormA remains relatively stable since in Algorithms 2 the Euclidean distances are

computed using the STOMP algorithm. In NormA, only the clustering operations are affected by the length of the subsequences to consider (Algorithm 1), which in all experiments we ran was always a very small number ($\sim$1-2% of all subsequences). We observe that the execution time for NormA-SJ decreases as we move from anomaly length 100 to length 200. This decrease is explained by the reduction of the number of non-overlapping subsequences to cluster, which drops from 242 (anomaly length 100) to 128 (anomaly length 200). Regarding NormA-smpl, we see a slight fluctuation in execution time, between 1.1-2.4 sec. For Series2Graph, we observe that the execution time of Series2Graph increases slightly for larger subsequence lengths. It is due to the scoring function (the last step of the algorithm). This function sums up all the edge weights of the subsequences we are interested in. Therefore, if the subsequence is large, the number of relevant edges is large as well, which slightly affects the computation time. LOF and IF are computing distances using all overlapping subsequences, and the computational time is therefore affected by their length. As shown in Figure 3.16(d), both of these two methods perform orders of magnitude worse than STOMP and NormA. GV and DAD do not scale with the anomaly length, either.

3.5.9 User Interfaces: Stand Alone Web Applications

We also proposed two user interfaces enabling the user to interact, visualize the detected anomalies and the different inner computational steps of NormA and Series2Graph illustrated by their mainframe screenshots in Figure 3.18 and Figure 3.17. Both of them are stand alone web applications developed using Python 3.6 and the Dash framework [29].

3.5.9.1 GraphAn: Series2Graph User Interface

The mainframe of GraphAn [14] is shown in Figure 3.17. Once the user opens the web application, they can upload a dataset (as well as the anomaly annotations, if available) that will appear as in Figure 3.17(a.1). The user can then change the values of ℓ_G and ℓ by clicking on the Series2Graph dropdown in the navigation bar in the middle, and subsequently, visualize and rotate/zoom in the embedding space (Figure 3.17(a.2)) and the resulted graph $G_\ell(\mathcal{N}, \mathcal{E})$ (Figure 3.17(a.3)). By clicking on the points in the embedding space, the user can visualize the corresponding subsequences (Figure 3.17(a.2.1)). Similarly, the user can click on a node in the graph in order to see which subsequences belong to it (Figure 3.17(a.3.1)). Once these steps are performed, the user can perform the Series2Graph anomaly score computation, which will be displayed under the uploaded data series (Figure 3.17(a.4)). The user can also run other anomaly detection methods: STOMP (Section 2.5.4.4), Isolation Forest (IF, defined in Section 2.5.3.2) and Local Outlier Factor (LOF, defined in Section 2.5.3.1). Their anomaly scores will be shown together with the Series2Graph anomaly scores (Figure 3.17(a.4)).

3.5.9.2 SAD: NormA User Interface

The SAD GUI [15] allows users to directly interact with NormA framework and interface across the entire range of steps of the algorithm that are executed under the hood. It first al-

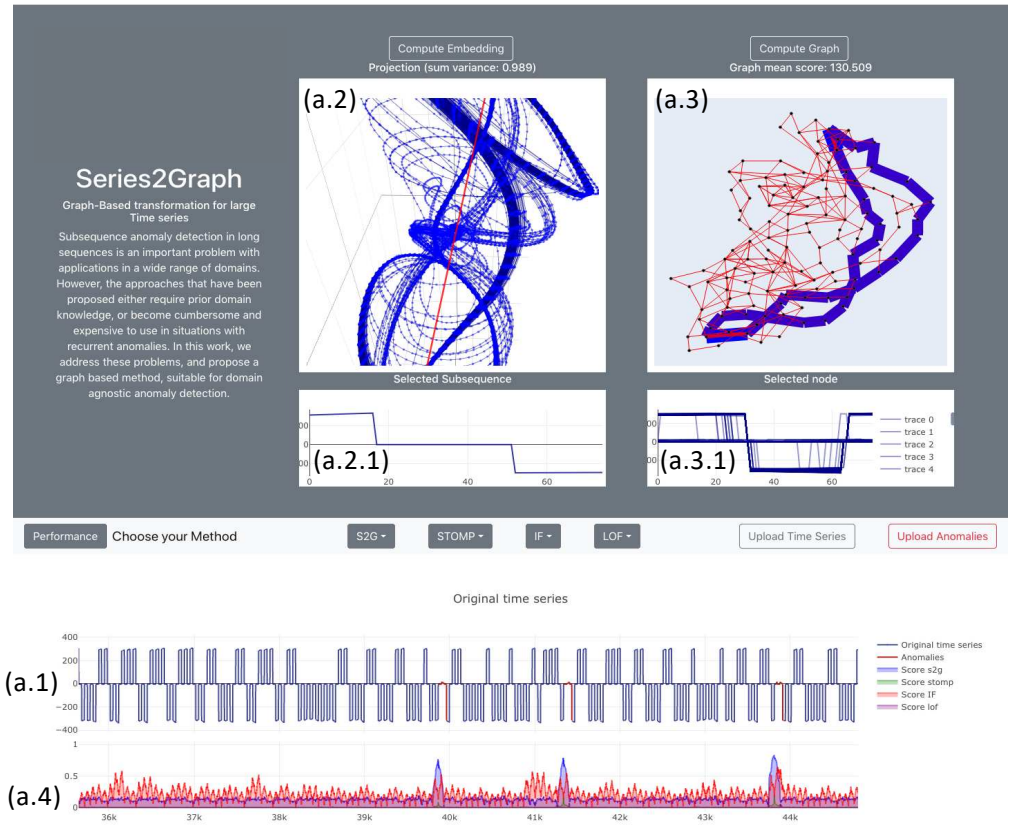


Figure 3.17: Screenshots of user interfaces GraphAn, dedicated to Series2Graph.

lows users to import their own datasets via an upload tab (top right button in Figure 3.18). The data series is then plotted in Figure 3.18(b.1) (synchronized with anomaly score of NormA in Figure 3.18(b.2)) If annotations (i.e., anomaly labels) are available, they can also be uploaded to SAD, which will use them to highlight the anomalous subsequences (red) in the time series plot. Accuracy and execution time measures are also provided (Figure 3.18(b.3)). The second functionality enables users to intervene and modify the operation of the NormA framework. SAD visualizes each step of the process (Subsequences selection, Clustering, Normal Model selection) and allows users to change the internal parameters of these steps of the algorithm. SAD displays the values for the internal parameters that were automatically selected by NormA, along with N_M and the subsequences that were used to compute it. Thus, SAD enables users to understand better how NormA works. Finally, the user can also run other anomaly detection methods such as STOMP (Section 2.5.4.4), Isolation Forest (IF, defined in Section 2.5.3.2) and Local Outlier Factor (LOF, defined in Section 2.5.3.1). Their anomaly scores will be shown together with the NormA anomaly scores (Figure 3.18(b.2)).

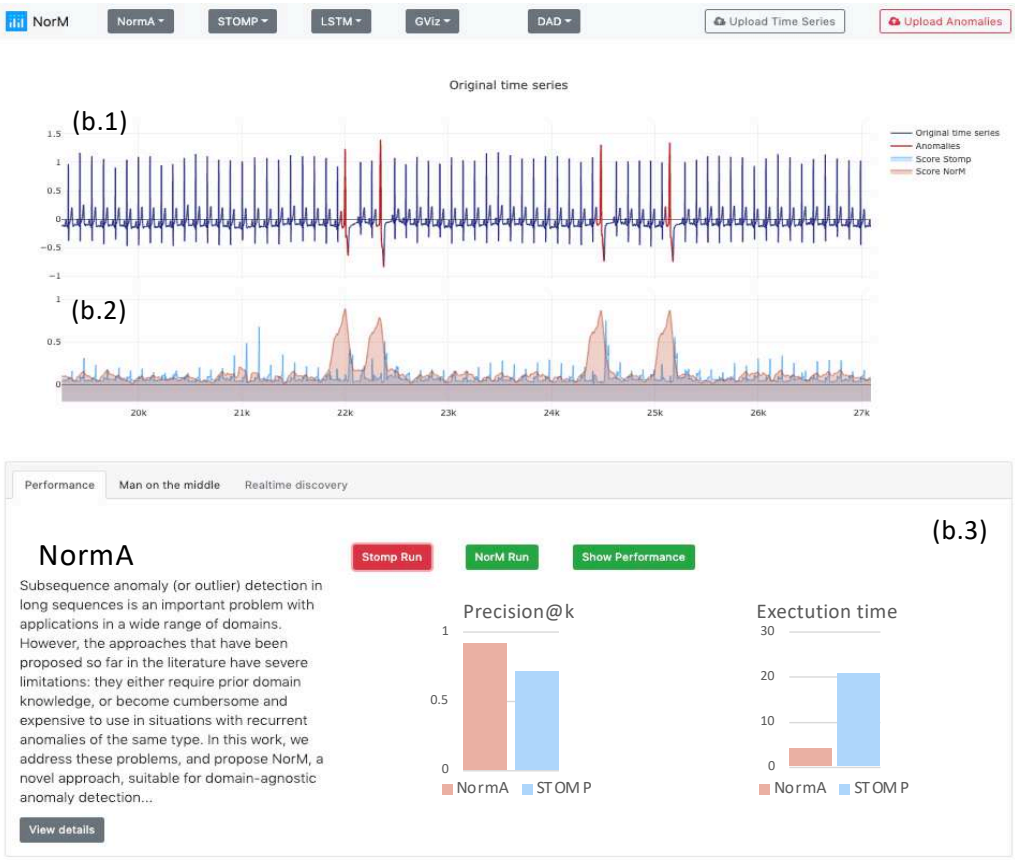


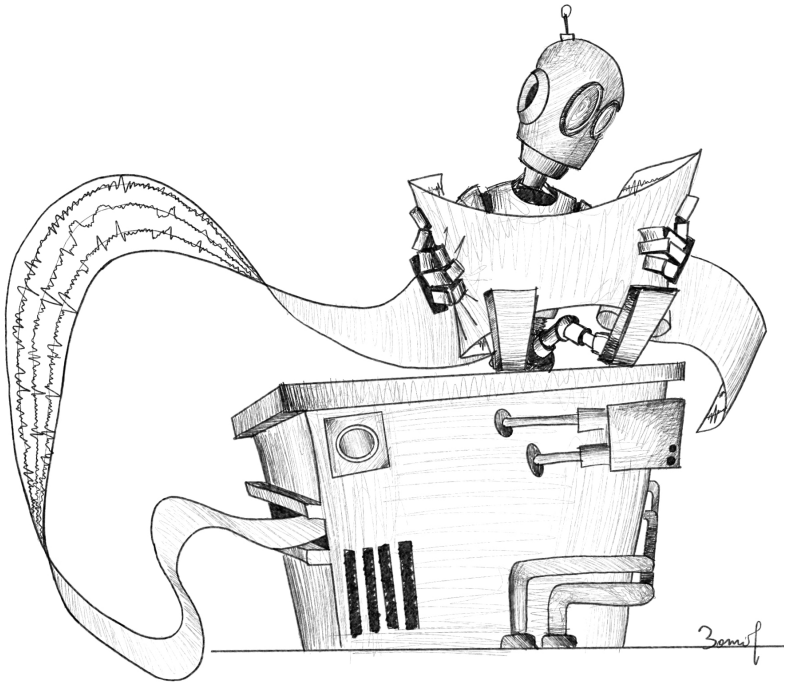
Figure 3.18: Screenshots of user interfaces SAD, dedicated to NormA.

3.6 Summary

In this chapter, we presented two new approaches, NormA and Series2Graph, that aim to solve the task of unsupervised subsequence anomaly detection in data series. We described in detail the computational steps of the two proposed algorithms. We then evaluated the parameter influences of the different approaches. We finally compare the anomaly detection accuracy and the execution time of the two proposed techniques compared to the current state-of-the-art methods. Both theoretically and empirically, we underlined the limitation of the discord-based approaches and density-based approaches when it comes to large data series that can contain several similar anomalies. We empirically demonstrated the superiority of our two proposed approaches both in anomaly detection accuracy and execution time.

Chapter 4

Streaming Subsequence Anomaly Detection



Chapter Outline and Abstract

4.1 Limitations of Previous approaches for the Streaming case 99
4.2 Proposed approach: SAND 99
4.2.1 Overview 100
4.2.2 Model initialization 102
4.2.3 Continuous Model Update 103
4.2.4 Subsequence Scoring 107
4.2.5 Execution Time Complexity Analysis 108
4.2.6 Parameters Influences 109
4.3 Experimental Evaluation 112
4.3.1 Implementation 112
4.3.2 Description of the evaluation metrics 112
4.3.3 Description of the datasets 113
4.3.4 Description of the baselines 113
4.3.5 Accuracy Evaluation 114
4.3.6 Time Performance Evaluation 117
4.3.7 User Interface: StandAlone Web Application 119
4.4 Summary 120

In the previous chapter, we tackled the unsupervised subsequence anomaly detection method for static data series. However, with the increasing demand for real-time analytics and decision-making, anomaly detection methods need to operate over streams of values and handle drifts in data distribution. Moreover, subsequence anomaly detection methods usually require access to the entire dataset and are not able to learn and detect anomalies in streaming settings. This chapter tackles the above-mentioned issues and proposes SAND, a novel online method suitable for domain-agnostic anomaly detection. SAND aims to detect anomalies based on their distance to a model that represents normal behavior. SAND relies on a novel streaming methodology to incrementally update such a model, which adapts to distribution drifts and omits obsolete data. The experimental results on several real-world and simulated datasets demonstrate that SAND correctly identifies single and recurrent anomalies without prior knowledge of the characteristics of these anomalies.

4.1 Limitations of Previous approaches for the Streaming case

The previous chapter proposed two approaches that overcome state-of-the-art unsupervised subsequence anomaly detection methods limitations for static data series. For several use cases in which data measurements are arriving continuously in several real-world cases, it requires anomaly detection to take place in real-time. Because drifts in data distribution are common, the detection needs to be independent of these changes.

Nevertheless, not all previous approaches (both existing and our proposed approaches) can be adapted to the streaming case. In this case, among all previous methods for subsequence anomaly detection, only discords methods (such as Matrix Profile incremental implementation [124]) and tree-based methods [79] can be used. The remaining methods cannot adapt to changes and learn new data characteristics, both of which are required when dealing with data streams due to their design. In such cases, the methods need to learn and modify their parameters as new data arrive.

4.2 Proposed approach: SAND

To address the problems mentioned above, we propose SAND [17], a novel approach suitable for subsequence anomaly detection in data streams. For this novel approach, we start from the same data structure proposed in Section 3.3.1 and modify the computational steps to enable streaming usage. SAND builds a data set of subsequences representing the different behaviors of the data series. These subsequences are weighted using statistical characteristics such as their cardinality (i.e., how many times the subsequence occurred) and their temporality (i.e., the time difference this subsequence has been detected for the last time). SAND enables this data structure to be updated from one batch to another while computing an anomaly score at every timestamp. Thus, SAND proposes a solution to the subsequences anomaly detection task on streaming data. SAND benefits from k-Shape (described in Section 2.5.2), a state-of-the-art data series clustering method, which we extend to enable the clustering result to be updated without storing any of the previous subsequences. We demonstrate experimentally that our method outperforms the current (static and streaming) state-of-the-art approaches. Our contributions are as follows.

- We describe the concepts and ideas used by the state-of-the-art methods on subsequence anomaly detection (static and streaming) and discuss their practical shortcomings.
- We extend k-Shape for streaming scenarios by enabling batch updates of the clustering partition. Our approach avoids entirely the storage of the previous subsequences, a critical step for operating over unbounded data series.
- We present SAND, our subsequence anomaly detection method specifically designed for operation over streaming sequence data. SAND exploits our streaming k-Shape

Symbol	Description
T	a data series
$\|T\|$	cardinality of T
ℓ	input subsequence length
$T_{i,\ell}$	subsequence of T of length ℓ, starting at index i
$\mathbb{T}_\ell^0$	initial batch of T of length b_{size}
$\mathbb{T}_\ell^t$	batch starting at timestamp t of T of length b_{size}
$\mathbb{C}$	partitioning of a clustering algorithm
$\mathcal{C}_i$	cluster in $\mathbb{C}$
$\bar{\mathcal{C}}_i$	centroid of $\mathcal{C}_i$
k	number of clusters

Table 4.1: Table of symbols.

to scale in memory and in execution time for unbounded streams. We propose a new weighting scheme for clusters and an automatic cluster creation procedure to handle distribution drifts. We finally propose a novel anomaly score computation that adapts dynamically to the current batch and gives less importance to old subsequences.

- We perform an experimental analysis using a large data corpus of real datasets (including a ground truth of annotated anomalies) from different fields. We evaluate both subtle changes in data characteristics (by concatenating datasets from the same domain) and drastic changes (by concatenating datasets from different fields). We empirically evaluate the influence of SAND's parameters on accuracy and execution time. Finally, we compare SAND with several state-of-the-art approaches.

To conclude, the problem (derived from Problem 1 and adapted to the streaming case) we solve in this chapter is defined as follows.

Problem 2 (Streaming Subseq. Anom. Detection)**.** *Given a data stream T, arriving in batches $\mathbb{T}_\ell^t$ (with b_{size} the size of the batches) and a targeted anomaly subsequence length ℓ, propose a function $f : \mathbb{T}_\ell^t \rightarrow \{\mathcal{N}, \mathcal{A}\}$ with $\mathcal{A}$, a set containing the η most abnormal subsequences of length ℓ.*

In this work, we focus on the $Top\text{-}k$ anomalies. Using a threshold ϵ instead to detect anomalies is a straightforward extension. Table 4.1 summarizes the symbols we use in this chapter.

4.2.1 Overview

In this section, we present SAND, our solution for unsupervised subsequence anomaly detection in data streams.

Overall, we compute and update a weighted set of subsequences over time. The summary of the computation steps is as follows:

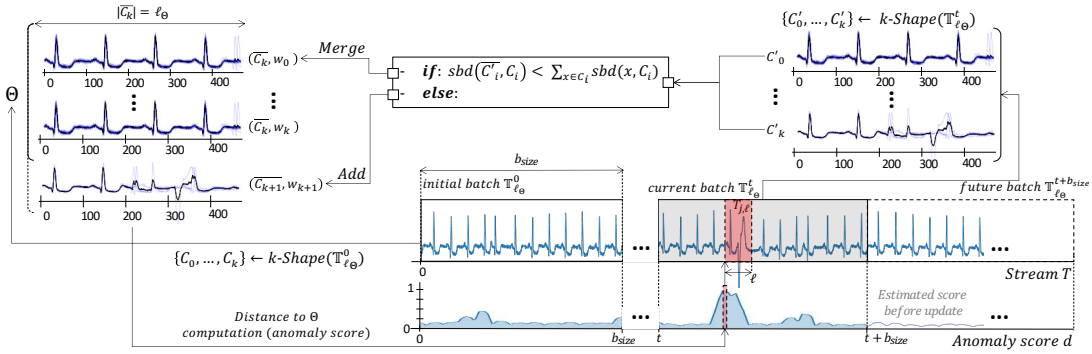


Figure 4.1: SAND computation framework.

1. **Model initialization:** We start by computing our initial model on the initial batch. We first select subsequence candidates and then perform the k-Shape clustering algorithm. These clusters are then scored and stored in memory (Section 4.2.2).

2. **Continuous Model update:** After each new batch, we compute a new clustering on the newest batch. We then match the new cluster with the similar one in the current model stored in memory. The matching procedure is based on a distance threshold corresponding to the intra-cluster average distance for each existing cluster. If the distance between an existing cluster and a new cluster is smaller than the threshold, we merge the two clusters. Otherwise, we create a new cluster. This procedure is described in Section 4.2.3.1.

3. **Centroid update:** We then propose a mechanism to compute the centroid of two merged clusters without keeping in memory the raw subsequences of these two clusters. This mechanism is a novel technical extension of k-Shape for streaming scenarios. The matching system and the centroid computation is summarized in Algorithm 8 and in Section 4.2.3.2.

4. **Weight update:** We finally update the weights for each cluster (new, merged, or unchanged) based on their previous score. The update procedure is summarized in Algorithm 9 and in Section 4.2.3.3.

5. **Subsequence scoring:** At any time, one can compute the anomaly score on the current batch using the current model stored in memory. We incrementally learn the mean and the standard deviation to compute the anomaly score such that the anomaly detection is adapted to the current batch subsequences/behaviors (Section 4.2.4).

Figure 4.1 depicts the general framework of the approach. We now describe in detail the different steps of our methodology. Algorithm 7 summarizes the parameters, as well as the update procedures of the set Θ. Next, we describe in detail the different steps.

Algorithm 7: SAND

input :
A data stream T: T should be composed of numerical values, evolving on a single dimension.
A subsequence length ℓ_Θ (with the condition $\ell_\Theta > \ell$): As empirically shown [18], for a given ℓ, $\ell_\Theta = 3 * \ell$ can be used as the default setting.
An initial number of clusters k: It will be used as the number of clusters for the k-Shape algorithm for the initial clustering, and at each new batch. In practice, there is no restriction to use a different k for the initialization and the other steps. For the sake of simplicity, we use the same k.
An initial batch $\mathbb{T}^0_{\ell_\Theta}$ **(and a batch size** $b_{size} < |T|$**)**: b_{size} is also used to set the size of every new batch $\mathbb{T}^t_{\ell_\Theta}$. In practice, one can use a different batch size for the initial batch and the others. For the sake of simplicity, we use the same batch size.
A real number $\alpha \in [0, 1]$: A parameter conditioning the rate of change of the centroids, the weights, the estimated mean, and standard deviation of the anomaly score.
output: A set Θ, a data series $score$

// Initialization
$\Theta, \mathbb{C}^0 \leftarrow \{\}, kShape(\mathbb{T}^0_{\ell_\Theta}, k)$;
$\Theta \leftarrow updateCentroid(\Theta, \mathbb{C}^0, init = True)$;
$\Theta \leftarrow updateParam(\Theta, \alpha)$;
$\mu, \sigma \leftarrow 0, 0$;
// Online Update
foreach *in coming batch* $\mathbb{T}^t_{\ell_\Theta}$ **do**
  $\mathbb{C}^t \leftarrow kShape(\mathbb{T}^t_{\ell_\Theta}, k)$;
  $\Theta \leftarrow updateCentroid(\Theta, \mathbb{C}^t, init = False)$;
  // see Alg. 8
  $\Theta \leftarrow updateParam(\Theta, \alpha)$;
  // see Alg. 9
  $score, \mu, \sigma \leftarrow computeScore(\Theta, \mathbb{T}^t_{\ell_\Theta}, \alpha, \mu, \sigma)$;
  // see Alg. 10
end

4.2.2 Model initialization

We start by describing the initialization step of the SAND. It step consists in building a set of clusters paired with weights, denoted as $\Theta = \{(\bar{\mathcal{C}}_0, w_0), (\bar{\mathcal{C}}_1, w_1)..., (\bar{\mathcal{C}}_k, w_k)\}$ similarly as described in Section 3.3.1 (note that, from a higher perspective, the previously discussed Normal Model N_M has the same data structure as Θ). In the latter, $\bar{\mathcal{C}}_k$ is the centroid of cluster $\mathcal{C}_k$ which is a sequence of length ℓ_Θ. Note that this Θ set will be our main data structure that will evolve through time. For simplicity purposes, in the remainder of this chapter, we say that $\bar{\mathcal{C}}_i$ is in Θ if there exists the tuple $(\bar{\mathcal{C}}_i, w_i)$ in Θ. We describe the initialization of this set below.

We first select the subsequences in an initial batch before clustering them. Formally, for a given data series T, and a batch of length b_{size} we define $\mathbb{T}^0_{\ell_\Theta} = \{T_{0,\ell_\Theta}, ..., T_{b_{size}-\ell_\Theta,\ell_\Theta}\}$ as the set of all overlapping subsequences in the first initial batch. For efficiency and accuracy matters, one can argue that only a subset of non-overlapping subsequences might be selected. However, using an appropriate clustering algorithm (such as k-Shape clustering algorithm), one can cluster highly overlapping subsequences. We then apply the k-Shape

clustering algorithm. For the sake of simplicity, we define k-Shape: $\mathbb{T}_{\ell_\Theta}, k \rightarrow \mathbb{C}$, with k being the number of clusters and $\mathbb{C} = \{\mathcal{C}_0, ..., \mathcal{C}_k\}$ being the set of clusters. The number of clusters k is a user defined parameter. We evaluate its influence in Section 4.2.6. Note that the number of clusters k will imply the initial size of the set Θ. A cluster $\mathcal{C}_i$ is defined as $\mathcal{C}_i \subset \mathbb{T}^0_{\ell_\Theta}$ and $\forall \mathcal{C}_i, \mathcal{C}_j \in \mathcal{C}, \mathcal{C}_i \cap \mathcal{C}_j = \emptyset$. The initialization step is defined as follows:

$$\begin{aligned} &\{\mathcal{C}_0, ..., \mathcal{C}_k\} = kShape(\mathbb{T}^0_{\ell_\Theta}, k) \\ &\forall (\bar{\mathcal{C}}_i, w_i) \in \Theta, \begin{cases} w'_i &= \frac{|\mathcal{C}_i|^2}{\sum_{\bar{\mathcal{C}}_j \in \Theta} sbd(\bar{\mathcal{C}}_i, \bar{\mathcal{C}}_j)}. \\ w_i &= \frac{w'_i}{\sum_{w'_j} w'_j} \end{cases} \end{aligned} \tag{4.1}$$

As described in the previous equation, we normalize the weights w_i to have their sum equal to 1. In Θ computation, k-Shape algorithm internally handles the realignment of the sequences and thus permits to use a high number of subsequences of T without realigning them beforehand. To be consistent with the SBD distance used in the k-Shape algorithm, we use the SBD distance in the scoring step to measure the isolation of a given cluster from the rest of the clusters. Theoretically, to be able to update sequences (i.e., centroids) in Θ, we have to store in memory the subsequences that were used to compute them. As mentioned earlier, we denote $\mathcal{C}_i$ the subsequences set, and $\bar{\mathcal{C}}_i$ its centroid. Nevertheless, storing $\mathcal{C}_i$ implies an infinite storage need for unlimited streams. Thus, in practice, we do not store $\mathcal{C}_i$, and we describe in Sections 4.2.3.2 how we update the sequences in Θ without storing their corresponding set $\mathcal{C}_i$. However, for the sake of simplicity, we still use $\mathcal{C}_i$ notation as virtual sets corresponding to $\bar{\mathcal{C}}_i$ in Θ.

4.2.3 Continuous Model Update

Once the initialization is done, the model is ready to receive new subsequences. Let $\mathbb{T}^t_{\ell_\Theta}$ be the set of subsequences (of size $|\mathbb{T}^t_{\ell_\Theta}| = b_{size}$) from the current batch arriving at time t. The length of this batch is a user parameter. We evaluate the influence of this parameter in Section 4.2.6. For every new batch, we perform a k-Shape clustering operation with k clusters (same value of k used in the initialization step), and we note the clustering result $\mathbb{C}^t$.

4.2.3.1 Matching Strategy

We then match $\mathbb{C}^t$ with the sequences in Θ. In practice, we define a threshold $\tau_{c,j}$ for each sequence $\bar{\mathcal{C}}_j$ in Θ. We then verify if the distance between a centroid of a new cluster in $\mathbb{C}^t$ and an existing sequence $\bar{\mathcal{C}}_j \in \Theta$ is smaller than $\tau_{c,j}$. Formally, given a cluster $\mathcal{C}^t_i$ in the clustering result $\mathbb{C}^t = \{\mathcal{C}^t_0, ..., \mathcal{C}^t_k\}$ on the current batch $\mathbb{T}^t_{\ell_\Theta}$, Θ, and a threshold $\tau_{c,j}$ for each sequence $\bar{\mathcal{C}}_j$, the matching process is operated as follows:

- if $\exists (\bar{\mathcal{C}}_j, w_j) \in \Theta, SBD\big(\bar{\mathcal{C}}_j, \bar{\mathcal{C}}^t_i\big) < \tau_{c,j}$: We consider that cluster $\mathcal{C}^t_i$ found in the current

batch is similar to an existing sequence $\bar{\mathcal{C}}_j$ in Θ. We thus update the sequence $\bar{\mathcal{C}}_j$ and its corresponding weight w_j using the new cluster $\mathcal{C}_i^t$.

- if $\forall(\bar{\mathcal{C}}_j, w_j) \in \Theta, SBD\big(\bar{\mathcal{C}}_j, \bar{\mathcal{C}}_i^t\big) > \tau_{c,j}$: We consider that cluster $\mathcal{C}_i^t$ is not similar enough with any of the existing sequences in Θ. Therefore, we include it as a new sequence in set Θ. We then compute its corresponding score and centroid, and insert it into Θ.

We assume that the cluster quality is a significant property to guarantee an accurate detection of anomalies, and the number of clusters directly impacts the execution time. As underlined in the previous paragraph, threshold $\tau_{c,j}$ has a direct impact on the number of new clusters that will be created at each batch and on the cluster quality. On the one hand, a high $\tau_{c,j}$ will make the cluster creation rare but might harm the cluster quality. On the other hand, a low $\tau_{c,j}$ will imply a large number of clusters created at each batch. Therefore, an optimal threshold that preserves the quality of the clusters without creating too many clusters is difficult to set for a user. Moreover, one threshold cannot be the same for every cluster. Therefore, to set automatically a threshold that adapts to clusters, we use the intra-cluster distance. For a given subsequence $\bar{\mathcal{C}}_j$ in Θ, we compute the intra-cluster distance as follows:

$$\tau_{c,j} = \sum_{T_{i,\ell} \in \mathcal{C}_j} SBD(T_{i,\ell}, \bar{\mathcal{C}}_j) \tag{4.2}$$

We then use $\tau_{c,j}$ to decide if a new cluster should be created or not. In practice, we do not store set $\mathcal{C}_j$. If one cluster changes, we need to update the threshold dynamically. We thus store the size of Θ clusters only. Formally, for a sequence $\bar{\mathcal{C}}_j$ in Θ and a given cluster $\mathcal{C}_i^t$ to be merged, we update threshold $\tau_{c,j}^*$ as follows:

$$\tau_{c,j}^* \leftarrow \frac{|\mathcal{C}_j| * \tau_{c,j}}{|\mathcal{C}_j| + |\mathcal{C}_i^t|} + \frac{|\mathcal{C}_i^t| * \sum_{T_{m,\ell} \in \mathcal{C}_i^t} SBD(T_{m,\ell}, \bar{\mathcal{C}}_i^t)}{|\mathcal{C}_j| + |\mathcal{C}_i^t|} \tag{4.3}$$

Finally, the matching procedure is executed in Algorithm 8, and the threshold update is executed in Algorithm 9.

4.2.3.2 Centroids Update

At this point we matched the arrival subsequences (in the current batch) with existing sequences in Θ. Let consider that cluster $\mathcal{C}_i^t$ has been matched with sequence $\bar{\mathcal{C}}_j \in \Theta$. We update sequence $\bar{\mathcal{C}}_j$ as described in Equation 2.1. We note $\bar{\mathcal{C}}_j{}^*$ the updated sequence $\bar{\mathcal{C}}_j$.

As mentioned earlier, we do not store set $\mathcal{C}_j$ in memory. We thus need to compute the shape update process dynamically. As described in Equation 2.1, the shape update process is computed using matrix S_j (of size ℓ_Θ^2). S_j is built by computing the dot product between all subsequences in the merged cluster $\mathcal{C}_j \cup \mathcal{C}_i^t$. Nevertheless, we can split the computation as follows:

Algorithm 8: Θ Centroids Update: *computeCentroid*

input : A set Θ, a clustering partition $\mathbb{C}^t$ on the batch $\mathbb{T}^t_{\ell_\Theta}$, a Boolean *init*.
output: A set Θ

foreach $\mathcal{C}_i^t \in \mathbb{C}^t$ **do**
  $newClusters \leftarrow \{\}$;
  if $(\exists(\bar{\mathcal{C}}_j, S_j, \tau_{c,j}, w_j, s_j) \in \Theta, sbd(\bar{\mathcal{C}}_j, \bar{\mathcal{C}}_i^t) < \tau_{c,j}) \wedge (init = False)$ **then**
    // The new cluster $\mathcal{C}_i^t$ can be merged with the existing cluster $\mathcal{C}_j$
    $S_j \leftarrow S_j + \sum_{T_{j',\ell_\Theta} \in \mathcal{C}_i^t} T_{j',\ell_\Theta}.T^T_{j',\ell_\Theta}$;
    $\bar{\mathcal{C}}_j \leftarrow \underset{\bar{\mathcal{C}}_j}{argmax} \frac{(\bar{\mathcal{C}}_j)^T.Q^T.S_j.Q.\bar{\mathcal{C}}_j}{(\bar{\mathcal{C}}_j)^T.\bar{\mathcal{C}}_j}$;
  else
    // The new cluster $\mathcal{C}_i^t$ cannot be merged with any existing cluster
    $S_i \leftarrow \sum_{T_{j,\ell_\Theta} \in \mathcal{C}_i} T_{j,\ell_\Theta}.T^T_{j,\ell_\Theta}$;
    $\bar{\mathcal{C}}_i \leftarrow Centroid(\mathcal{C}_i)$;
    $newClusters \leftarrow newClusters \cup (\bar{\mathcal{C}}_i, S_i, None, None, None)$;
  end
end
return $\Theta \cup newClusters$;

$$S_j = \sum_{T_{m,\ell_\Theta} \in \mathcal{C}_j} T_{m,\ell_\Theta}.T^T_{m,\ell_\Theta} + \sum_{T_{m,\ell_\Theta} \in \mathcal{C}_i^t} T_{m,\ell_\Theta}.T^T_{m,\ell_\Theta} \tag{4.4}$$

The left part of the above sum is already computed from the previous batch. The only new computation to be performed is the right part. Therefore, we just need to update matrix S_j by adding the sum of the dot product of the subsequences of the cluster $\mathcal{C}_i^t$ to be merged with $\bar{\mathcal{C}}_j$. By doing so, updating the cluster shape does not require storing all the subsequences in memory, but just matrix S_j. It results in a gain in memory space for large data series and execution time. Formally, matrix S_j (and therefore $\bar{\mathcal{C}}_j$) is updated as follows:

$$S_j^* \leftarrow S_j + \sum_{T_{m,\ell_\Theta} \in \mathcal{C}_i^t} T_{m,\ell_\Theta}.T^T_{m,\ell_\Theta} \tag{4.5}$$

Note that we just need to compute the initial S_j for all initial clusters and store them in memory. We then compute the update $\bar{\mathcal{C}_j}^*$ using the updated S_j^* at every new batch. The centroids computation and update are executed in Algorithm 8.

4.2.3.3 Weights Update

Once the sequences $\bar{\mathcal{C}}_j$ are updated, we can update their corresponding weights w_j. One could decide to update the weights like in Equation 4.1 using their current statistics (cardinality and distance to other sequences in Θ). On the specific case of data series without any

Algorithm 9: Θ **Parameters Update:** $computeParam$

input : A set Θ, a float α.
output: A set Θ

foreach $(\bar{\mathcal{C}}_i, S_i, \tau_{c,i}, w_i, s_i) \in \Theta$ **do**
  $w_i^t \leftarrow \frac{|\mathcal{C}_i|^2}{\sum_{\bar{\mathcal{C}}_j \in \Theta} sbd(\bar{\mathcal{C}}_j, \bar{\mathcal{C}}_i)}$;
  if $(\bar{\mathcal{C}}_i, S_i, \tau_{c,i}, w_i, s_i)$ *is a new cluster* **then**
    // Initialize the parameters for the new cluster $\mathcal{C}_i$
    $w_i, s_i \leftarrow w_i^t, |\mathcal{C}_i|$;
    $\tau_{c,i} \leftarrow \sum_{T_{m,\ell} \in \mathcal{C}_i^t} sbd(T_{m,\ell}, \bar{\mathcal{C}}_i^t)$;
  else
    // update the parameters for the merged cluster $\mathcal{C}_i$
    $w_i \leftarrow (1-\alpha) * w_i + \frac{\alpha * w_i^t}{max(1, \mathcal{A}_i - b_{size})}$;
    $\tau_{c,i} \leftarrow \frac{s_i * \tau_{c,i}}{|\mathcal{C}_i|} + \frac{(|\mathcal{C}_i| - s_i) * \sum_{T_{m,\ell} \in \mathcal{C}_i} sbd(T_{m,\ell}, \bar{\mathcal{C}}_i^t)}{|\mathcal{C}_i|}$;
    $s_i \leftarrow |\mathcal{C}_i|$;
  end
end
$return\ \Theta$;

changes of normal behaviors, this would be the good choice. However, if a new behavior is detected, one should be able to forget the previous behaviors (or reduce its importance). For that purpose, we dynamically update the weight using its previous value. We introduce a user parameter α with values between $[0, 1]$, such that it represents the rate of change. We note w_i^* the updated weight and compute it as follows:

$$
\begin{aligned}
w_i^t &\leftarrow \frac{|\mathcal{C}_i|^2}{\sum_{(\mathcal{C}_j, w_j) \in \Theta} sbd(\bar{\mathcal{C}}_j, \bar{\mathcal{C}}_i)} \\
w_i^* &\leftarrow (1-\alpha) * w_i + \frac{\alpha * w_i^t}{max(1, \mathcal{A}_i - b_{size})} \\
&\text{with: } \mathcal{A}_i = t - t_{last,i}
\end{aligned}
\tag{4.6}
$$

In the above definition, t is the time index of the current batch, and $t_{last,i}$ is defined as the temporal index of the latest (as regards to the time index) subsequence in cluster $\mathcal{C}_i$. Note that as previously expressed, $\mathcal{C}_i$ is not stored. However, it is trivial to count the number of subsequences ($|\mathcal{C}_i|$), and to compute $t_{last,i}$ without storing $\mathcal{C}_i$. Moreover, note that, at each iteration, we normalize the weight to have their sum equal to 1. As one can see, the new weight w_i^* will be a weighted mean (by α) between its old value w_i and its value at the current time w_i^t. If no new subsequence has been added to a cluster, then $w_i^t = w_i^{t-b_{size}}$. However, it also means that this cluster might correspond to an old (and potentially irrelevant now) behavior. It has to be taken into account, and we include a time decay component in the weight computation (as described in Equation 4.6). We have two different cases:

- $\mathcal{A}_i \leq b_{size}$: Cluster $\mathcal{C}_i$ contains at least one subsequence in the current batch. It

means that the cluster is still active. Thus $max(1, \mathcal{A}_i - b_{size}) = 1$, and no time decay is applied.

- $\mathcal{A}_i > b_{size}$: Cluster $\mathcal{C}_i$ does not contain any subsequence from the current batch. It means that the cluster might not be active anymore. We can thus start to apply some decay by dividing the current weight w_i^t by $\mathcal{A}_i - b_{size}$.

Depending on the value of α, the score of the clusters without any new subsequences will converge to zero, giving more importance to the currently activated clusters. In the specific case when an old behavior starts again to happen, its corresponding weight will increase faster (knowing that the cardinality of this cluster is already big). If one does not expect any old behavior to happen again, one can decide to remove clusters with scores approximately equal to zero. In practice, this is more efficient in memory. Nevertheless, for the remainder of this chapter, we do not remove any cluster. The weight update is performed in Algorithm 9.

4.2.4 Subsequence Scoring

At this point, we can update the set Θ at every new batch. We now describe how we compute the score for all the subsequences inside a given batch. For a given subsequence $T_{j,\ell} \in \mathbb{T}_\ell$ in the current batch (of length $\ell < \ell_\Theta$), we compute the following score:

$$d_j = \sum_{\bar{\mathcal{C}}_i} w^i * min_{x \in [0, \ell_\Theta - \ell]} \left\{ dist(T_{j,\ell}, (\bar{\mathcal{C}}_i)_{x,\ell}) \right\} \tag{4.7}$$

Even though the weights w^i are adjusted depending on the activity of their related clusters, a certain noise could be observed on the score. Based on $\bar{\mathcal{C}}_i$ shape and its possible evolution, the score values distribution might evolve as well. We thus normalize the score at each batch. For a subsequence $T_{j,\ell} \in \mathbb{T}_\ell$ in the current batch $\mathbb{T}^t_{\ell_\Theta}$, we compute the normalization as $d_j^* = \frac{d_j - \mu_t^*}{\sigma_t^*}$. We compute the estimated mean and standard deviation μ_t^* and σ_t^* as:

$$\begin{aligned} \mu_t^* &\leftarrow \alpha * \mu_t + (1 - \alpha) * \mu_{t - b_{size}} \\ \sigma_t^* &\leftarrow \alpha * \sigma_t + (1 - \alpha) * \sigma_{t - b_{size}} \end{aligned} \tag{4.8}$$

In the latter equation, μ_t and σ_t are the mean and the standard deviation of d_j over batch $\mathbb{T}^t_{\ell_\Theta}$. At each batch, we use the previously updated mean and standard deviation to adapt the distance to set Θ. Otherwise, in the unusual case when a batch contains a higher rate of anomalies than the previous (and future) batches, the normalization (without using the previous batch mean and standard deviation) may result in missing the anomalies in the batch. The score computation procedure is performed in Algorithm 10.

Algorithm 10: Score Computation: *computeScore*

input : A set Θ, a subsequence $\mathbb{T}^t_{\ell_\Theta}$, a float α, two floats μ, σ
output: A data series *score*, two floats μ^*, σ^*

$score \leftarrow []$ **foreach** $T_{j,\ell} \in \mathbb{T}^t_{\ell_\Theta}$ **do**
    // For each subsequence in the batch
    $score[j] \leftarrow \sum_{\bar{\mathcal{C}}_i} w^i * min_{x \in [0, \ell_\Theta - \ell]} \{dist(T_{j,\ell}, (\bar{\mathcal{C}}_i)_{x,\ell})\}$;
    $score[j] \leftarrow \frac{score[j] - \mu}{\sigma}$;
end
// Update mean and variance
$\mu^* \leftarrow \alpha * \mu(score) + (1 - \alpha) * \mu$;
$\sigma^* \leftarrow \alpha * \sigma(score) + (1 - \alpha) * \sigma$;
return $score, \mu^*, \sigma^*$;

4.2.5 Execution Time Complexity Analysis

In this section, we analyze the execution time complexity of the various steps of the SAND framework. Note that the time complexity of the k-Shape algorithm is $O(max(|\mathbb{T}^t_{\ell_\Theta}| * k * \ell_\Theta * log(\ell_\Theta), |\mathbb{T}^0_{\ell_\Theta}| * \ell^2_\Theta, k * \ell^3_\Theta))$ per iteration, with $|\mathbb{T}^t_{\ell_\Theta}| = b_{size}$.

Initialisation Step: We study the time complexity of the different steps separately. We first analyze the theoretical execution time needed to perform the initialization. The latter is composed of small computations related to the weights that the SBD distance uses between sequence of length ℓ_Θ with complexity $O(k^2 * \ell_\Theta log(\ell_\Theta))$. Thus, the complexity of the initialization step has the k-Shape algorithm as a bottleneck.

Batch Execution Time Complexity Analysis: At each batch, one need to run the k-Shape algorithm. Then at each step we need to compute the SBD distance between every new cluster and every sequence in Θ. The SBD computation complexity is $O(\ell_\Theta log(\ell_\Theta))$. Thus, the complexity of the first step is $O(|\Theta| * k * \ell_\Theta log(\ell_\Theta))$. As explained in the initialization section, the weight computation complexity is $O(k^2 * \ell_\Theta log(\ell_\Theta))$. Nevertheless, we always have $k \leq |\Theta|$, thus the weights computation step is negligible.

Then for a given cluster $\mathcal{C}_i$, the shape update process requires the computation of the matrices S_i and its eigendecomposition. Thus, the shape update operation has complexity $O(max(|\mathcal{C}_i| * \ell^2_\Theta, \ell^3_\Theta))$ with $|\mathcal{C}_i|$ being the number of subsequences in the cluster we want to extract the shape. We are storing previously computed matrices S_i in memory and we are computing the matrices S_i corresponding to the new clusters $\mathcal{C}^t_i$. For a given time index t, $\sum_{\mathcal{C}_i \in \mathcal{C}} |\mathcal{C}_i| = |\mathbb{T}^t_{\ell_\Theta}| = b_{size}$. Moreover, knowing that only k new clusters need to be merged, the shape update operation cannot be done more than k times. Thus the overall complexity of the shape update is $O(max(b_{size} * \ell^2_\Theta, k * \ell^3_\Theta))$. Therefore, this complexity does not depend on the time evolution, and the execution time needed remains constant for the entire stream.

Scoring Execution Time Complexity Analysis: The anomaly distance between subsequences of length ℓ and the set Θ computation is defined by the computation of Equation 4.7, which is bounded by $O((b_{size} - \ell + 1) * \ell_\Theta * |\Theta|)$ using the Fourier transform to compute efficiently the correlation and distances over overlapping subsequences.

Overall complexity: We now analyze the overall complexity of the batch operation. Note that some operations share the same complexity, so can be grouped. Overall, the time complexity of the batch update operation is defined as follows:

$$SAND = max \left\{ \begin{array}{l} b_{size} * k * \ell_\Theta * log(\ell_\Theta) \\ 2 * b_{size} * \ell_\Theta^2 \\ 2 * k * \ell_\Theta^3 \\ |\Theta| * k * \ell_\Theta * log(\ell_\Theta) \\ (b_{size} - \ell + 1) * \ell_\Theta * |\Theta| \end{array} \right\} \tag{4.9}$$

One should note that the only parameter that varies while time evolves is the size of the set Θ (involved in the two last lines of Equation 4.9). Nevertheless, this evolution is most likely to be slow. Thus, the overall complexity does not depend on the time evolution and remains constant regardless of the increasing number of batches. The important parameters on the execution time are the batch size b_{size}, the number clusters k and the length ℓ_Θ. Moreover, we note that the complexity is linear to the size of the batch, which implies that SAND scales gracefully to large batches.

4.2.6 Parameters Influences

In this section, we evaluate and analyze influences of SAND parameters on accuracy and execution time. Recall that there are only four main parameters (ℓ_Θ,k, b_{size}, and α) that may affect the anomaly detection accuracy. Note that two parameters can *jointly* influence the accuracy and, therefore, we vary two parameters simultaneously. We first start by analyzing parameters influences independently. Figure 4.3 depicts the Precision@k (Fig. 4.3(1)), the execution time in seconds to compute a batch (Fig. 4.3(2)) and the final number of clusters created (Fig. 4.3(3)) with a color range between black and yellow (with black the lowest and yellow the highest) for the double-normality datasets (results for single-normality follow similar trends).

4.2.6.1 Influence of the Centroids Length, ℓ_Θ

We first evaluate parameter ℓ_Θ as regards to parameter ℓ. We define $\ell_\Theta = a * \ell$. Figure 4.2 depicts the Precision@k and the execution time when a varies between 1 and 10 for double normality datasets (results for single-normality datasets follow similar trends). For accuracy purposes, ℓ_Θ should be greater than two times ℓ. Above $a > 2$, the Precision@k reaches its maximum value and stays constant. Thus, for a value of a above a given value, the length of the centroids does not have a significant impact on the accuracy. Moreover, as explained in Section 4.2.5, the centroids length does have an impact on the execution time. Thus, one needs to choose a large enough length to maximize the accuracy without increasing significantly the execution time. We pick $\ell_\Theta = 4 * \ell$.

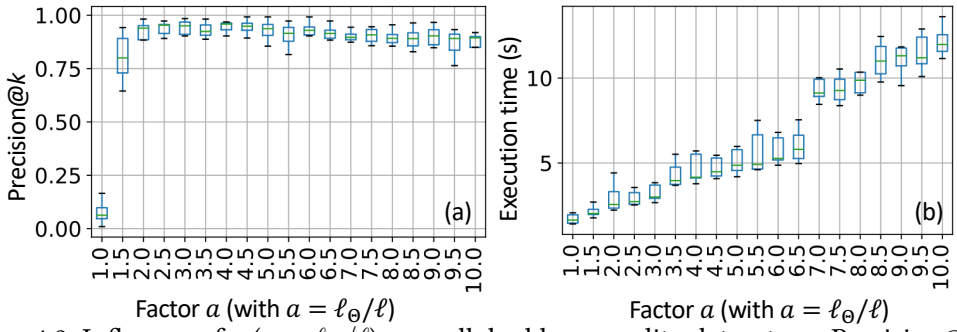


Figure 4.2: Influence of a ($a = \ell_\Theta/\ell$) over all double normality datasets on Precision@k (a) and execution time in sec (b).

4.2.6.2 Influence of Initial Number of Clusters, k

We analyze the influence of the SAND initial number of clusters k (independently) on the detection accuracy and the execution time per batch. The y-axis of Figure 4.3(b) depicts the evolution of the accuracy and execution time per batch when we vary the initial number of clusters k. Note that the other parameter is set to its default value ($b_{size} = 5000$). We notice that parameter k does not have any impact on the detection accuracy. Nevertheless, increasing k leads to a higher number of clusters after the final batch and a higher execution time per batch (as theoretically explained in Section 4.2.5). Therefore, a low initial number of clusters seems to be an optimal choice. In our experiments, we select $k = 6$.

4.2.6.3 Influence of Batch Size

We then measure the influence of the batch size on accuracy and execution time. As depicted in Figure 4.3(c) We note that a bigger batch requires more execution time. Nevertheless, we show in Section 4.3.6 that throughput remains constant to this parameter. We observe a drop in accuracy for small batches, but we also notice a slow reduction of accuracy while the batches increase. In this case, there is a change of normality in the middle of the data series. Thus small batches are more able to adapt to this change. We select the nearly optimal choice of $b_{size} = 5000$ for our experiments.

4.2.6.4 Influence of α

We measure the impact of α on accuracy and execution time (see Figure 4.3(b)). As expected, α does not have an impact, neither on the number of clusters created nor on execution time. However, when normality changes, the model needs to adapt fast enough. Thus, a high α leads to a more accurate result (for the single normality datasets, the impact on accuracy slowly decreases as α increases, since no adaptation is needed for these datasets). In our experiments, we use $\alpha = 0.5$, which provides good accuracy for both single and double normality datasets.

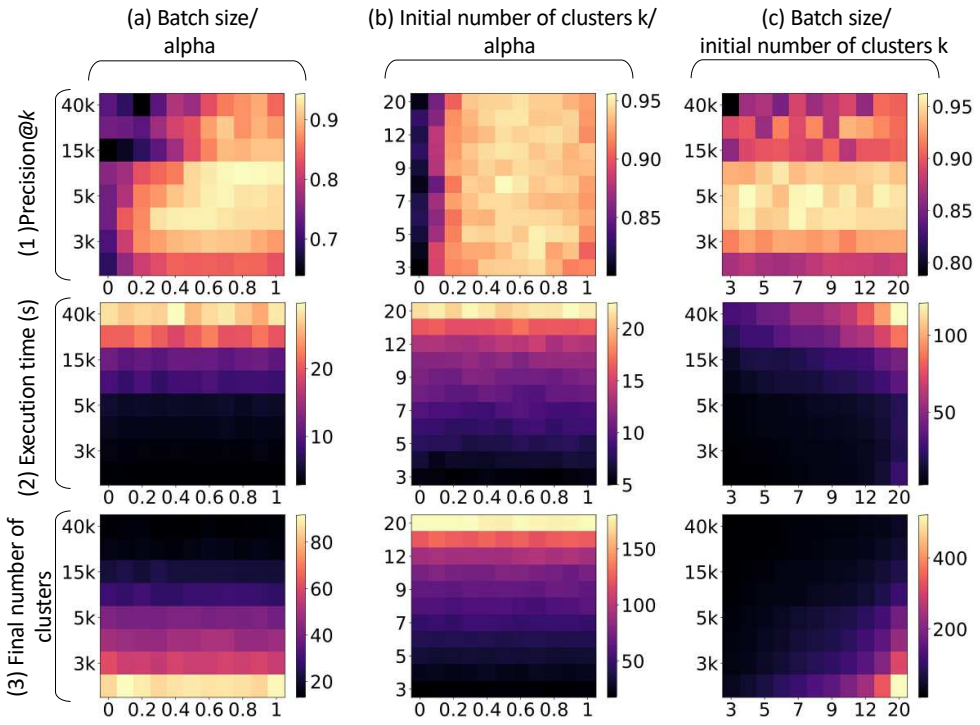


Figure 4.3: Influence of batch size b_{size}, rate of change α, and initial number of clusters k on accuracy (1st line), execution time (2nd line) and final number of clusters created (3rd line), over all double normality datasets.

4.2.6.5 Influence of Batch Size and α

We now evaluate the influence of the batch size joined with α. As previously underlined, the execution time and the number of clusters created are independent of α but only dependent on the batch size. We note that a high value of batch size joined with low values of α implies low accuracy for double normality datasets (Fig. 4.3(a.1)). On the contrary, we observe that a high value of α joined with a low value of batch size implies a lower accuracy for single normality datasets. Overall, parameters that are on the diagonal (such as $b_{size} = 5000$ and $\alpha = 0.5$) are optimal.

4.2.6.6 Influence of Initial Number of Clusters k and α

We then evaluate the initial number of clusters k joined with α (Figure 4.3(b)). As mentioned earlier, the execution time and the number of clusters created are independent of α. Moreover, for the double normality datasets, the initial number of clusters k does not have any impact on accuracy, while α does.

4.2.6.7 Influence of Batch Size and Initial Number of Clusters k

Finally, we measure the influence of batch size joined with the initial number of clusters k (see Figure 4.3(c)). One can see that the combination of a small-batch size and a high k leads to a significant number of clusters, and large batch sizes combined with a high initial number of clusters lead to a high execution time per batch. With regards to accuracy, only the batch size has an impact.

4.3 Experimental Evaluation

In this section, we compare the performance of SAND with state-of-the-art subsequence anomaly detection methods in terms of accuracy and efficiency. We measure the ability of SAND to detect abnormal subsequences in a benchmark of real datasets. Despite operating in a streaming setting, we confirm that SAND performs similarly to state-of-the-art (non-streaming) subsequence anomaly detection methods that operate over the entire dataset. We then demonstrate the shortcomings of the non-streaming methods, as well as the ability of SAND to adapt to changes of normality. We compare the scalability of SAND to state-of-the-art streaming methods for anomaly detection when we vary different parameters and show that SAND is an order of magnitude faster.

4.3.1 Implementation

4.3.1.1 Technical Details

We implemented our algorithms in C (compiled with gcc 5.4.0) and Python 3.6. The evaluation was conducted on a server with Intel Xeon CPU E5-2650 2.20GHz and 250GB RAM. We implemented our method as pip package.

4.3.2 Description of the evaluation metrics

We use the Precision@k as the accuracy measure. The latter is the ratio of correctly identified anomalies in the η subsequences corresponding to the η highest anomaly score. We set η as the number of anomalies in the datasets (as depicted in Table 3.3). Note that this parameter η is only used for evaluation purposes and is not required for practical usage. We then use the throughput metric and the execution time in seconds to evaluate the scalability. The throughput is defined as the number of subsequences that can be handled in one second and corresponds to an upper bound of the data acquisition speed of the method. The higher the throughput, the better the model will handle high-frequency data streams.

4.3.3 Description of the datasets

We benchmark our algorithm using real and synthetic datasets, for all of which a ground truth of annotated anomalies is available. We simulate streams by selecting static data series. We carefully select two specific types of data series. We first select several real data series. The first data series is a Simulated Engine Disks data (SED) from the NASA Rotary Dynamics Laboratory [2] representing disk revolutions recorded over several runs (3K rpm speed). We also include MIT-BIH Supraventricular Arrhythmia Database (MBA) [40, 83], which are electrocardiogram recordings from four patients containing multiple instances of two different kinds of anomalies (either supraventricular contractions or premature heartbeats). All the previously enumerated datasets and their characteristics are grouped in Table 3.3.

To evaluate the capacity of our method to adapt to changes over time, we create synthetic datasets that contain more than one specific normality. We note them $Double$-Normality, $Triple$-Normality, up until $Sextuple$-Normality. We build them by concatenating our real single normality datasets. We thus perform two kinds of concatenation. We first concatenate different datasets from the same domain (i.e., two Electro-Cardiogram from two different patients) to evaluate methods to adapt to subtle changes. We then concatenate datasets from different domains (i.e., Electro-Cardiogram with SED) to evaluate methods to adapt to drastic changes.

4.3.4 Description of the baselines

We first compare with four state-of-the-art static methods (i.e., methods that take as input the entire data series): Isolation Forest (named IF and described in Section 2.5.3.2), NormA (defined in Section 3.3), Series2Graph (named S2G and defined in Section 3.4), and STOMP (described in Section 2.5.4.4); as detailed below, we use the parameters suggested in the original papers. For IF, we use 100 trees as explained in [75]. For NormA, as described in Section 3.3, we use the default parameters for sampling rate, $r = 0.4$, and subsequence length, $\ell_{N_M} = 4 * \ell$. For S2G, we use as parameters local convolution $\lambda = 1/3 * \ell_G$, bandwidth h (set using Scott's rule [102]), and number of angles $r = 50$ (as defined in Section 3.4). We finally compare SAND to NormA-mn, a variation of NormA, where we adapted the computation of the anomaly score based on the average anomaly score in a given window length: we compute the anomaly score of a subsequence $T_{i,\ell}$, and we subtract the average anomaly score of all subsequences within the interval $[i - 2 * \ell_\Theta, i + 2 * \ell_\Theta]$. This adaptation of the scoring step enables NormA to operate on multi-normality datasets.

We then compare SAND to dynamic methods (i.e., methods that receive subsequences of the data series incrementally). We first build baselines from the state-of-the-art static methods called NormA-batch and S2G-batch, which operate locally (and independently) on each new arriving batch. We also compare SAND to two state-of-the-art dynamic methods: IMondrian Forest(mentioned in Section 2.5.3.3) and STAMPI (mentioned in Section 2.5.4.5). The first method is an alternative to Isolation Forest that uses a tree structure (called Mondrian tree, initially proposed for classification purpose [66]) with the characteristic to be incrementally modified while new points (subsequences in our case) arrive. Similarly to

Isolation Forest, we use 100 different trees. Similar to STOMP, the second approach is using the nearest neighbor distance to identify abnormal points. On the contrary to STOMP, this method has the specificity of being updatable incrementally. One can either keep track of all the previous points and update the distance profile until the end of the stream or keep track of the distance profile over a fixed window length (called batch size in our case). We consider using a fixed window length equal to the batch size for the following reason: (i) keeping track of the entire distance profile requires a large amount of computation and makes the latency increase quadratically. (ii) STAMPI (using the discord definition) can suffer from the fact that similar anomalies can happen in a stream. Therefore, keeping the old distance profile might lead to a higher false-negative rate than keeping only a fixed window length. The second point is confirmed by Table 5.2. For SAND, we set the additional parameters as follows: $\alpha = 0.5$, $\ell_\Theta = 4 * \ell$ and $k = 6$.

4.3.5 Accuracy Evaluation

4.3.5.1 Clustering accuracy evaluation

In this section, we evaluate the clustering accuracy of our extension of the k-Shape algorithm. For that purpose, we use all the UCR datasets [30]. We run the original k-Shape on the entire dataset, and we run our extended k-Shape on 1/10th of the dataset at each new batch incrementally. Figure 4.4(a) depicts the rand score [93] accuracy comparison between the original and our extended k-Shape algorithm. We observe that our algorithm has a similar rand score on average as the usual k-Shape. Thus our technique provides a way to use the original k-Shape for streaming scenarios with the same accuracy. Moreover, Figure 4.4(b) depicts the average execution time (in seconds) for a simple solution for incremental k-Shape (i.e., storing all subsequences to compute the centroids from scratch at every new batch), and our proposed solution for incremental k-Shape (i.e., without storing all the subsequences). The results confirm the benefits of our proposed solution for an incremental k-Shape algorithm.

4.3.5.2 SAND accuracy evaluation

We now compare the Precision@k of our method with several other methods, both static and dynamic. All methods share the subsequence length ℓ as the main parameter. For each dataset, we set the subsequence length $\ell = \ell_A$ as shown in Table 3.3. We use a batch size of 5000 points for IMondrian Forest, STAMPI, and our approach. Table 5.2 depicts the Precision@k accuracy of the aforementioned methods for our datasets corpus. One can notice that for single normality datasets, static methods NormA, Isolation Forest, and Series2Graph (that are using the entire series to build their model), have good performances. Nevertheless, online methods IMondrian Forest and SAND are still performing well by being slightly less accurate than the static methods. STAMPI has a medium Precision@k due to the limitations caused by similar anomalies in the single normality datasets. However, STAMPI has better performance than STOMP, which confirms that using a fixed sliding window limits the number of similar anomalies and provides better accuracy. Finally, we

	Static					*Streaming*				
Data Series	**NormA**	**IF**	**STOMP**	**S2G**	**NormA-mn**	**NormA-batch**	**S2G-batch**	**IMondrian F**	**STAMPI**	**SAND**
Single Normality (100,000 points)										
MBA(803)	0.99(0.01)	**1.00(0.00)**	0.72	**1.00**	0.97(0.01)	0.70(0.07)	0.97	0.99(0.01)	0.46	0.97(0.02)
MBA(805)	**0.99(0.00)**	**0.99(0.01)**	0.10	**0.99**	**0.99(0.00)**	0.82(0.05)	0.90	0.96(0.02)	0.35	0.98(0.01)
MBA(806)	0.86(0.02)	0.75(0.06)	0.59	**1.00**	0.88(0.02)	0.74(0.07)	0.70	0.85(0.03)	0.66	0.80(0.03)
MBA(820)	**0.98(0.01)**	0.92(0.03)	0.90	0.91	0.97(0.00)	0.74(0.08)	0.71	0.95(0.02)	0.84	0.96(0.01)
SED	0.92(0.05)	0.65(0.02)	0.57	**1.00**	0.98(0.01)	0.80(0.06)	0.90	0.31(0.05)	0.87	0.96(0.00)
Average	0.95	0.86	0.58	**0.98**	0.96	0.76	0.84	0.81	0.64	0.93
Double Normality (200,000 points)										
Same Domain										
MBA(803 + 805)	0.91(0.10)	0.53(0.03)	0.32	**0.99**	0.95(0.02)	0.76(0.03)	0.94	0.97(0.01)	0.40	0.94(0.01)
MBA(803 + 806)	0.70(0.01)	0.75(0.00)	0.58	0.75	0.89(0.03)	0.68(0.02)	0.84	0.87(0.02)	0.52	**0.96(0.02)**
MBA(803 + 820)	0.83(0.27)	0.75(0.05)	0.78	0.76	0.92(0.01)	0.66(0.04)	0.81	**0.93(0.01)**	0.67	0.88(0.00)
MBA(805 + 806)	0.74(0.00)	0.68(0.04)	0.20	0.73	0.85(0.01)	0.76(0.06)	0.41	0.81(0.04)	0.44	**0.95(0.01)**
MBA(805 + 820)	0.76(0.03)	0.49(0.04)	0.51	0.51	**0.97(0.01)**	0.71(0.02)	0.71	0.94(0.01)	0.61	0.90(0.02)
MBA(806 + 820)	0.77(0.02)	0.78(0.00)	0.83	0.81	0.92(0.01)	0.75(0.02)	0.20	0.51(0.04)	0.79	**0.93(0.01)**
Average	0.78	0.66	0.54	0.76	0.92	0.72	0.65	0.84	0.57	**0.93**
Different Domains										
MBA(803) + SED	0.56(0.19)	0.45(0.00)	0.67	0.06	0.60(0.14)	0.72(0.02)	0.84	0.65(0.08)	0.64	**0.96(0.01)**
MBA(805) + SED	0.69(0.20)	0.37(0.01)	0.30	0.11	0.87(0.05)	0.84(0.03)	0.41	0.68(0.09)	0.57	**0.95(0.02)**
MBA(806) + SED	0.74(0.05)	0.57(0.01)	0.62	0.07	**0.84(0.01)**	0.79(0.02)	0.72	0.46(0.04)	0.79	0.80(0.03)
MBA(820 + SED	0.91(0.03)	0.38(0.01)	0.82	0.10	**0.92(0.02)**	0.72(0.06)	0.40	0.52(0.01)	0.85	0.88(0.00)
Average	0.72	0.44	0.60	0.09	0.81	0.77	0.59	0.58	0.71	**0.90**
Triple Normality (300,000 points)										
Same Domain										
MBA(803 + 805 + 806)	0.84(0.00)	0.43(0.02)	0.37	0.82	0.84(0.01)	0.71(0.04)	0.59	0.82(0.01)	0.44	**0.92(0.03)**
MBA(803 + 805 + 820)	0.60(0.23)	0.37(0.02)	0.54	0.63	0.86(0.06)	0.66(0.02)	0.79	**0.95(0.00)**	0.56	0.88(0.02)
MBA(803 + 806 + 820)	0.83(0.00)	0.68(0.05)	0.74	0.67	0.82(0.04)	0.67(0.06)	0.47	0.69(0.06)	0.66	**0.91(0.01)**
MBA(805 + 806 + 820)	0.60(0.12)	0.41(0.02)	0.53	0.44	0.85(0.02)	0.65(0.05)	0.25	0.70(0.02)	0.61	**0.94(0.00)**
Average	0.72	0.47	0.54	0.64	0.84	0.69	0.52	0.79	0.57	**0.91**
Different Domains										
MBA(803 + 805) + SED	0.60(0.13)	0.26(0.00)	0.41	0.10	0.60(0.12)	0.77(0.03)	0.52	0.75(0.02)	0.53	**0.95(0.00)**
MBA(803 + 806) + SED	0.67(0.14)	0.34(0.01)	0.62	0.06	0.67(0.04)	0.70(0.03)	0.73	0.67(0.05)	0.64	**0.88(0.00)**
MBA(803 + 820) + SED	0.60(0.12)	0.26(0.00)	0.75	0.10	0.64(0.02)	0.64(0.06)	0.52	0.66(0.01)	0.72	**0.87(0.01)**
MBA(805 + 806) + SED	0.75(0.11)	0.31(0.00)	0.35	0.09	**0.79(0.02)**	0.78(0.02)	0.40	0.66(0.05)	0.59	0.76(0.01)
MBA(805 + 820) + SED	0.62(0.06)	0.24(0.00)	0.54	0.09	**0.94(0.01)**	0.71(0.03)	0.26	0.73(0.01)	0.67	0.91(0.02)
MBA(806 + 820) + SED	0.37(0.11)	0.31(0.00)	0.78	0.10	0.82(0.02)	0.74(0.05)	0.39	0.40(0.08)	0.81	**0.86(0.00)**
Average	0.60	0.28	0.58	0.09	0.74	0.72	0.47	0.65	0.66	**0.87**
Quadruple Normality (400,000 points)										
Same Domain										
MBA(803 + 805 + 806 + 820)	0.86(0.01)	0.32(0.02)	0.53	0.57	0.86(0.03)	0.67(0.05)	0.42	0.70(0.01)	0.57	**0.95(0.00)**
Different Domains										
MBA(803 + 805 + 806) + SED	0.47(0.03)	0.23(0.00)	0.44	0.10	0.74(0.06)	0.74(0.01)	0.50	0.72(0.01)	0.55	**0.91(0.01)**
MBA(803 + 806 + 820) + SED	0.30(0.10)	0.23(0.00)	0.71	0.09	0.60(0.27)	0.67(0.04)	0.50	0.58(0.02)	0.71	**0.85(0.04)**
MBA(805 + 806 + 820) + SED	0.54(0.01)	0.21(0.00)	0.55	0.09	0.55(0.30)	0.66(0.03)	0.28	0.60(0.01)	0.67	**0.90(0.02)**
Average	0.43	0.22	0.57	0.09	0.63	0.69	0.43	0.63	0.64	**0.89**
Quintuple Normality (500,000 points)										
Different Domains										
MBA(803 + 805 + 806 + 820) + SED	0.40(0.01)	0.16(0.05)	0.55	0.08	0.81(0.05)	0.67(0.04)	0.38	0.69(0.02)	0.62	**0.90(0.00)**

Table 4.2: Precision@k accuracy for NormA (and NormA-batch), Isolation Forest (IF), STOMP, S2G (and S2G-batch), IMondrian Forest, STAMPI, and SAND applied to our datasets corpus (including concatenations of different datasets from same and different domains). The standard deviation of 10 runs is reported in parentheses.

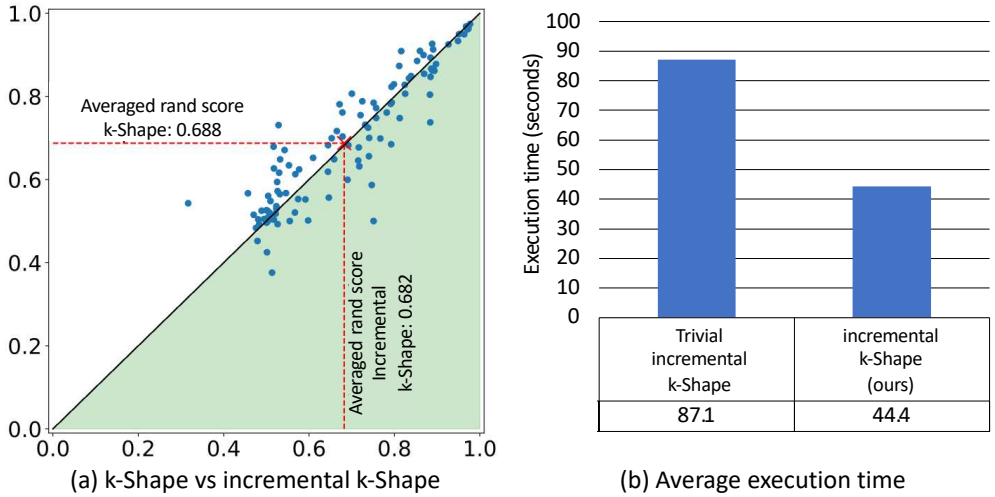


Figure 4.4: Comparison between the original and our extended k-Shape on rand score (a), and execution time (b).

note that SAND outperforms both STAMPI and IMondrian Forest, and performs equal or very close to Series2Graph and NormA. Moreover, one can notice that the online adaptation of NormA and S2G (NormA-batch and S2G-batch) are not performing as well as their respective static version for the single normality datasets.

Regarding double normality datasets, we observe that the static methods suffer from a significant drop in accuracy, while the online methods have on average similar Precision@k accuracy to the single normality datasets. However, we note that both NormA-batch and S2G-batch perform better than NormA and S2G for double normality datasets from different domains, but not for datasets from the same domain. Only NormA-mn is robust to multiple normality datasets, but SAND is still more accurate. Moreover, SAND outperforms both IMondrian Forest and STAMPI. More precisely, SAND significantly outperforms the competitors (both static and online) for double normality datasets created by concatenated datasets of different domains. The latter underlines the superior adaptability of SAND, regardless of the dataset composition.

The same observation can be made for triple, quadruple, and quintuple normality datasets, for which the Precision@k contrast between SAND and the other state-of-the-art methods is even stronger. More generally, one should underline that static methods (Isolation Forest, Series2Graph, and NormA) see their Precision@k dropping while the number of different normality increases, in opposite to online methods (IMondrian Forest, STAMPI, and SAND) that seem to be more stable. Only STOMP seems to have a stable accuracy while the number of normality sections increases.

A careful look can be addressed to Series2Graph performance on multiple normality datasets. Similar to NormA and Isolation Forest, one can notice an accuracy drop. Nevertheless, this drop is significantly bigger for datasets concatenated from different domains. It underlines a limitation of Series2Graph to data series that have a strong heterogeneous range of values through time. The embedding space needs to be changed to adapt to this specific case.

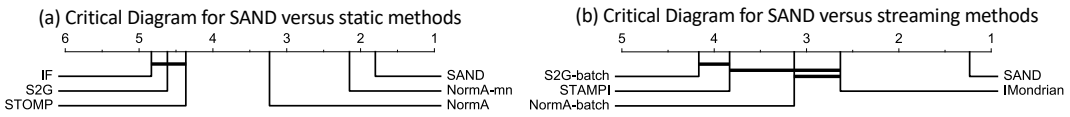


Figure 4.5: Critical difference diagrams using a Wilkoxon pair-wised signed rank test (with $\alpha = 0.05$) on both single and multiple normality datasets.

To conclude, SAND has equivalent accuracy for single normality datasets with state-of-the-art static methods and significantly outperforms both static and online state-of-the-art methods for datasets containing normality changes. To assess the significance of these differences, Figure 4.5 depicts a critical difference diagram computed using a Wilcoxon pair-wised signed-rank test (with $\alpha = 0.05$) on single and multiple normality datasets. Overall, Figure 4.5(a) underlines that SAND achieves the highest rank of all methods and statistically outperforms all previous static state-of-the-art methods (i.e., thick lines connect methods performing similarly; SAND outperforms methods with statistically significant differences). Similarly, Figure 4.5(b) confirms that SAND outperforms state-of-the-art online methods as well.

4.3.6 Time Performance Evaluation

In this section, we evaluate the scalability of our methods and the state-of-the-art streaming method analyzed in the previous section. For that purpose, we first assess the global execution time (in seconds) needed to perform the update operation of the model and the subsequences scoring (in batch). We then measure the throughput when we vary different parameters.

We first evaluate the execution time needed for the model update and batch scoring. Figure 4.6 presents results on the double normality datasets (results with single normality datasets exhibit similar trends; we omit them for brevity). Figure 4.6(d) depicts the global execution time for the model update and the batch scoring for SAND (in blue), IMondrian Forest (in red), and STAMPI (in green). One should note that STAMPI performs both the model update and the batch scoring at the same time. We thus report zero as the model update execution time of STAMPI. Nevertheless, SAND is significantly faster than IMondrian Forest for the model update operation, up to three orders of magnitude faster for the batch scoring operation. Overall, the total execution time (represented by the dotted lines) of SAND is significantly lower than the two other methods.

We now measure the influence of the batch size on throughput (using a fixed subsequence length of 75). Figure 4.6(a) illustrates with the dotted lines the standard deviation envelope over all the double normality datasets. SAND throughput remains stable as the batch size increases. It confirms our expectation (cf. Section 4.2.5). On the contrary, STAMPI and IMondrian forest throughput decreases. Thus, SAND has a significantly higher throughput for large batches (>5000 subsequences).

We then evaluate the influence of the subsequence length on the throughput (while keeping an almost constant batch size of approximately 20,000 subsequences). The latter

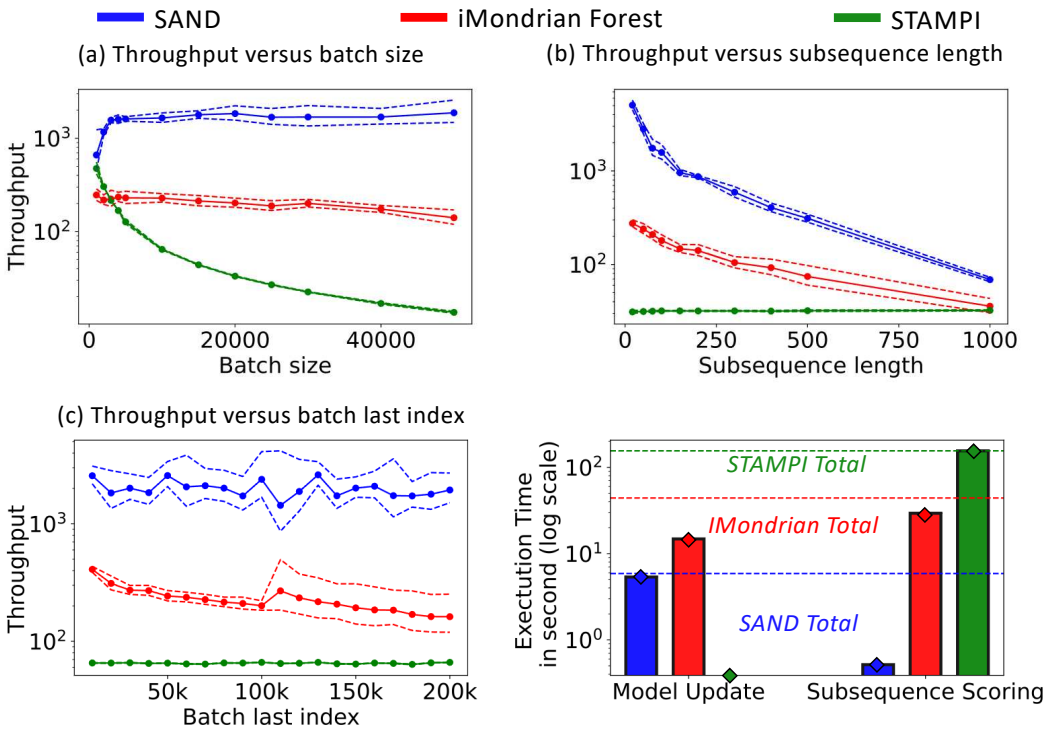
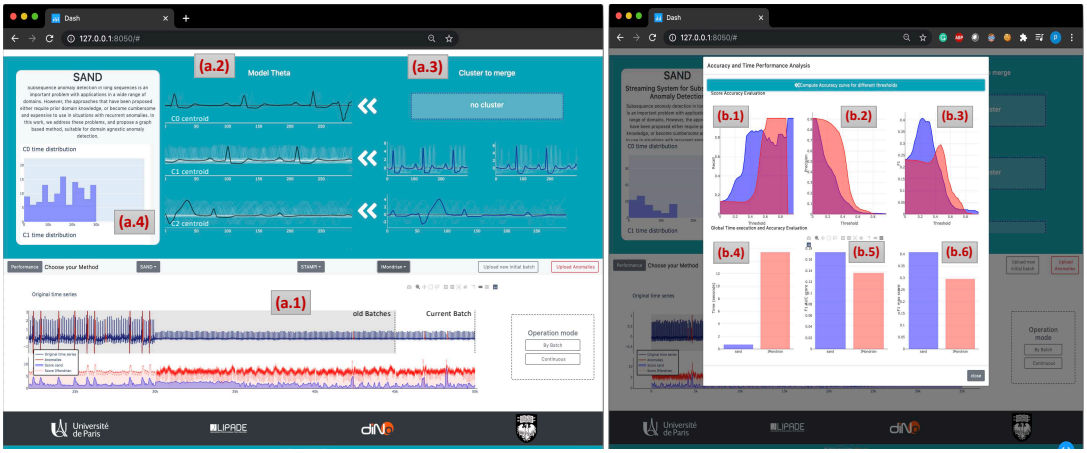


Figure 4.6: Throughput vs batch size (with fixed subsequence length $\ell = 75$), subsequence length (with fixed batch size $b_{size} = 20000$), and position in the stream (with $\ell = 75$ and $b_{size} = 10000$).

is illustrated in Figure 4.6(b). We notice that the throughput of SAND and IMondrian is reducing when the subsequence length is increasing. The latter has been predicted by the high importance of the length in Equation 4.9. On the contrary, STAMPI throughput is constant, whatever the value of the subsequence length. We notice that the throughput of IMondrian Forest is equivalent to the one of STAMPI for a subsequence length of 1000 points for double normality datasets. Overall, SAND throughput is significantly higher for small subsequence length (up to 200 points) and remains higher for subsequence length up to 1000 points. We observe that most of the subsequence anomalies are of moderate size (e.g., for cardiology datasets [40, 83], abnormal heartbeats are usually shorter than 200 points) and rarely measure up to 1000 points (e.g., electrical consumption datasets, patient respiration, and Space Shuttle Marotta Valve [57, 103]).

We finally investigate the evolution of throughput as the stream evolves (we use 10,000 as batch size and 75 as subsequence length). Figure 4.6(c) shows the throughput of SAND, IMondrian Forest, and STAMPI at each batch (represented with its index in time on the x-axis). Despite the variability between different datasets (represented by the dotted line envelope), we note that SAND throughput is relatively constant (this is true for both multi-normality and single normality datasets). It confirms the statement made in Section 4.2.5, and proves empirically that the size evolution of $|\Theta|$ does not affect execution time. Similarly, STAMPI throughput is constant over time. On the contrary, IMondrian For-



(a) Screenshot SAND system main frame

(b) Screenshot when performance button is pressed

Figure 4.7: Screenshots of the user interfaces of SAND.

est throughput reduces over time. We observe a perturbation on its throughput for double normality datasets, taking place when the normality changes (at index 100,000). At that point, the insertion of elements in the Mondrian trees is faster because these elements are significantly different (larger distances) than current elements and, therefore, the IMondrian Forest throughput increases. Nevertheless, it starts rapidly to decrease on average. Overall, SAND throughput is constantly one order of magnitude larger than the one of IMondrian Forest and STAMPI.

4.3.7 User Interface: StandAlone Web Application

We also proposed a user interface enabling the user to interact, visualize the detected anomalies and the different inner computational steps of SAND [16]. The GUI is a standalone web application developed using Python 3.6 and the Dash framework [29]. Figure 4.7 displays the different frames of the GUI. The mainframe is shown in Figure 4.7(a). Once the user opens the web application, he can upload a dataset (as well as the anomaly annotations, if available). At first, only the initial batch is displayed. When new batches arrive, by default, only the current and the three newest batches are displayed (as in Figure 4.7(a.1)). Nevertheless, the user can navigate through older sections of the data series. Moreover, if annotations are provided, they will be colored in red. The user can then change the values of ℓ and the batch size b_{size} by clicking on the SAND dropdown in the navigation bar. Then, by clicking on the initialization button, the anomaly score on the initial batch will be displayed under the data series plot. Moreover, model Θ (the centroids of the model) is depicted in Figure 4.7(a.2). By clicking on one of the centroids, the user can visualize the corresponding weight and the time distribution of the subsequences contained in the selected centroid (Figure 4.7(a.4)). Once the model is initialized, it is ready to be updated with new batches arriving. The user then may (i) decide to manually add the next batch (by clicking on the "by batch" button) or (ii) have the system process batches continuously (by clicking on the "continuous" button). When a new batch arrives, both the data series and

the anomaly score plots are updated. The model is also updated, and the new clusters from the current batch that need to be merged with an existing cluster will be aligned with it (under the label "cluster to be merged" as in Figure 4.7(a.3)). The new clusters appear under “clusters to create.”

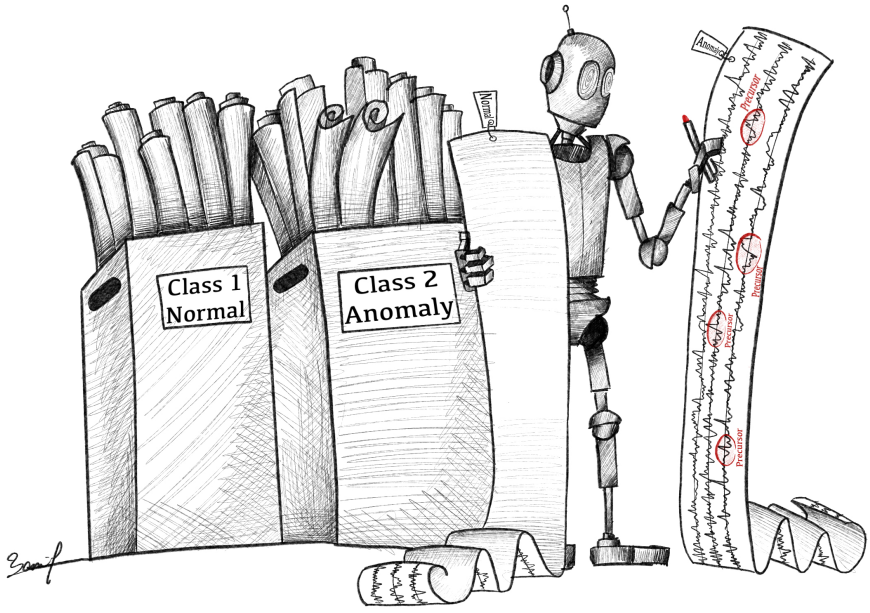
The user can also run other anomaly detection methods: STAMPI (mentioned in Section 2.5.4.5) and Isolation Mondrian Forest (named IMondrian and mentioned in Section 2.5.3.3). Their anomaly scores will be shown together with the SAND anomaly scores (Figure 4.7). If annotations are provided, performance analysis can be carried out by clicking on the performance button: a new frame will appear (Figure 4.7(b)) displaying accuracy (Figures 4.7(b.1),(b.2) and (b.3)) and time execution evaluations (Figure 4.7(b.4)).

4.4 Summary

To conclude, we tackled in this chapter the specific case of unsupervised subsequence anomaly detection for streaming data series. We first analyzed the limitation of state-of-the-art approaches and our approaches described in Section 3.3 and Section 3.4. We then introduced SAND, a novel unsupervised approach for subsequence anomaly detection in streaming sequences. SAND is based on a set representation of the subsequences in a data stream (inspired from NormA described in Section 3.3) and can detect both single and recurrent anomalies. We proposed a user interface implementation of our approach that simplifies the usage and facilitates the comprehension of SAND. Finally, we conducted an experimental analysis of several synthetic and real datasets. These experiments demonstrate the benefits of our approach in terms of efficiency and accuracy.

Chapter 5

Supervised Identification of precursors of anomaly



Chapter Outline and Abstract

5.1 Class Activation Map: an interesting tool for anomaly precursor identification . . . 123
5.1.1 Limitations for Multivariate Data Series anomaly detection 123
5.2 A first Baseline for Multivariate Data Series . . . 123
5.2.1 cCNN: a first architecture . . . 124
5.2.2 Limitations . . . 124
5.3 Proposed Approach: Dimension-wise Class Activation Map . . . 124
5.3.1 Dimension-wise Architecture . . . 125
5.3.2 Limitation of CAM for Dimension-wise Architecture . . . 127
5.3.3 Computation of dCAM . . . 128
5.4 Experimental Evaluation . . . 132
5.4.1 Experimental Setup . . . 132
5.4.2 Baselines and Training Setup . . . 134
5.4.3 Classification Accuracy evaluation . . . 134
5.4.4 Discriminant Feature identification evaluation . . . 136
5.4.5 C-acc versus Dr-acc . . . 137
5.4.6 Execution time evaluation . . . 139
5.4.7 Use Case: Surgeon skills explanation . . . 140
5.5 Summary . . . 143

In the previous chapter, anomalies were considered unknown. However, one can consider that experts precisely know which event he wants to detect and has a collection of data series that corresponds to these anomalies. In that case, we have a database of anomalies at our disposal. As a consequence, one can decide to adopt supervised methods. A question that naturally arises is the following one: is it possible to detect subsequences that happened before the known anomaly that might lead to an explanation of the anomaly (and potentially predict it). If the supervised model is trained to classify anomalies from normal data series, the features and subsequences that discriminate the anomaly from the normal class can be seen as symptoms or precursors (if these subsequences happened before the anomaly). Thus precursors detection for known anomalies can be tackled as a discriminant features identification for a classification model. For some Convolutional Neural Network-based models, the Class Activation Map (CAM) can be used as an explanation for the classification result. CAM has been used for highlighting the parts of an image that contribute the most to a given class prediction, and has also been adapted to data series. Nevertheless, CAM for data series suffers from one important limitation. Since CAM is a univariate time series with high values aligned with the subsequences of the input that contribute the most for a given class identification, in the specific case of multivariate data series as input, no information can be retrieved from CAM on the level of contribution of specific dimensions. In this chapter, we address the above-mentioned limitations and propose a novel data organization and a new CAM that highlights both the temporal and dimensional informations.

5.1 Class Activation Map: an interesting tool for anomaly precursor identification

In this chapter, we assume that the user has a data series example of abnormal and normal data series at his disposal. Thus, as already mentioned, one can train a convolutional neural network (CNN, ResNet, or InceptionTime) to classify anomalies from normality and then use Class Activation Map (CAM) to identify subsequences that can explain the anomalies. Thus, such subsequences correspond to features that the model used to discriminate the two classes (normal and anomaly). In the general case, it corresponds to tackle the task of discriminant feature identification for classification models. Thus, we now formulate the problem analyzed in this chapter as follows:

Problem 3 (Multivariate Data Series Classification Explanation)**.** *Given a set $\mathcal{T}$ of multivariate data series $T = \{T^{(0)}, T^{(1)}, ..., T^{(D)}\}$ of D dimensions belonging to classes $\mathcal{C}_j \in \mathcal{C}$, and a model $f : \mathcal{T} \rightarrow \mathcal{C}$, find a function $g(T, f)$ that returns a multivariate series $g(T, f, \mathcal{C}_j) = \{T^{(0)'}, T^{(1)'}, ..., T^{(D)'}\}$, in which $T^{(i)'}$ is a series that has high values if the corresponding subsequences in T_i discriminate T of belonging to another class than $\mathcal{C}_j$.*

5.1.1 Limitations for Multivariate Data Series anomaly detection

As mentioned earlier, a CAM that highlights the discriminative subsequences of class $\mathcal{C}_j$, $CAM_{\mathcal{C}_j}(T)$, is a univariate data series. The information provided by $CAM_{\mathcal{C}_j}(T)$ is sufficient for the case of univariate series classification, but not for multivariate series classification. Even though the significant temporal index may be correctly highlighted, no information can be retrieved on which dimension is significant or not. Therefore, we focus in this chapter on proposing a method that solves the issue as mentioned above.

5.2 A first Baseline for Multivariate Data Series

A new solution would be to decide to use a 2D convolutional neural network with kernel size $(\ell, 1)$, such that each kernel slides on each dimension separately. Thus, for an input data series T, $\mathbf{A}_m(T)$ would become a multivariate data series for the variable $m \in [1, n_f]$, and $A_m^{(d)}(T) \in \mathbf{A}_m(T)$ would be a univariate time series that would correspond to the dimension d of the initial data series. Note that one can use similar kernels of size (ℓ, ℓ) as those used for pictures. It implies that subsequences of length ℓ for ℓ dimensions will be merged by one kernel. However, the ordering of dimensions in data series is not (most of the time) important. Thus using kernels of size (ℓ, ℓ) will force the user to choose the most meaningful order for the dimensions, which is very difficult in practice.

Symbol	Description
T	a data series
$\|T\|$	length of T
$T^{(i)}$	i^{th} dimension of T
D	number of dimensions
$\mathcal{C}$	set of all classes
$\mathcal{C}_j$	one class of $\mathcal{C}$
$w_m^{\mathcal{C}_j}$	weight of connecting the m^{th} convolutional layer and class $\mathcal{C}_j$ neuron
$A_m(T)$	output of the m^{th} convolutional layer for input T
$z_{\mathcal{C}_j}(T)$	output of $\mathcal{C}_j$ neuron for input T
$CAM_{\mathcal{C}_j}(T)$	Class Activation Map for class $\mathcal{C}_j$ and input T
Σ_T	set of all possible permutations of T dimensions
S_T^i	T with one possible permutation of its dimensions ($S_T^i \in \Sigma_T$)
k	number of permutations
n_g	number of permutations that have been correctly classified by the model

Table 5.1: dCAM Table of symbols.

5.2.1 cCNN: a first architecture

We call this solution cCNN, and we use cCAM to refer to the corresponding Class Activation Map. Figure 5.1 illustrates cCNN architecture and cCAM. This architecture is equivalent to train one single CNN on each dimension independently. Note that any other architectures, such as ResNet and InceptionTime, with a Global Average Pooling (GAP) can be used. We denote these baselines as cResNet and cInceptionTime.

5.2.2 Limitations

New limitations arise from this solution. First, the dimensions are not compared together: each kernel of the input layer will take as input only one of the dimensions at a time. Thus, features depending on more than one dimension will not be detected. However, cCNN, cResNet and cInceptionTime represent interesting first baselines to address the problem. We further study and demonstrate the limitations of these baselines in the experimental section. Table 5.1 summarizes the symbols we use in this chapter.

5.3 Proposed Approach: Dimension-wise Class Activation Map

In this chapter, we describe our proposed approach, dCAM (dimension-wise Class Activation Map). Based on a new architecture that we call dCNN (as well as the variant architectures dResNet and dInceptionTime), dCAM aims to provide a multivariate CAM pointing to the discriminant features within each dimension. Contrary to the previously described baseline, one kernel on the first convolutional layer will take as input all the dimensions

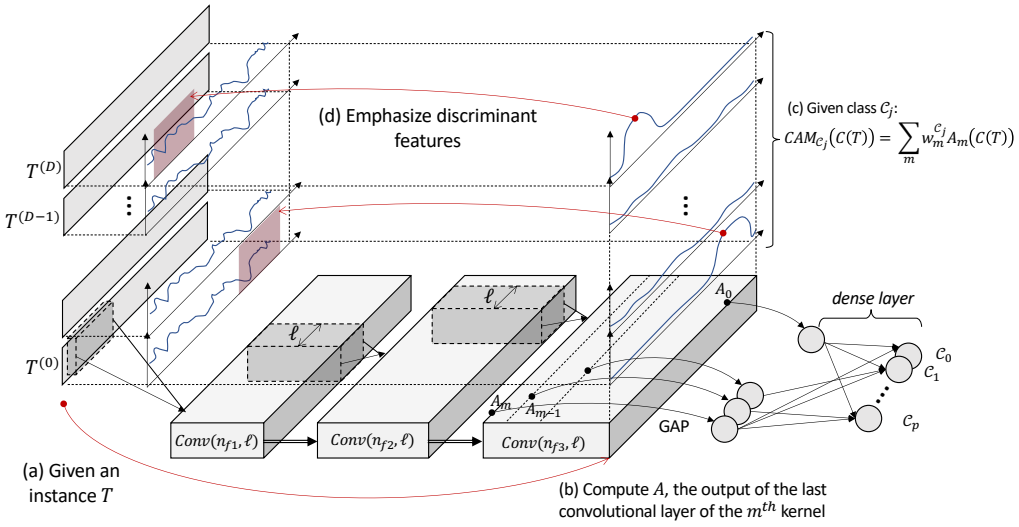


Figure 5.1: Illustration of Class Activation Map for cCNN architecture with three convolutional layers (n_{f1}, n_{f2}, and n_{f3} different kernels respectively of size all equal to ℓ).

together with different permutations. Thus, similarly to the standard CNN architecture, features depending on more than one dimension will be detectable while still having a multivariate CAM. Nevertheless, the latter has to be processed such that the significant subsequences are detected.

We first describe the proposed architecture dCNN that we need in order to provide a dimension-wise Class Activation Map (dCAM), while still being able to extract multivariate features. We then demonstrate that the transformation needed to change CNN to dCNN can also be applied to other, more sophisticated architectures, such as ResNet and InceptionTime, which we denote as dResNet and dInceptionTime. We demonstrate that using permutations of the input dimensions makes the classification more robust when important features are localized into small subsequences within some specific dimensions.

We then present in detail how we compute dCAM (based on a dCNN/dResNet/dInceptionTime architecture). Our solution benefits from the permutations injected into the dCNN to identify the most discriminant subsequences used for the classification decision.

5.3.1 Dimension-wise Architecture

As mentioned earlier, the classical CNN architecture mixes all dimensions in the first convolutional layer. Thus, CAM is a univariate data series and does not provide any information on which dimension is the discriminant one for the classification. To address this issue, we can use a two-dimensional CNN architecture by re-organizing the input (i.e., the cCNN solution we described earlier). In this architecture, one kernel (of size $(1, \ell, 1)$) slides on each dimension independently. Thus, for a given D-dimensional data series $(T^{(0)}, ..., T^{(D)})$ of length n, the convolutional layers returns an array of three dimensions (n_f, D, n), each

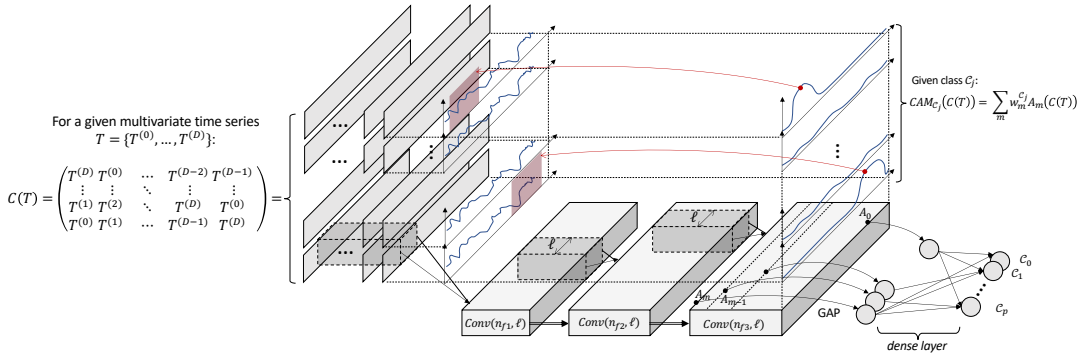


Figure 5.2: dCNN architecture and application of the CAM.

row $m \in [0, D]$ corresponding to the extracted features on dimension m. Nevertheless, the kernels $(1, \ell, 1)$ get as input for each dimension independently. Evidently, such an architecture cannot learn features that depend on multiple dimensions.

5.3.1.1 dCNN: a first architecture

In order to have the best of both cases, we propose the dCNN architecture, where we transform the input into a cube, in which each row contains a given combination of all dimensions. One kernel (of size $(D, \ell, 1)$) slides on all dimensions D times. The latter allows the architecture to learn features on multiple dimensions simultaneously. Moreover, the resulting CAM is a multivariate data series. In this case, one row of the CAM corresponds to a given combination of the dimensions. However, we still need to be able to retrieve information for each dimension separately, as well. To do that, we make sure that each row contains a different permutation of the dimensions. As the weights of the kernels are at fixed positions (for specific dimensions), a permutation of the dimensions will result in a different CAM. Formally, for a given data series T, we note $C(T) \in \mathbb{R}^{(D,D,n)}$ the input data structure of dCNN, defined as follows:

$$C(T) = \begin{pmatrix} T^{(D)} & T^{(0)} & ... & T^{(D-2)} & T^{(D-1)} \\ : & : & ... & : & : \\ T^{(1)} & T^{(2)} & ... & T^{(D)} & T^{(0)} \\ T^{(0)} & T^{(1)} & ... & T^{(D-1)} & T^{(D)} \end{pmatrix}$$

Note that each row and column of $C(T)$ contain all dimensions. Thus, a given dimension $T^{(i)}$ is never at the same position in $C(T)$ rows. It is a crucial property for the computation of dCAM. In practice, we guarantee the latter property by shifting by one position the order of the dimensions. Thus $T^{(0)}$ in the first row is aligned with $T^{(1)}$ in the second row. A different order of T dimensions will thus generate a different matrix $C(T)$.

Figure 5.2 depicts the dCNN architecture. The input $C(T)$ is forwarded into a classical two-dimensional CNN. The rest of the architecture is independent of the input data structure. The latter means that any other two-dimensional architecture (containing a Global

Average Pooling (GAP)) can be used by only adapting the input data structure. Similarly, the training procedure can be freely chosen by the user. For the remainder of the chapter, we will use the cross-entropy loss function and the ADAM optimizer (mentioned in Section 2.6.2.4).

Observe that multiple permutations of the original multivariate series will be processed by several convolutional layers, enabling the kernel to examine multiple different combinations of dimensions and subsequences. Note that the kernels of the dCNN will be sparse, which has a significant impact on overfitting.

5.3.1.2 Other architectures

As mentioned earlier, any architecture using a GAP layer after the last convolutional layer can benefit from dCAM. Thus, different (and more sophisticated) architectures can be used with our approach. To that effect, we propose two new architectures dResNet and dInceptionTime, based on the state-of-the-art architectures ResNet [114] and InceptionTime [52]. The transformations that lead to dResNet and dInceptionTime are very similar to that from CNN to dCNN, using $C(T)$ as input to the transformed networks. The convolutional layers are transformed from 1D (as proposed in the original architecture [114, 52]) to 2D. Similarly, for dCNN, the kernel sizes are $(D, \ell, 1)$ and convolute over each row of $C(T)$ independently.

We demonstrate in the experimental section that these architectures do not affect the usage of our proposed approach dCAM, and we evaluate the choice of architecture on both classification and discriminant feature identification. In the following sections, we describe our methods assuming the dCNN architecture. Nevertheless, it works exactly the same for the other two architectures.

5.3.2 Limitation of CAM for Dimension-wise Architecture

At this point, we have our network trained to classify instances among classes $\mathcal{C}_0, \mathcal{C}_1, ..., \mathcal{C}_p$. We now describe in detail how to compute dCAM that will identify discriminant features within dimensions. We assume that the network has to be accurate enough in order to provide a meaningful discriminant features identification. We evaluate in the experimental section the relation between the classification accuracy of the network and the discriminant features identification accuracy.

At first glance, one can compute the regular Class Activation Map $CAM_{\mathcal{C}_j}(C(T)) = \sum_m w_m^{\mathcal{C}_j} A_m(C(T))$. However, a high value on the i^{th} row at position t on $CAM_{\mathcal{C}_j}(C(T))$ does not mean that the subsequence at position t on the i^{th} dimension is important for the classification. It rather means that the combination of dimensions at the i^{th} row of $C(T)$ is important. Thus, one cannot use CAM as for cCNN. We thus describe how to compute dCAM, which corresponds to the importance of one dimension only at each row.

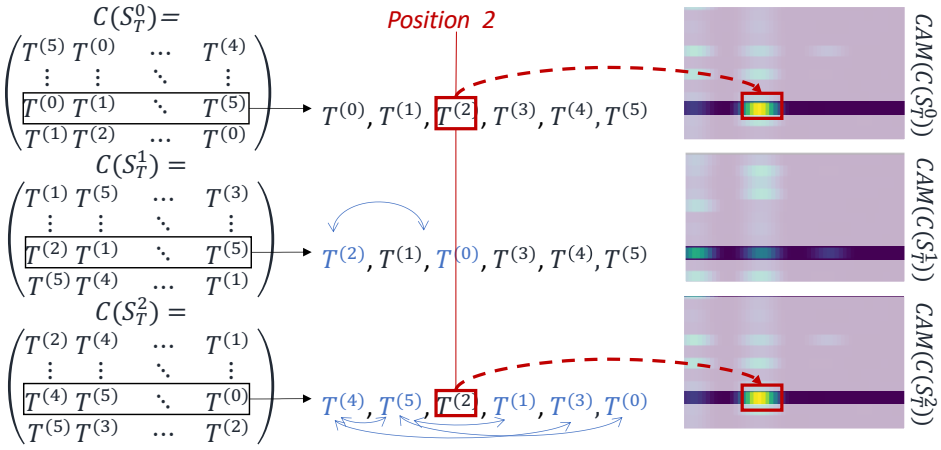


Figure 5.3: Example of Class Activation Map results for different permutations.

5.3.3 Computation of dCAM

We now describe in detail to computation steps involved to produce dCAM from a network dCNN.

5.3.3.1 Permutations of the Dimensions

Given those different combinations of dimensions (i.e., one row of $C(T)$) produce different outputs (i.e., the same row in $CAM_{\mathcal{C}_j}(C(T))$), the positions of the dimensions within the $C(T)$ rows have an impact on the Class Activation Map. Consequently, for a given combination of dimensions, we can assume that at least one dimension at a given position is responsible for the high value in the Class Activation Map row. For the remainder of the chapter, we use Σ_T for the set of all possible permutations of T dimensions, and $S_T^i \in \Sigma_T$ for a single permutation of T. For instance, for a given data series $T = \{T^{(0)}, T^{(1)}, T^{(2)}\}$, one possible permutation is $S_T^i = \{T^{(1)}, T^{(0)}, T^{(2)}\}$.

Figure 5.3 depicts an example of Class Activation Maps for different permutations. In this Figure, for three given permutations of T (i.e., S_T^0, S_T^1 and S_T^2), we notice that when $T^{(2)}$ is in position two of the second row of $C(S_T^i)$, the Class Activation Map $CAM(C(S_T^i))$ is greater than when $T^{(2)}$ is not in position two. We infer that the second dimension of T in position two is responsible for the high value. Thus, we may examine different dimension combinations by keeping track of which dimension at which position is activating the Class Activation Map the most. In the remainder of this section, we describe the steps necessary to retrieve this information.

Definition 26. *For a given data series $T = \{T^{(0)}, T^{(1)}, ..., T^{(D)}\}$ of length n and its input data structure $C(T)$, we define function idx, such that $idx(T^{(i)}, p_j)$ returns the row indices in $C(T)$ that contain the dimension $T^{(i)}$ at position p_j.*

We can now define the following transformation $\mathcal{M}$.

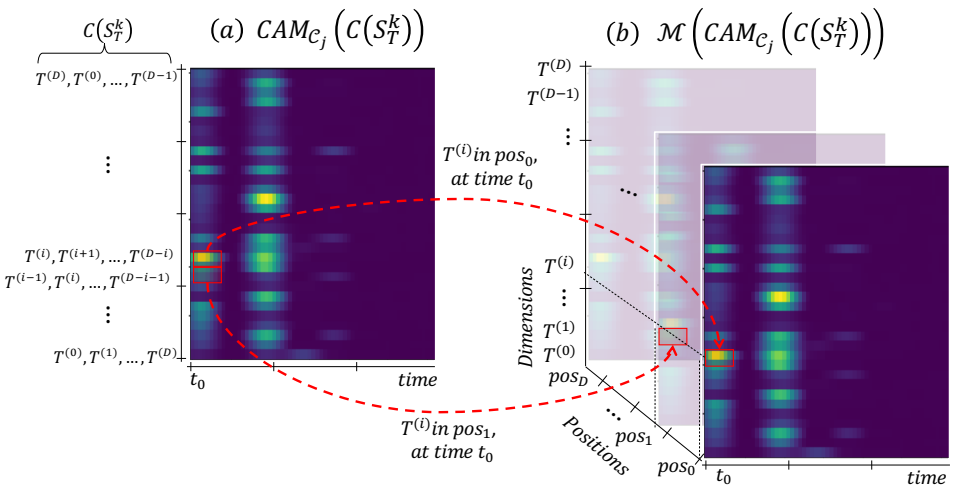


Figure 5.4: Transformation $\mathcal{M}$ for a given data series T.

Definition 27. *For a given data series* $T = \{T^{(0)}, T^{(1)}, ..., T^{(D)}\}$ *of length* n*, a given class* $\mathcal{C}_j$ *and Class Activation Map, we define* $\mathcal{M}(CAM_{\mathcal{C}_j}(C(T))) \in \mathbb{R}^{(D,D,n)}$ *(with* $CAM_{\mathcal{C}_j}(C(T)) \in \mathbb{R}^{(D,n)}$ *and* $CAM_{\mathcal{C}_j}(C(T))_i$ *its* i^{th} *row) as follows:*

$$\mathcal{M}(CAM_{\mathcal{C}_j}(C(T))) = \begin{pmatrix} CAM_{\mathcal{C}_j}(C(T))_{idx(T^{(0)},0)} & \ldots & CAM_{\mathcal{C}_j}(C(T))_{idx(T^{(0)},D)} \\ CAM_{\mathcal{C}_j}(C(T))_{idx(T^{(1)},0)} & \ldots & CAM_{\mathcal{C}_j}(C(T))_{idx(T^{(1)},D)} \\ \vdots & \ldots & \vdots \\ CAM_{\mathcal{C}_j}(C(T))_{idx(T^{(D)},0)} & \ldots & CAM_{\mathcal{C}_j}(C(T))_{idx(T^{(D)},D)} \end{pmatrix} \quad (5.1)$$

Figure 5.4 depicts the $\mathcal{M}$ transformation. As explained in Definition 27, the $\mathcal{M}$ transformation enriches the Class Activation Map by adding the dimension position information. Note that if we change the dimension order of T, their $\mathcal{M}(CAM_{\mathcal{C}_j}(C(T)))$ changes as well. Indeed, for a given dimension $T^{(i)}$ and position p_j, $idx(T^{(i)}, p_j)$ will not have the same value for two different dimension orders of T. Thus, computing $\mathcal{M}(CAM_{\mathcal{C}_j}(C(T)))$ for different dimension orders of T will provide distinct information regarding the importance of a given position (subsequence) in a given dimension. We expect that subsequences (of a specific dimension) that discriminate one class from another will also be associated (most of the time) with a high value in the Class Activation Map.

5.3.3.2 Merging CAM on a set of permutations

We compute $\mathcal{M}(CAM_{\mathcal{C}_j}(C(S_T)))$ for different $S_T \in \Sigma_T$. Note that the total number of permutations $|\Sigma_T|$ is enormous for high-dimensional data series. In practice, we only compute $\mathcal{M}$ for a randomly selected subset of Σ_T. We thus merge $k = |\Sigma_T|$ permutations S_T^k, by

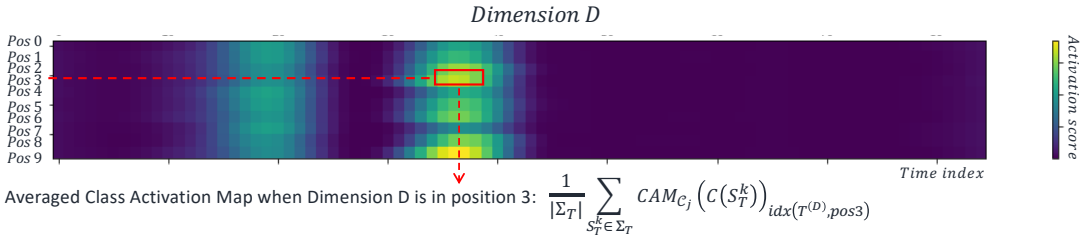


Figure 5.5: Example of one row of matrix $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ for a specific dimension D.

computing the averaged matrix $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ of all the $\mathcal{M}$ transformations of the permutations. Formally, $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ is defined as follows:

$$\bar{\mathcal{M}}_{\mathcal{C}_j}(T) = \frac{1}{|\Sigma_T|} \sum_{S_T^k \in \Sigma_T} \mathcal{M}(CAM_{\mathcal{C}_j}(C(S_T^k)))$$

Figure 5.6 illustrates the process of computing $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ from the set of permutations of T, Σ_T. $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$ can be seen as a summarization of the importance of each dimension at each position in $C(T)$, for all the computed permutations. Note that all permutations of T are forwarded into the dCNN network without training it again. Thus, even though the permutations of T generate radically different inputs to the network, the network can still classify most of the instances correctly. For k permutations, we use n_g to denote the number of permutations that the model has correctly classified. We provide in the experimental section an analysis of n_g/k w.r.t the classification accuracy of the model and the impact that n_g/k has on the discriminant features identification accuracy.

Figure 5.5 illustrates one row of matrix $\bar{\mathcal{M}}_{\mathcal{C}_j}(T)$. In the illustrated heatmap, each row is associated to a position and corresponds to the averaged Class Activation Map when dimension D is in that position. We first observe that the averaged activation is not constant for all time indexes. We also observe that the averaged activation is not constant for all positions. In the following section, we use the latter information to retrieve the discriminant dimensions.

5.3.3.3 dCAM Extraction

We can now use the previously computed $\bar{\mathcal{M}}_{\mathcal{C}_j}$ to extract explanatory information on which subsequences are considered important by the network. First, we note that each row of $C(T)$ corresponds to the input format of the standard CNN architecture. Thus, we expect that the result of a row of $\bar{\mathcal{M}}_{\mathcal{C}_j}$ (one of the ten lines in Figure 5.6(b)) is similar to the standard CAM. Hence, we can assume that $\mu(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)) = \sum_{d \in D} \sum_{p \in D} \bar{\mathcal{M}}_{\mathcal{C}_j}^{d,p}(T)/(2 * |D|)$ is equivalent to standard Class Activation Map $CAM_{\mathcal{C}_j}(T)$ (this approximation is depicted in Figure 5.6(d)).

Moreover, in addition to the temporal information, we can extract temporal information per dimension. We know that for a given position p and a given dimension d, $\bar{\mathcal{M}}_{\mathcal{C}_j}^{d,p}(T)$ represents the averaged activation for a given set of permutations. If the activation $\bar{\mathcal{M}}_{\mathcal{C}_j}^{d,p}(T)$

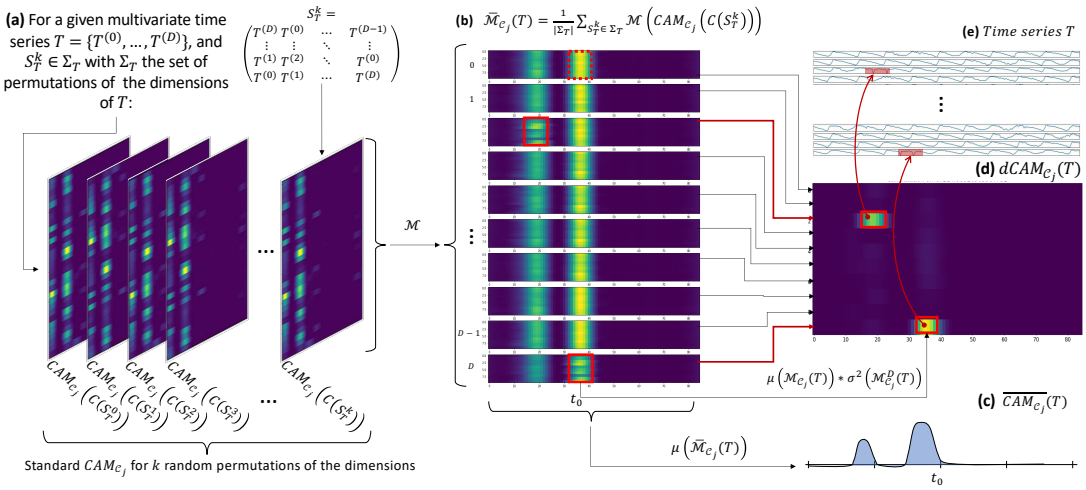


Figure 5.6: dCAM computation framework.

for a given dimension is constant (regardless of its value, or the position p), then the position of dimension d is not important, and no subsequence in that dimension d is discriminant. On the other hand, a high or low value at a specific position p means that the subsequence at this specific position is discriminant. While it is intuitive to interpret a high value, interpreting a low value is counter-intuitive. Usually, a subsequence at position p with a low value should be regarded as non-discriminant. Nevertheless, if the activation is low for p and high for other positions, then the subsequence at position p is the consequence of the low value and is thus discriminant. We experimentally observe this situation, where a non-discriminant dimension has a constant activation per position (e.g., see dotted red rectangle in Figure 5.6(b): this pattern corresponds to a non-discriminant subsequence of the dataset). On the contrary, for discriminant dimensions, we observe a strong variance for the activation per position: either high values or low values (e.g., see solid red rectangles in Figure 5.6(b): these patterns correspond to the (injected) discriminant subsequences highlighted in red in Figure 5.6(e)). We thus can extract the significant subsequences per dimension by computing the variance of all positions of a given dimension. We can filter out the irrelevant temporal windows using the averaged $\mu(\bar{\mathcal{M}}_{\mathcal{C}_j}(T))$ for all dimensions, and use the variance to identify the important dimensions in the relevant temporal windows. Formally, we define $dCAM_{\mathcal{C}_j}(T)$ as follows.

Definition 28. *For a given data series T and a given class $\mathcal{C}_i$, $dCAM_{\mathcal{C}_j}(T)$ is defined as:*

$$dCAM_{\mathcal{C}_j}(T) = \begin{pmatrix} \sigma^2(\bar{\mathcal{M}}^{d_0}_{\mathcal{C}_j}(T)_{t_0}) * \mu(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_0}) & \ldots & \sigma^2(\bar{\mathcal{M}}^{d_0}_{\mathcal{C}_j}(T)_{t_n}) * \mu(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_n}) \\ \vdots & \ldots & \vdots \\ \sigma^2(\bar{\mathcal{M}}^{d_{D-1}}_{\mathcal{C}_j}(T)_{t_0}) * \mu(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_0}) & \ldots & \sigma^2(\bar{\mathcal{M}}^{d_{D-1}}_{\mathcal{C}_j}(T)_{t_n}) * \mu(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_n}) \\ \sigma^2(\bar{\mathcal{M}}^{d_D}_{\mathcal{C}_j}(T)_{t_0}) * \mu(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_0}) & \ldots & \sigma^2(\bar{\mathcal{M}}^{d_D}_{\mathcal{C}_j}(T)_{t_n}) * \mu(\bar{\mathcal{M}}_{\mathcal{C}_j}(T)_{t_n}) \end{pmatrix} \quad (5.2)$$

5.4 Experimental Evaluation

We now present the results of the experimental evaluation with several real datasets from different domains. Overall, the experimental section is organized as follows:

1. We first introduce in Section 5.4.1 the datasets, the baselines, and the evaluation metrics used to measure both classification accuracy, discriminant feature identification accuracy, and execution time.
2. We evaluate in Section 5.4.3 our proposed approaches and the baselines on classification accuracy on 28 publicly available multivariate data series datasets.
3. We evaluate in Section 5.4.4 our proposed approaches and the baselines on discriminant feature identification accuracy on 20 synthetic multivariate data series datasets.
4. We analyze in Section 5.4.5 the relationship between classification accuracy and the discriminant feature identification accuracy.
5. We report scalability experiments in Section 5.4.6, where we measure the execution time needed by the baselines and our approaches for training and for identifying discriminant features.

5.4.1 Experimental Setup

We implemented our algorithms in Python 3.5 using the PyTorch library [94]. The evaluation was conducted on a server with Intel Core i7-8750H CPU 2.20GHz x 12, with 31.3GB RAM, and Quadro P1000/PCIe/SSE2 GPU with 4.2GB RAM, and on Jean Zay cluster with Nvidia Tesla V100 SXM2 GPU with 32 GB RAM.

5.4.1.1 Datasets

We conduct our experimental evaluation using real datasets from the UCR/UEA archive [30] to evaluate the classification performance of our approaches compared to the state-of-the-art methods.

To evaluate the discriminant features identification, we conduct our experimental evaluation using real datasets injected with known discriminant patterns and a real use case from the medical domain. We use the StarLightCurves (classes 2 and 3 only), ShapesAll (classes 1 and 2 only), and Fish (class 1 and 2 only) datasets from the UCR archive [30], in which we inject subsequences that will generate discriminant features. We build two types of datasets to study the ability of the algorithms to identify the discriminant patterns guiding the classification decision, (1) when these patterns occur in a subset of the dimensions at different timestamps, and (2) when these patterns occur in a subset of the dimensions at the same timestamp.

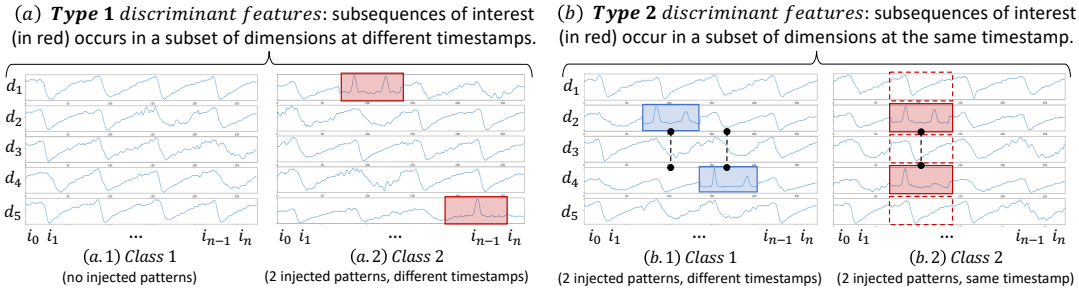


Figure 5.7: Synthetic datasets: (a) $Type\ 1$, in which the discriminant subsequence is two injected patterns from class 2 StarLightCurves dataset in random dimensions at random positions, (b) $Type\ 2$, in which the discriminant factor is the fact that the two injected patterns are injected at the same position.

1. For the $Type\ 1$ datasets, we build each dimension of Class 1 by concatenating random instances from one class of one of our two UCR seed datasets. We build Class 2 by injecting in the data series of the other class of our two UCR datasets a pattern in 2 random dimensions at a random position in the series.

2. For the $Type\ 2$ datasets, we build each dimension of Class 1 by concatenating random instances from one of the classes of our two UCR datasets and injecting patterns from the other class in x random dimensions and at different positions. We build Class 2 by injecting patterns at the same positions of 2 random dimensions.

Examples of $Type\ 1$ and $Type\ 2$ 5-dimensional datasets based on StarLightCurves are depicted in Figures 5.7(a), and 5.7(b), respectively. In our experiments, we use 1000 such datasets.

5.4.1.2 Evaluation Measures

We first evaluate the classification accuracy, $C\text{-}acc$. This measure corresponds to the ratio of correctly classified instances among all instances in the test dataset.

We then evaluate the discriminant features accuracy, $Dr\text{-}acc$, for Class 1 (as depicted in Figure 5.7). We define $Dr\text{-}acc$ as the Area Under the Precision versus Recall Curve (PR-AUC) for CAM/cCAM/dCAM obtained from the models and the ground truth. The ground-truth is a series that has 1 at the positions of discriminant features (the ground-truth of the series depicted in Figure 5.7(a.2) contains 1 at the positions of the injected patterns, marked with the red rectangles, and 0 otherwise). We motivate the choice of PR-AUC (instead of the Area Under the Receiver Operating Characteristic Curve (ROC-AUC)) by the fact that we are more interested in measuring the accuracy of identifying the injected patterns (representing at max 0.02 percent of the dataset) than measuring the accuracy of not detecting the non-injected patterns. In this very unbalanced case, PR-AUC is more appropriate than ROC-AUC [31].

We note that even though we are annotating each point of the injected subsequences as discriminant, only some subparts of these sequences might be discriminant, thus leading to $Dr\text{-}acc$ less than 1. Finally, for CNN/ResNet/InceptionTime we compute the $Dr\text{-}acc$ scores by assuming that their (univariate) CAM values are the same for all dimensions. We mark their $Dr\text{-}acc$ scores with a star in Table 5.2.

5.4.2 Baselines and Training Setup

We compare our model, dCNN,dResNet and dInceptionTime, to the classical CNN, ResNet and InceptionTime model [51, 34, 114, 52], and the cCNN, cResNet and cInceptionTime baseline introduced in Section 5.2. We are using the same architecture setup for all models. We then use CAM for CNN, ResNet, InceptionTime, cCAM for cCNN, cResNet and cInceptionTime, and dCAM for dCNN, dResNet and dInceptionTime to identify discriminant features. For CNN,cCNN and dCNN, we are using 5 convolutional layers with $(64, 128, 256, 256, 256)$ filters respectively. We are using a kernel size of 3 and a padding of 2. The depth of the convolutional-based networks are empirically chosen. For ResNet, cResNet, and dResNet we use three blocks with three convolutional layers of 64 filters (for the first two blocks) and 128 layers (for the last block). For each block, we are using kernel sizes equal to 8, 5, and 3 for three layers of the block. For InceptionTime, cInceptionTime and dInceptionTime, we are using the same architecture as originally defined [52].

We split our dataset into training and validation sets with 80 and 20 percent of the total dataset, respectively (equally balanced between the two classes). The training dataset is used to train the model, and the validation dataset is used as a validation dataset during the training phase. For synthetic datasets, we generate a fully new test dataset and to evaluate $C\text{-}acc$ and $Dr\text{-}acc$. We train all models with a learning rate $\alpha = 0.00001$, a maximum batch size of 16 instances (less if GPU memory cannot fit 16 instances), and a maximal number of epochs equal to 1000 (we use early stopping and stop before 1000 epochs if the model starts overfitting the test dataset). For dCAM, we use $k = 100$ (number of random permutations). We can make the assumption that the number of permutations should be high when the number of dimension is high. Nevertheless, $k = 100$ is a value that we empirically verified to be sufficient over our use cases and benchmark datasets.

5.4.3 Classification Accuracy evaluation

We first evaluate the classification performance of our proposed approaches (denoted as c-Baselines and d-Baselines in Table 5.2) and the ones of the different baselines (denoted as Baselines in Table 5.2) over the UCR/UEA multivariate data series. We run each method ten times and report the average $C\text{-}acc$.

We observe that ResNet-based architecture performs better than CNN-based and InceptionTime-based architectures. Moreover, we note that, overall, dCNN and dResNet have a better $C\text{-}acc$ than CNN and ResNet, respectively. This observation confirms that our proposed architectures (dResNet, dCNN) do not result in any loss in accuracy; on the contrary, they are slightly more accurate than usual architectures (ResNet, CNN). We no-

Metadata				C-acc ***(averaged on 10 runs)***								
				Baselines			***c-Baselines***			***d-Baselines***		
Datasets name	Nb classes	Data length	Nb dimensions	CNN	ResNet	InceptionTime	cCNN	cResNet	cInceptionTime	dCNN	dResNet	dInceptionTime
AtrialFibrillation	3	640	2	0.41	0.40	0.64	0.56	0.53	0.68	0.49	0.45	0.61
Libras	15	45	2	0.96	0.96	0.82	0.80	0.82	0.65	0.91	0.94	0.66
BasicMotions	4	100	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
RacketSports	4	30	6	0.94	0.99	0.90	0.95	0.98	0.85	0.94	0.98	0.92
Epilepsy	4	206	3	1.00	1.00	0.97	1.00	1.00	1.00	1.00	1.00	0.99
StandWalkJump	3	2500	4	0.70	0.66	0.65	0.83	1.00	0.81	0.95	1.00	0.75
UWaveGestureLibrary	8	315	3	0.88	0.89	0.89	0.76	0.74	0.64	0.84	0.89	0.83
Handwriting	2	152	3	0.83	0.90	0.55	0.42	0.70	0.38	0.76	0.89	0.52
NATOPS	6	51	24	0.99	1.00	0.95	0.86	0.89	0.83	0.97	0.99	0.91
PenDigits	10	8	2	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
FingerMovements	2	50	28	0.70	0.68	0.71	0.57	0.63	0.55	0.72	0.71	0.66
ArticularyWordRecognition	25	144	9	0.99	0.99	0.93	0.82	0.94	0.74	0.98	0.99	0.88
HandMovementDirection	4	400	10	0.44	0.42	0.51	0.34	0.35	0.40	0.45	0.44	0.33
Cricket	12	1197	6	1.00	1.00	0.98	0.94	0.97	0.87	1.00	1.00	0.98
LSST	14	36	6	0.62	0.66	0.4 0	0.56	0.59	0.49	0.62	0.66	0.51
EthanolConcentration	4	1751	3	0.35	0.36	0.34	0.36	0.36	0.36	0.35	0.39	0.34
SelfRegulationSCP1	2	896	6	0.86	0.83	0.87	0.88	0.84	0.88	0.86	0.86	0.88
SelfRegulationSCP2	2	1152	7	0.59	0.58	0.62	0.6 0	0.59	0.59	0.57	0.6 0	0.63
Heartbeat	2	405	61	0.83	0.86	0.83	0.76	0.76	0.76	0.84	0.86	0.83
PhonemeSpectra	39	217	39	0.31	0.37	0.27	0.31	0.33	0.28	0.33	0.40	0.32
EigenWorms	5	17984	6	0.9 0	0.92	0.82	0.71	0.92	0.73	0.92	0.92	0.81
MotorImagery	2	3000	64	0.58	0.57	0.56	0.56	0.57	0.56	0.65	0.68	0.66
FaceDetection	2	62	144	0.57	0.59	0.71	0.55	0.70	0.70	0.57	0.61	0.63
Mean				0.758	0.766	0.735	0.701	0.747	0.684	0.770	**0.793**	0.723
Rank				3.78	3.13	4.82	5.52	4.34	5.86	3.56	**2.04**	5.17

Table 5.2: C-acc averaged accuracy for 10 runs for CNN, ResNet, InceptionTime, cCNN, cResNet, cInceptionTime, dCNN, dResNet, dInceptionTime over UCR/UEA datasets.

tice that dResNet is on averaged one rank higher than ResNet. Similar observations can be made when comparing dCNN and CNN.

Moreover, Table 5.2 confirms that using cCNN baselines (or cResNet and cInceptionTime) implies a drop in classification accuracy. For instance, CNN architecture is 0.05 more accurate than cCNN architecture. Thus, c-Baselines cannot guarantee at least equivalent accuracy. Figure 5.8(a) depicts the comparison between dCNN C-acc (on the y-axis) and CNN/cCNN C-acc (on the x-axis, with CNN illustrated with blue circle and cCNN with red crosses). The dotted line corresponds to cases when both classifiers have the same accuracy. We observe that almost all cCNN C-acc (red crosses) are above the dotted line, showing that dCNN is more accurate for most of the datasets. Similarly, we observe that most of the CNN C-acc (blue circles) are above the dotted lines, which means that dCNN is more accurate than CNN. The same observation can be made when examining Figure 5.8(b), in which dResNet is compared with ResNet and cResNet.

However, the same observation is not true when comparing dInceptionTime with InceptionTime and cInceptionTime. Even though in Figure 5.8(c) most of the red crosses are above the dotted line, indicating that dInceptionTime is most of the time more accurate than cInceptionTime, the blue circles are equally distributed above and under the dotted line. Thus, dInceptionTime is not more accurate than InceptionTime. The results in Table 5.2 also show that the averaged C-acc across all datasets (as well as the averaged rank) is lower for dInceptionTime than for InceptionTime. Nevertheless, the results also show that the performance of dInceptionTime is very close to that of InceptionTime. Thus, transforming the original architecture into one that supports dCAM does not penalize classification performance.

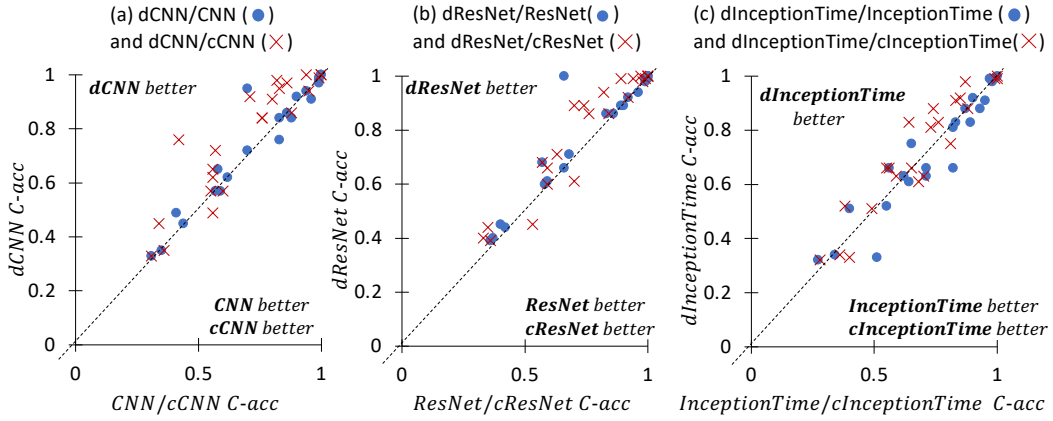


Figure 5.8: C-acc comparison of (a) dCNN with cCNN and CNN, (b) dResNet with cResNet and ResNet, and (c) dInceptionTime with cInceptionTime and InceptionTime on UCR/UEA datasets.

5.4.4 Discriminant Feature identification evaluation

We now evaluate the classification accuracy (C-acc) and the discriminant feature identification accuracy (Dr-acc) on synthetically built datasets. Table 5.3 depicts both C-acc and Dr-acc on $Type$ 1 and 2 datasets, when varying the number of dimensions from 10 to 100. In this experiment, we keep as baselines only ResNet and cResNet, which are the most accurate methods among all other baselines.

We first notice that for low dimensional (D=10) datasets, ResNet, dResNet, dCNN, and dInceptionTime are performing nearly perfect *C-acc*. Moreover, ResNet performs well for low-dimensional data series but starts to fail for a larger number of dimensions. While the drop is already significant for the $Type$ 1 dataset built from the StarLightCurve dataset, it is even stronger for $Type$ 2 datasets, for which ResNet fails to classify instances with a number of dimensions $D \geq 20$. On the contrary, dCNN, dResNet, and dInceptionTime, which use the random permutations in the input, are not sensitive to the number of dimensions and have an almost perfect C-acc for most of $Type$ 1 datasets. We observe a C-acc drop for dCNN, dResNet and dInceptionTime as dimensions increase for $Type$ 2 datasets. However, this drop is significantly less pronounced than that of ResNet. Overall, dCNN, dResNet, and dInceptionTime, which have on average the three highest ranks, are the most accurate methods.

Regarding cResNet, although it achieves a nearly perfect *C-acc* for $Type$ 1 datasets, we observe that it fails to classify correctly instances of $Type$ 2 datasets. As explained in Section 5.2.2, the input data structure is not rich enough to allow comparisons among dimensions, which is the main way to find discriminant features between the two classes of $Type$ 2 datasets. Overall, Figure 5.9(a) shows that dCNN, dResNet and dInceptionTime are equivalent to cResNet for $Type$ 1 (Figure 5.9(a.1)), outperforming all the baselines for $Type$ 2 (Figure 5.9(a.2)), and in general are better than the baselines (ResNet and cResNet) for both types (Figure 5.9(a.3) with $F(Type\ 1, Type\ 2) = \frac{2*Cacc(Type\ 1)*Cacc(Type\ 2)}{Cacc(Type\ 1)+Cacc(Type\ 2)}$).

Datasets			*C-acc* ***(averaged on 10 runs)***					*Dr-acc* ***(averaged on 50 instances)***					
								CAM	**cCAM**	**dCAM**			
Dataset name	Type	Dimensions	ResNet	cResNet	dCNN	dResNet	dInception	ResNet	cResNet	dCNN	dResNet	dInception	Random
StarLightCurve	Type 1	10	0.95	1.00	1.00	1.00	1.00	0.07*	0.92	0.46	0.38	0.21	0.02
		20	0.71	1.00	1.00	1.00	0.98	0.02*	0.92	0.38	0.45	0.36	0.01
		40	0.60	1.00	0.99	1.00	0.93	0.008*	0.94	0.28	0.42	0.39	0.005
		60	0.57	1.00	0.98	0.99	0.91	0.004*	0.92	0.23	0.24	0.13	0.003
		100	0.64	1.00	0.96	0.97	0.79	0.003*	0.92	0.2	0.26	0.10	0.002
	Type 2	10	0.71	0.53	1.00	1.00	0.93	0.0256*	0.025	0.26	0.43	0.10	0.021
		20	0.61	0.55	0.98	1.00	0.70	0.016*	0.01	0.28	0.43	0.05	0.01
		40	0.58	0.51	0.88	0.58	0.56	0.0068*	0.006	0.20	0.05	0.03	0.005
		60	0.55	0.53	0.64	0.59	0.55	0.0058*	0.005	0.01	0.003	0.009	0.003
		100	0.59	0.5	0.59	0.56	0.60	0.0024*	0.002	0.003	0.004	0.02	0.002
ShapesAll	Type 1	10	1.00	1.00	1.00	1.00	1.00	0.09*	0.79	0.66	0.7	0.55	0.02
		20	0.86	1.00	1.00	1.00	0.99	0.03*	0.74	0.56	0.66	0.51	0.011
		40	0.65	1.00	1.00	1.00	1.00	0.008*	0.88	0.45	0.74	0.76	0.005
		60	0.65	1.00	1.00	1.00	0.96	0.005*	0.65	0.44	0.72	0.79	0.003
		100	0.57	1.00	0.98	1.00	0.85	0.003*	0.83	0.31	0.49	0.48	0.002
	Type 2	10	0.82	0.54	1.00	1.00	0.93	0.0467*	0.04	0.63	0.50	0.32	0.02
		20	0.57	0.52	1.00	1.00	0.89	0.0132*	0.013	0.50	0.73	0.40	0.01
		40	0.60	0.52	0.90	0.72	0.73	0.005*	0.005	0.40	0.20	0.36	0.005
		60	0.59	0.51	0.65	0.61	0.72	0.0037*	0.003	0.22	0.34	0.46	0.003
		100	0.59	0.50	0.55	0.58	0.55	0.0027*	0.002	0.005	0.02	0.05	0.002
rank			3.9	3	1.65	**1.6**	2.85	4.45	3	2.6	**2.15**	2.75	

Table 5.3: C-acc and Dr-acc averaged accuracy for 10 runs for ResNet, cResNet, dCNN, dResNet, dInceptionTime over synthetic datasets.

We now compare the different methods using the Dr-acc measure. We observe that the baseline cCAM (computed with cCNN) is outperforming CAM (computed with ResNet) and dCAM (all of dCNN, dResNet and dInceptionTime) for $Type\ 1$ datasets. The latter is explained by the fact that such dataset classes can be discriminated by looking at each dimension independently. Thus, cCAM is naturally the best solution. Nevertheless, as $Type\ 2$ datasets require comparisons among dimensions to discriminate the classes, cCAM fails on them, with a Dr-acc very similar to the one of a random classifier. It confirms that such a baseline cannot be considered as a general solution for multivariate data series classification. We then compare CAM and dCAM (used with dCNN, dResNet and dInceptionTime). We note that dCAM significantly outperforms CAM. As depicted in Figure 5.9(b), we also observe that Dr-acc reduces for all models as the number of dimensions increases. Nevertheless, Dr-acc of dCAM remains relatively high for both the $Type\ 1$ (Figure 5.9(b.1)) and $Type\ 2$ (Figure 5.9(b.2)) datasets (for a number of dimensions under 60). The previous result demonstrates the superiority of dCAM over state-of-the-art methods. The superiority of dCAM is also confirmed by looking at the average ranks in Table 5.3, which indicates that dCAM computed from ResNet has the highest rank of 2.15.

5.4.5 C-acc versus Dr-acc

In this section, we conduct three different analyses. We first analyze the relation between C-acc and Dr-acc. We then evaluate the impact that C-acc has on the number of permutations that have been correctly classified n_g. We finally evaluate the impact that n_g has on Dr-acc. We perform these three analyses for dCNN, dResNet and dInceptionTime.

First, Figure 5.10(1) depicts the relation between C-acc and Dr-acc for dCNN (Figure 5.10(a.1)), dResNet (Figure 5.10(b.1)) and dInceptionTime (Figure 5.10(c.1)) for all the

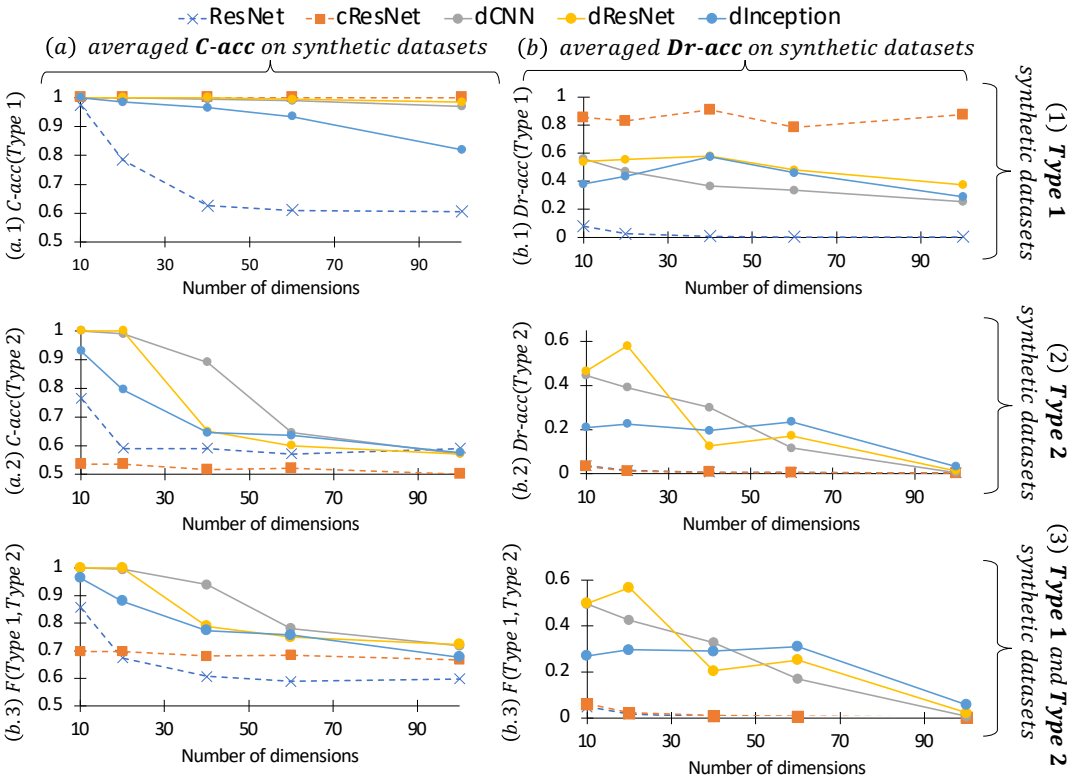


Figure 5.9: Evaluation of the influence of the number of dimensions on our approaches and the baselines C-acc and Dr-acc.

synthetic datasets. We notice that all methods have a logarithmic relation (dotted red line) between Dr-acc (on the x-axis) and C-acc (on the y-axis). It confirms that the accuracy of the trained model has a significant impact on discriminant feature identification. With the lack of blue dot on the bottom right corners of Figure 5.10(a.1),(a.2), and (a.3), we can conclude that an inaccurate model will most probably lead to an inaccurate discriminant features identification.

Figure 5.10(3) depicts on the y-axis the ratio of correctly classified permutations (n_g) among all permutations (k) versus the C-acc (on the x-axis). In this case, for all of dCNN (Figure 5.10(a.3)), dResNet (Figure 5.10(b.3)) and dInceptionTime (Figure 5.10(c.3)), we observe that there exists a linear relationship between n_g/k and C-acc from 0.7 to 1. It means that n_g will be greater when the model is more accurate. Nevertheless, for C-acc between 0.5 and 0.7, we observe a high variance for n_g/k. Thus, an inaccurate model may still lead to a high n_g.

Finally, Figure 5.10(2) depicts the relation between n_g/k (on the y-axis) and Dr-acc (on the x-axis). We observe a similar relationship between C-acc and Dr-acc, which means that a low n_g may lead to inaccurate discriminant features identification.

To conclude, as was mentioned in Section 5.3.1, the experimental results confirm that an inaccurate model (for all of dCNN, dResNet and dInceptionTime) cannot be used to identify

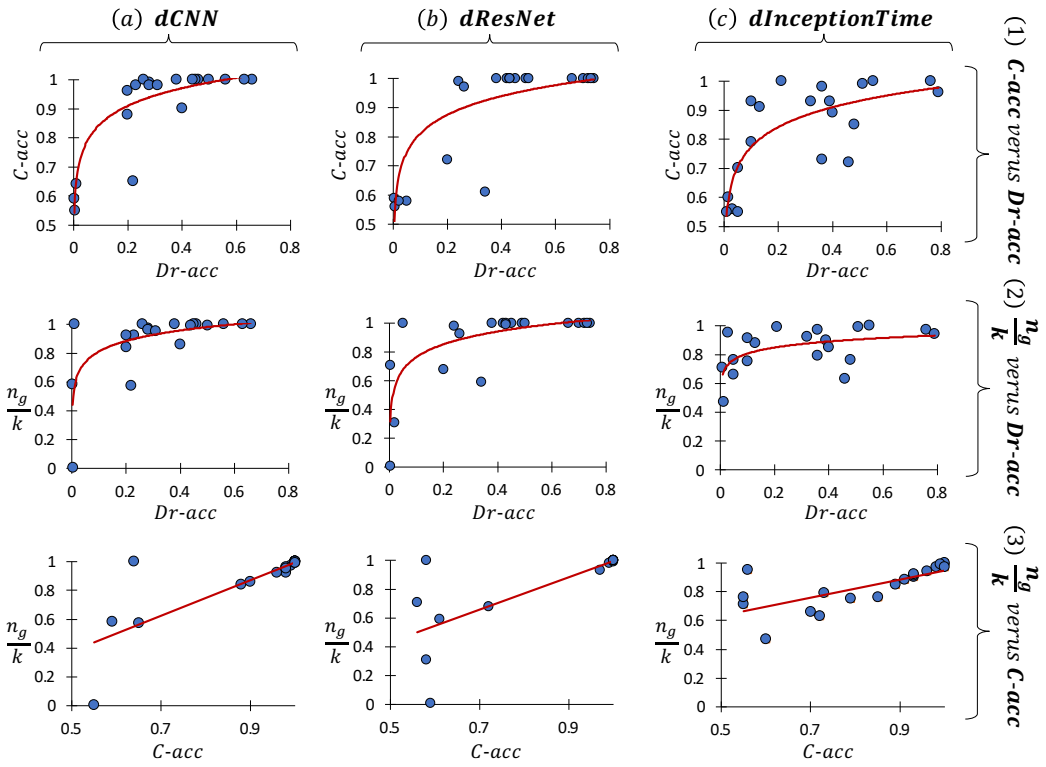


Figure 5.10: Evaluation of $C\text{-}acc$, $Dr\text{-}acc$ and the ratio between the number of permutations k and the number of permutations correctly classified n_g for dCNN, dResNet and dInceptionTime.

discriminant features. Moreover, as it is not possible in practice to measure $Dr\text{-}acc$ for a real use case, we can use n_g/k to estimate the discriminant feature identification accuracy. Even though Figure 5.10(2) demonstrates that a high n_g/k does not always lead to a high $Dr\text{-}acc$, in practice, we can safely assume that a low n_g/k will most probably correspond to a low $Dr\text{-}acc$.

5.4.6 Execution time evaluation

In this section, we evaluate the execution time of our proposed approaches and the baselines. Figure 5.11(a) depicts the training execution time (for one epoch) when we vary the data series length with a constant number of dimensions fixed to ten (Figure 5.11(a.1)), and when we vary the number of dimensions with a constant data series length fixed to 100 (Figure 5.11(a.2)). We notice that overall, CNN-based and InceptionTime-based architectures are faster than ResNet-based architectures. Moreover, the traditional CNN, ResNet and InceptionTime are faster when the number of dimensions and the data series length is increasing. Nevertheless, both dCNN/dResNet/dInceptionTime and cCNN/cResNet/cInceptionTime (that are the only solutions that can provide a multivariate Class Activation Map) require the same execution time for the training steps.

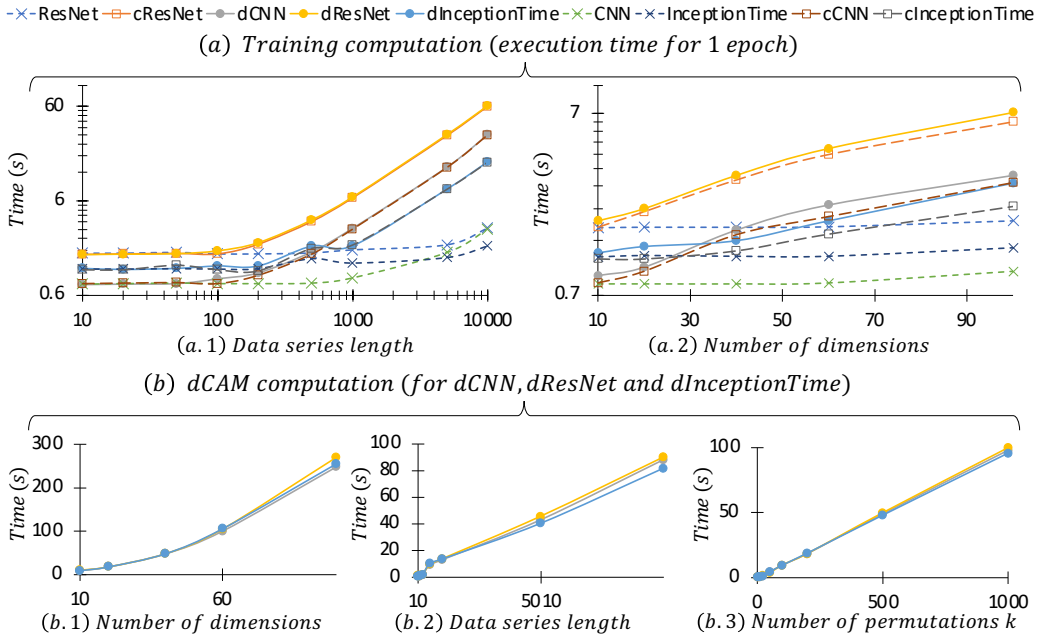
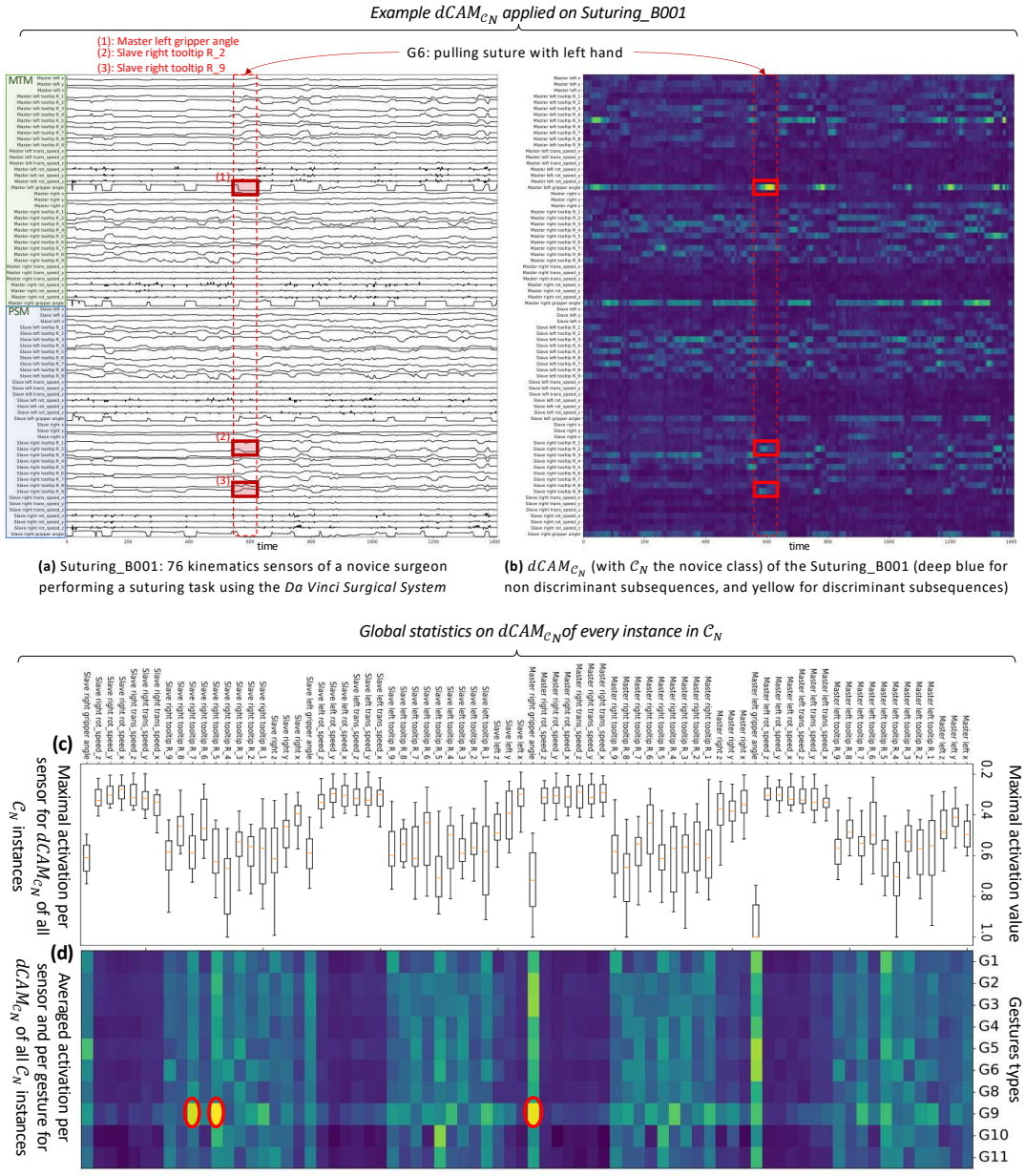


Figure 5.11: Execution time (in second) for the (a) training computations when we vary (a.1) the data series length and (a.2) the number of dimensions. (b) Execution time for dCAM computation (using dCNN, dResNet and dInceptionTime) when we vary (b.1) the number of dimensions, (b.2) the data series length, (b.3) the number of permutations k.

Moreover, we measure the execution time required to compute dCAM (for all of dCNN, dResNet and dInceptionTime) when we vary the number of dimensions with a constant data series length fixed to 400 (Figure 5.11(b.1)), when we vary the data series length with a constant number of dimensions fixed to 10 (Figure 5.11(b.2)), and when we vary the number of permutations (Figure 5.11(b.3)). Note that the dCAM execution times are very similar for the three types of architectures. Moreover, the execution time increases super linearly with the number of dimensions but is linear to the data series length and the number of permutations k.

5.4.7 Use Case: Surgeon skills explanation

We now illustrate the applicability of our method to a real-world use case. In this use case, we train our dCNN network on the JIGSAWS dataset [37] to identify novice surgeons, based on kinematic data series when performing *surgical suturing* tasks (i.e., wound stitching) using robotic arms and surgical grippers.



(a) Suturing_B001: 76 kinematics sensors of a novice surgeon performing a suturing task using the *Da Vinci Surgical System*

(b) $dCAM_{\mathcal{C}_N}$ (with $\mathcal{C}_N$ the novice class) of the Suturing_B001 (deep blue for non discriminant subsequences, and yellow for discriminant subsequences)

Figure 5.12: Example of the result of dCAM on a multivariate data series of the JIGSAWS dataset. (a) the depicted data series corresponds to a novice surgeon performing a suture operation (joined with the corresponding dCAM in (b)). General statistical results over the entire novice class $\mathcal{C}_N$ are depicted such as (c) box-plots of the maximal activation value per sensor and (d) the averaged activation per sensor per gesture performed.

5.4.7.1 Dataset

The data series are retrieved from the $DaVinciSurgicalSystem$. The multivariate data series are composed of 76 dimensions (an example of multivariate data series is depicted in Figure 5.12(a)). Each dimension corresponds to a sensor (with an acquisition rate of 30 Hz). The sensors are divided into four groups: patient-side manipulators (left and right PSMs: green rectangle in Figure 5.12(a) top left), and left and right master tool manipulators (left and right MTMs: blue rectangle in Figure 5.12(a) bottom left). Each group contains 19 sensors. These sensors are: (i) 3 variables for the Cartesian position of the manipulator, (ii) 9 variables for the rotation matrix, (iii) 6 variables for the linear and angular velocity of the manipulator, and (iv) one variable for the gripper angle.

To perform a suture, the surgeons perform different gestures (11 in total). For example, $G1$ refers to reaching for the needle with the right hand, while $G11$ refers to dropping the suture at the end and moving to end points. Each gesture corresponds to a specific time segment of the dataset, involving all sensors. For example, the dotted red rectangle in Figure 5.12(a) represents gesture $G6$: pulling the suture with the left hand. Surgeons that reported having more than 100 hours of experience are considered experts, surgeons with 10-100 hours are considered intermediate, and surgeons with less than 10 hours are labeled as novices. We have 19 multivariate data series in the novice class, denoted as $\mathcal{C}_N$, 10 in the intermediate, $\mathcal{C}_i$, 10 multivariate data series in the expert class, $\mathcal{C}_E$. More information on this dataset can be found in [37].

5.4.7.2 Training

For the training procedure, we use 80% of the dataset (randomly selected from the three classes) for training. The rest 20% of the dataset is used for validation and early stopping. Since the instances do not have the same length, we use batches composed of one instance when training the models in the GPU.

5.4.7.3 Evaluation

Similar to what has been reported in previous work [51], we achieve 100% accuracy on the train and test datasets (for 10 different randomly selected train and test sets). We proceed to compute the $dCAM_{\mathcal{C}_N}$ for every instance of the novice class $\mathcal{C}_N$. The $dCAM_{\mathcal{C}_N}$ of the multivariate data series named $Suturing\_B001$ (Figure 5.12(a)) is displayed in Figure 5.12(b). In the latter, the deep blue color indicates low activated subsequences (i.e., non-discriminant of belonging to the novice class $\mathcal{C}_N$), while the yellow color is pointing to highly activated subsequences. First, we note that some groups of sensors (dimensions) are more activated than others. In Figure 5.12(b), the left and right "MTM gripper angles" are the most activated sensors. Figure 5.12(c), which depicts the box-plot of the maximal activated values per sensors, confirms that in the general case, MTM gripper angles, as well as the MTM and PSM tooltip rotation matrices (three of these sensors are highlighted in red in Figure 5.12(a)), are the most discriminant sensors. On the contrary, linear and angular speeds are not discriminant and hence cannot explain the novice class $\mathcal{C}_N$.

Figure 5.12(d) depicts the averaged activation per sensor per gesture. Overall, $dCAM_{\mathcal{C}_N}$ identifies gesture $G9$ (using the right hand to help tighten the suture) as a discriminant gesture, because of the discriminant subsequences present in the sensors "right MTM gripper angle", "5^{th} element", and "7^{th} element" (marked with red ovals in Figure 5.12(d)). These three identified sensors (dimensions) are relevant to the right PSM tooltip rotation matrix and are important for the suturing process. Similarly, we observe that gesture $G6$ (i.e., pulling suture with left hand) is discriminant, and activated the most by the "left NTM gripper angle" sensor. This result is consistent with a previous study [51], which also identified gesture $G6$ as a discriminant of belonging to the novice class. Nevertheless, dCAM provides more accurate (and useful) information: it does not only identify the discriminant gesture (that relates to all sensors), but also the discriminant sensors.

5.4.7.4 Results Conclusion

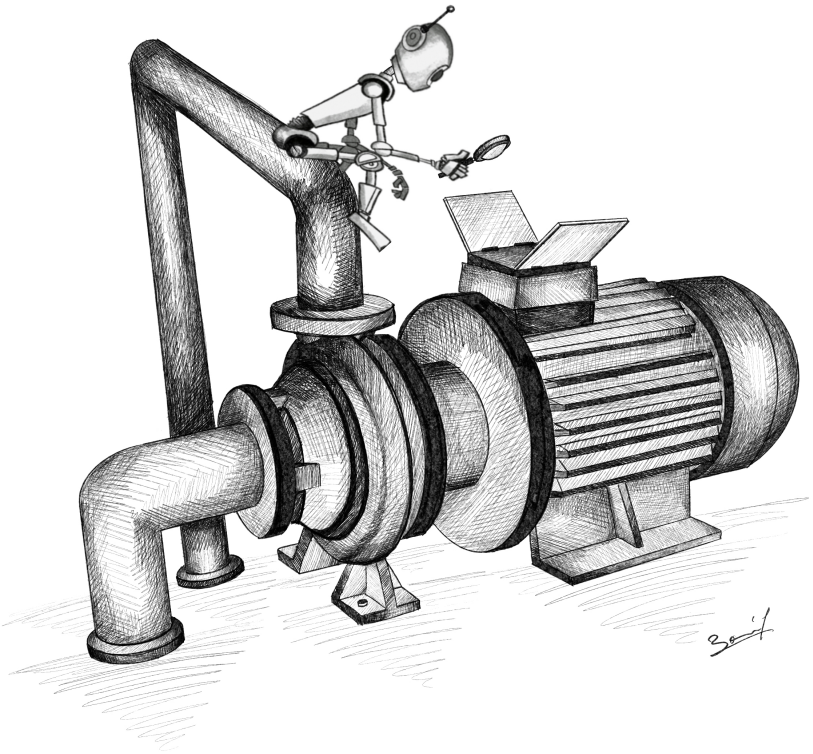
The application of the proposed dCAM approach in the robot-assisted surgeon training use case demonstrated its effectiveness. Our approach was able to provide meaningful explanations for the classification decisions, based on specific gestures (subsequences), and specific sensors (dimensions) that describe particular aspects of these gestures, i.e., the positioning and rotation angles of the tip of the stitch gripper. Such explanations can help surgeons to improve their skills.

5.5 Summary

Even though data series classification using deep learning has attracted a lot of attention, existing techniques for explaining the classification decisions fail in the case of multivariate data series. Moreover, simple extensions, such as cCNN, do not meet usual architectures' performances, such as CNN. In this work, we described a novel approach, dCAM, based on convolutional neural networks, enabling us to detect the discriminant subsequences within individual dimensions of a multivariate data series. In order to apply dCAM, we provided a transformation to the network architecture that we call dCNN (for the usual CNN architecture) dResNet and dInceptionTime (for the ResNet and the InceptionTime architectures). The experimental evaluation with synthetic and real datasets demonstrates our approach's benefits and superiority in discriminant feature discovery and classification explanation in multivariate time series. Moreover, we evaluated the influences between several important parameters in order to give the user all the keys to estimate the accuracy of dCNN and dCAM.

Chapter 6

Use Case: Pump Vibration Case



Chapter Outline and Abstract

6.1 Use Case: Precursors of Anomalous Vibration Detection in Nuclear Power Plants . . . 147
6.1.1 Dataset and Use case description . . . 147
6.1.2 Unsupervised Approaches . . . 149
6.1.3 Supervised Approaches . . . 149
6.2 Experimental Analysis . . . 149
6.2.1 Vibration Detection Evaluation . . . 150
6.2.2 Precursors identification Quantitative Evaluation . . . 158
6.2.3 Precursor identification Qualitative Evaluation . . . 160
6.3 Summary and Conclusion . . . 162

In this chapter, we illustrate the applicability and the interest of our developed methods in a real-world application. We study the detection precursors of unwanted vibrations occurring in turbine-driven feed-water pump systems inside EDF nuclear power plants. We first describe the dataset and the industrial context. We then explore two scenarios: we first tackle the task as if the experts had provided no label and apply NormA and Series2Graph. We evaluate their accuracy using the labels of the experts. We then tackle the task as a supervised problem (using the label provided by the experts) and apply dCNN/dCAM. We first compare the accuracy of both unsupervised and supervised methods. We then investigate the precursors detected by dCAM and discuss the validity of these results compared to expert knowledge.

Symbol	Description
TPA	Turbine-driven feed-water pump
$T^N_{\mathcal{M}}$	Set of multivariate data series without any vibration
$T^A_{\mathcal{M}}$	Set of multivariate data series containing a vibration
APP	Turbine-driven feedwater pump system
AGR	Feedwater pump turbine lubrification and control fluid system
MT	Temperature measure
MD	Flow measure
MP	Pressure measure
MC	Speed measure

Table 6.1: Table of symbols and acronyme related to the use case

6.1 Use Case: Precursors of Anomalous Vibration Detection in Nuclear Power Plants

We now illustrate the applicability and the interest of our developed methods in a real-world application. This use case is about discovering possible precursors of unwanted vibrations happening in turbine-driven feed-water pump systems inside EDF nuclear power plants. These pumps (two different pumps noted TPA1 and TPA2) aim to increase the water pressure (from 1 to 80 bar) before passing the water through the steam generator (with a pressure of 80 bar). However, these vibrations are considered problematic when the position of the pump varies by few microns, and a boolean sensor is activated when it happens. Thus, knowledge experts are interested in finding, if there exist, possible precursors of these unwanted vibrations in other sensors surrounding the pump and discovering unusual patterns that could explain why a pump is eventually vibrating or at least alert the imminent occurrence of vibrations. Table 6.1 summarizes the symbols we use in this chapter.

6.1.1 Dataset and Use case description

At this point, we need to create our datasets of abnormal data series (i.e. vibrations) and anomaly-free data series. Following the suggestion of expert knowledge, we selected 120 sensors inside 12 sub-systems of the nuclear power plant. Figure 6.1(a) summarizes the sub-systems analyzed and the number of sensors collected. The experts observe that TPA are vibrating of few micrometers. They classify these vibrations in two types: (i) Vibrations between 70 and 90 micrometers, (ii) Vibrations above 90 micrometers. In this study, we collected every unwanted vibration of type (ii) that happened in every French 1300MW nuclear plant. In total, we have 444 vibrations. We then create our multivariate data series by selecting every sensor's measurement between 75 minutes before the vibrations and 5 minutes after. We set the sampling rate to 1 point every 6 seconds, and each multivariate data series contains 96,000 points. We note the set of data series containing a vibration $T^A_{\mathcal{M}}$. We then select 444 intervals of 80 minutes for which no vibration has been recorded (i.e., vibrations less than 70 micrometers) at least one day before and after. Finally, we note the set of data series without any vibration $T^N_{\mathcal{M}}$. We also selected the non-vibration periods

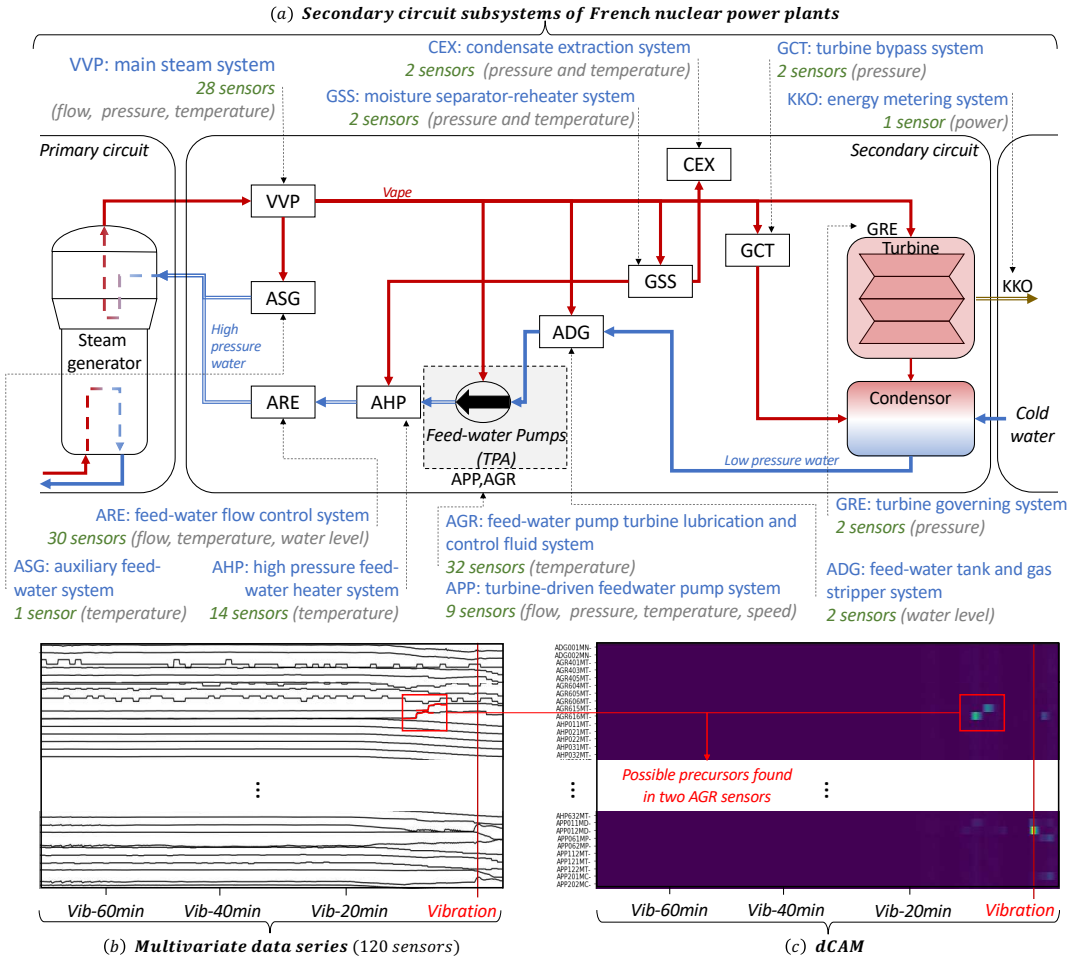


Figure 6.1: Simplified schema of the secondary circuit of EDF 1300MW nuclear power plant. We collect in total 120 sensors from 8 subsystems (solid black boxes) surrounding the feed-water pumps (TPA). Blue arrows: water flow. Red arrows: steam flows.

corresponding to the same operating conditions as the vibration periods. Namely, when the nuclear facility ramps up or down between 15% and 67% of maximum power. This operating area, where the second feed-water pump is coupled, is conducive to vibrations. The latter is a critical step and must be conducted thoroughly. Otherwise, the precursors highlighted would be already-known differences in operational conditions and solicitations. In this case, we used the sensor related to the power regime: its distribution is the same across both classes. We also took the same distribution of years for both classes to minimize the influence of degradation due to aging. What we want to highlight are unexpected solicitations that would have led to vibrations later. The latter will trigger immediate and cost-effective corrective actions. We thus have, in total, 888 multivariate data series (for which 444 of them correspond to unwanted vibrations) of dimensions $D = 129$. All the sensors are returning continuous values. In total the datasets contains 85,248,000 points. Formally, we define the dataset as $T_{\mathcal{M}} = T^N_{\mathcal{M}} \cup T^A_{\mathcal{M}}$, with $T^N_{\mathcal{M}}, T^A_{\mathcal{M}} \in \mathbb{R}^{(444,120,800)}$.

Overall, the task is to detect the vibration correctly and discover subsequences in one or several sensors that happened before the vibration and could potentially explain it. For that purpose, we tackle this task as defined in Problem 1 with NormA and Series2Graph (with subsequence length of 2 minutes), and as defined in Problem 2 with dCNN/dCAM. We have shown in Chapter 5 that dResNet is slightly more accurate than dCNN and dInception Time. However, for the sake of simplicity, we used the most usual architecture dCNN as our baseline for this use case. In the following two sections, we describe the experimental settings of the unsupervised approaches (NormA and Series2Graph) and the supervised approach (dCNN/dCAM). The general objective is to evaluate and compare both unsupervised and supervised methods on a real industrial use case. In addition, we will measure the limit of unsupervised methods and the gain that human labels can bring in consistency.

6.1.2 Unsupervised Approaches

First, we use NormA and Series2Graph score function as $f : T_{\mathcal{M}} \rightarrow \{\mathcal{N}, \mathcal{A}\}$ and $\mathcal{A}$ is the set of subsequences (within each dimension of multivariate data series in $T_{\mathcal{M}}$) with a score higher than a given threshold. In practice, as both NormA and Series2Graph are not dedicated to multivariate data series, we train NormA and Series2Graph models on each dimension independently (across every time series). We set the parameters as their default values described in Section 3.5.5 and Section 3.5.6. We first build a model (N_M for NormA, and the directed graph G for Series2Graph) using all data series of a given dimension. In total, we build one model per dimension. We then compute an anomaly score for each data series (for each dimension). We then use this anomaly score to classify instances as an anomaly or normal. We finally analyze the anomaly score of each dimension independently.

6.1.3 Supervised Approaches

We then train dCNN to classify $T^A_{\mathcal{M}}$ and $T^N_{\mathcal{M}}$. Formally, dCNN is defined as the function $f : T^{\mathcal{N}}_{\mathcal{M}}, T^{\mathcal{A}}_{\mathcal{M}} \rightarrow \{\mathcal{N}, \mathcal{A}\}$. We then use dCAM as the function $g : T^{\mathcal{A}}_{\mathcal{M}}, f \rightarrow S$ that return the set S of subsequences that explain the classification as the vibration class (as the red subsequences and rectangle depicted in Figure 6.1(b,c)). In practice dCAM returns a multivariate data series score for each instance in $T^{\mathcal{A}}_{\mathcal{M}}$ (as depicted in Figure 6.1(b,c)).

6.2 Experimental Analysis

We now evaluate both unsupervised and supervised methods' accuracy on the vibration detection and prediction use case. We conduct two evaluations described as follows: (i) We first evaluate the detection accuracy. We thus, measure the accuracy of the anomaly score of NormA and Series2Graph when used to detect instances (i.e., entire multivariate data series) containing a vibration. We also measure the classification accuracy of dCNN. (ii) We then measure the ability of dCAM (applied on the trained dCNN) to detect possible precursors. We compare first the sensors considered necessary by NormA, Series2Graph,

and dCNN/dCAM and verify if the results are consistent with expert information on the power plant structure. We then evaluate the temporal position of the detected precursors and verify temporal consistency based on expert knowledge. We finally explore the group of subsequences detected by dCAM and analyze them by their shapes. We conclude the analysis by the expert validation of the complete analysis.

6.2.1 Vibration Detection Evaluation

We now evaluate the detection accuracy. We first measure the accuracy of the anomaly scores of NormA and Series2Graph. We then measure the classification accuracy of dCNN. Note that the detection accuracy is not a challenging problem for the experts (with boolean sensors positioned on the pumps that raise an alert if there are any vibrations). However, this is a required step before tackling the precursors' detection. We thus experimentally evaluate the vibration detection as a validation.

6.2.1.1 Unsupervised Detection

We first start by analyzing how unsupervised approaches, such as NormA and Series2Graph, can detect data series with a vibration. As the methods are unsupervised, we build the models on both data series with and without anomalies (without specifying which data series belong to which class). We set the parameters as defined in Section 3.5.5 and Section 3.5.6, and we use a subsequence length of 20 points. As the methods can only be used for univariate data series, we build one model per dimension (i.e., sensor). We then compute an anomaly score data series for each data series instance. We note $Score^d_{NormA}(T^{(d)})$ (with $|Score^d_{NormA}(T^{(d)})| = |T^{(d)}|$) the anomaly score of data series $T^{(d)}$ using the NormA model trained on dimension d (with $0 \leq d < 129$). Similarly, we note $Score^d_{S2G}(T^{(d)})$ (with $|Score^d_{S2G}(T^{(d)})| = |T^{(d)}|$) the anomaly score of $T^{(d)}$ using the Series2Graph model trained on dimension d.

Figure 6.2 depicts six examples of $Score^d_{S2G}(T^{(d)})$ with dimension d corresponding to $APP202MC-$ (speed sensor from the turbine-driven feedwater pump system with two examples illustrated in Figure 6.2(a)), $AGR412MT-$ (temperature sensor from the feedwater pump turbine lubrification and control fluid system with two examples illustrated in Figure 6.2(b)), and $APP062MP-$ (pressure sensor from the turbine-driven feedwater pump system with two examples illustrated in Figure 6.2(c)). In these six examples illustrated in Figure 6.2, we observe that the anomaly score highlights some specific patterns. The latter patterns are either very close to the vibration timestamp represented by the solid vertical red line (such as the top plot of Figure 6.2(a)) or largely earlier (such as the bottom plot of Figure 6.2(a)). Both of these patterns can be relevant to detect vibration instances.

We thus compute the maximal value of $Score^d_{S2G}(T^{(d)})$ and $Score^d_{NormA}(T^{(d)})$ and use it as a unique score for $T^{(d)}$. We thus expect that for two data series $T, T' \in T^N_{\mathcal{M}}, T^A_{\mathcal{M}}$, there exist a dimension d such as $max(Score^d_{NormA}(T^{(d)})) > max(Score^d_{NormA}(T'^{(d)}))$ and $max(Score^d_{S2G}(T^{(d)})) > max(Score^d_{S2G}(T'^{(d)}))$. We now evaluate the last assumption. Figure 6.3 and Figure 6.4 depicts for each sensor d the value distribution (histogram) of the

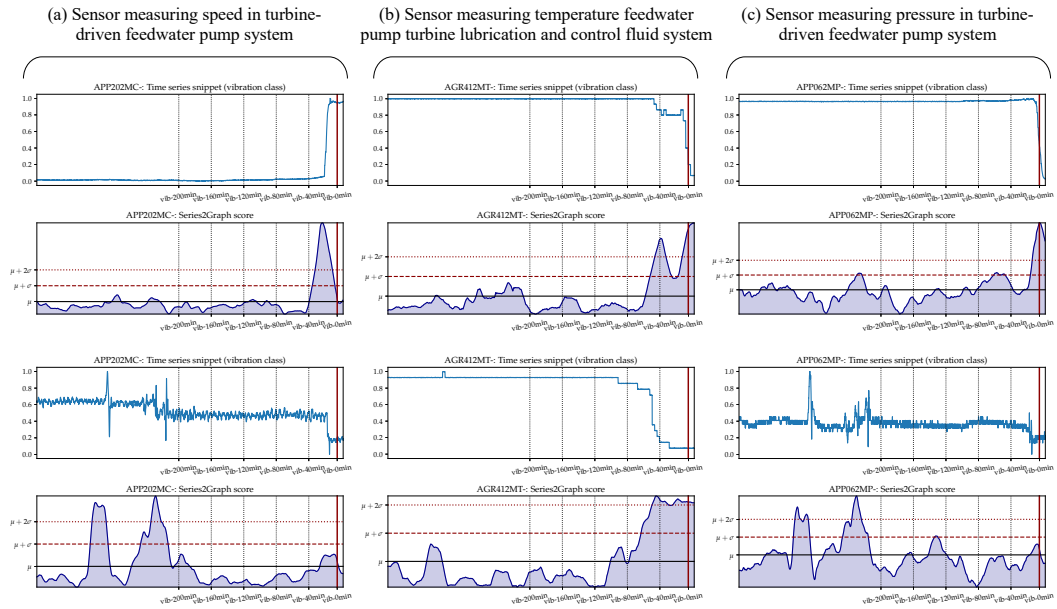


Figure 6.2: Example of anomaly score produce by Series2Graph algorithm for time series containing a vibration from three sensors.

$max(Score^{d}_{NormA}(T^{(d)}))$ and $max(Score^{d}_{S2G}(T^{(d)}))$ respectively. The histograms of each sensor d are sorted by $s(d)$ defined as follow:

For NormA:

$$s(d) = \sum_{bins>0} \left[Hist_{T \in T^{A}_{\mathcal{M}}}\left(max(Score^{d}_{NormA}(T^{(d)}))\right) - Hist_{T \in T^{N}_{\mathcal{M}}}\left(max(Score^{d}_{NormA}(T^{(d)}))\right) \right] \quad (6.1)$$

For Series2Graph:

$$s(d) = \sum_{bins>0} \left[Hist_{T \in T^{A}_{\mathcal{M}}}\left(max(Score^{d}_{S2G}(T^{(d)}))\right) - Hist_{T \in T^{N}_{\mathcal{M}}}\left(max(Score^{d}_{S2G}(T^{(d)}))\right) \right] \quad (6.2)$$

The latter value is high when the score distribution is shifted on the right compared to the blue distribution. Thus, the plots at the bottom of Figure 6.3 and Figure 6.4 are the sensor for which the data series with a vibration obtained a higher anomaly score than the data series without any vibrations. The latter sorting equation is for visual purposes only but provides a first approximate answer on which sensor could be helpful to detect vibrations correctly.

In general, we observe that, for a large number of sensors, the distributions overlap entirely. Thus data series with or without vibrations have a similar anomaly score. For instance, as illustrated in Figure 6.4, we observe that some sensor $APP122MT-$ data series belonging to the vibration class can be discriminated using the Series2Graph anomaly score. Likewise, in Figure 6.3, the same observation can be done for NormA anomaly score on sensor $AGR418MT-$.

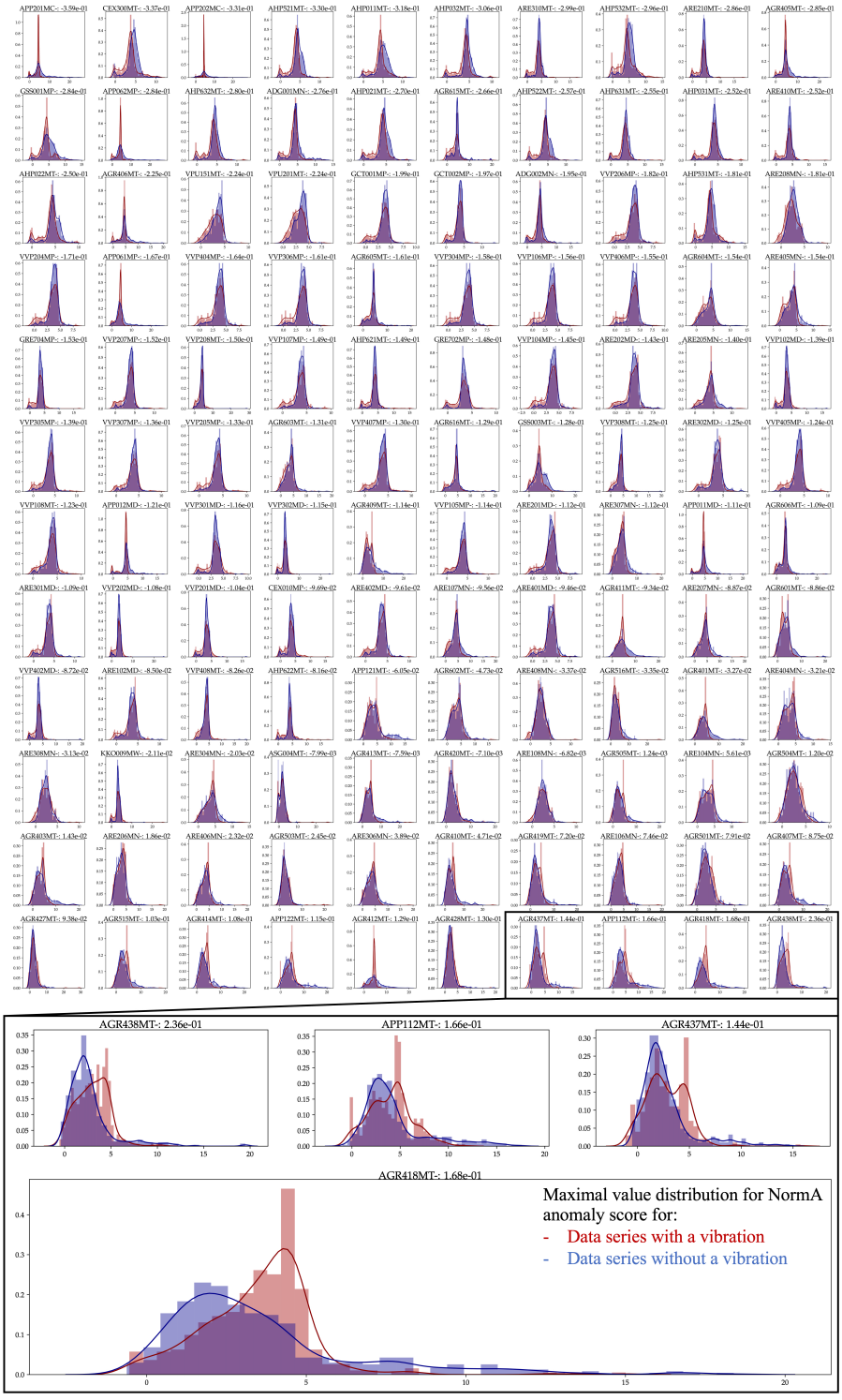


Figure 6.3: For each sensor, maximal NormA anomaly score distribution for time series with a vibration (in red) and without any vibration (in blue). The distribution plots are sorted based on the difference of the red and blue distribution (in that order) as depicted in the title of each subplot.

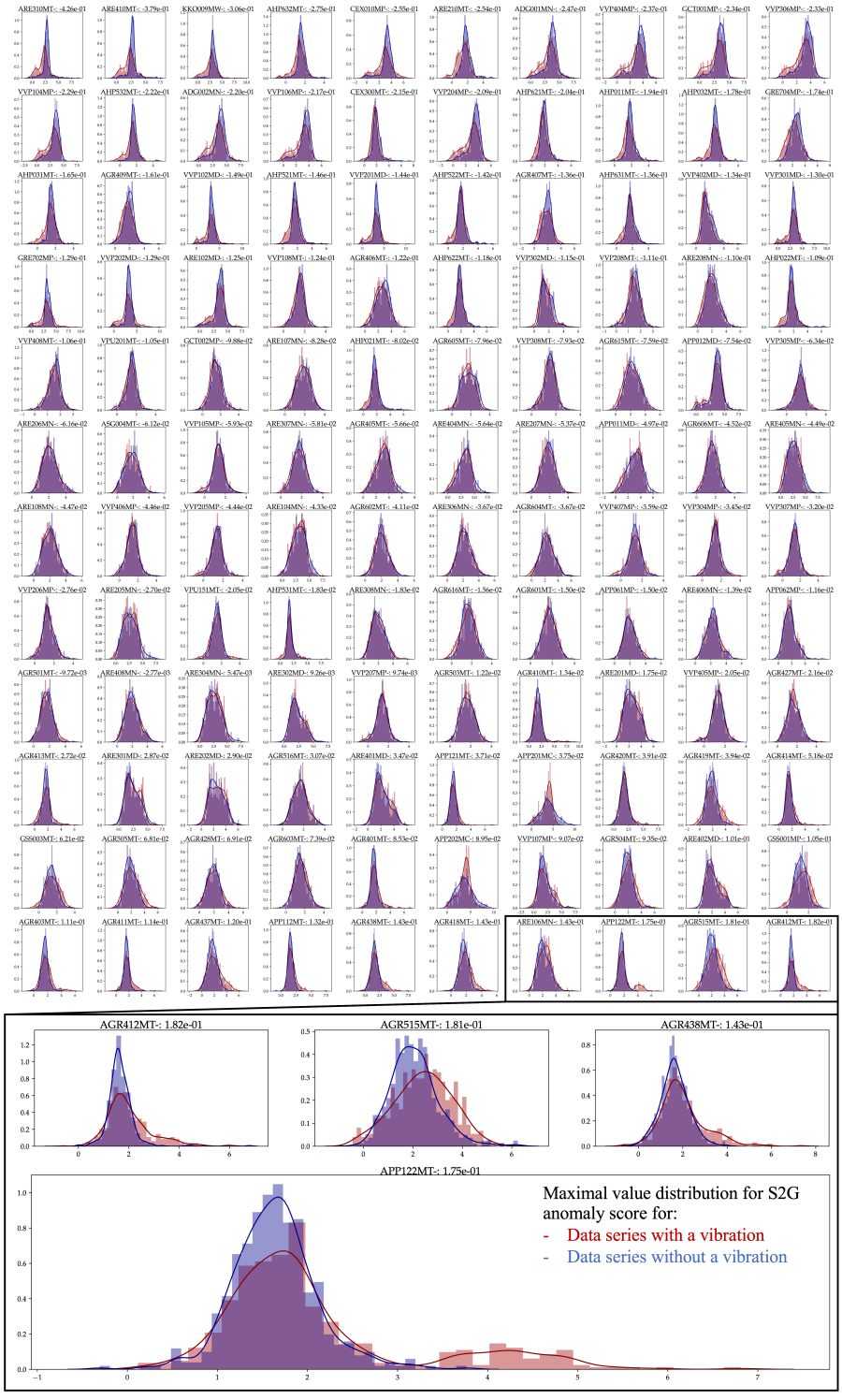


Figure 6.4: For each sensor, maximal Series2Graph anomaly score distribution for time series with a vibration (in red) and without any vibration (in blue). The distribution plots are sorted based on the difference of the red and blue distribution (in that order) as depicted in the title of each subplot.

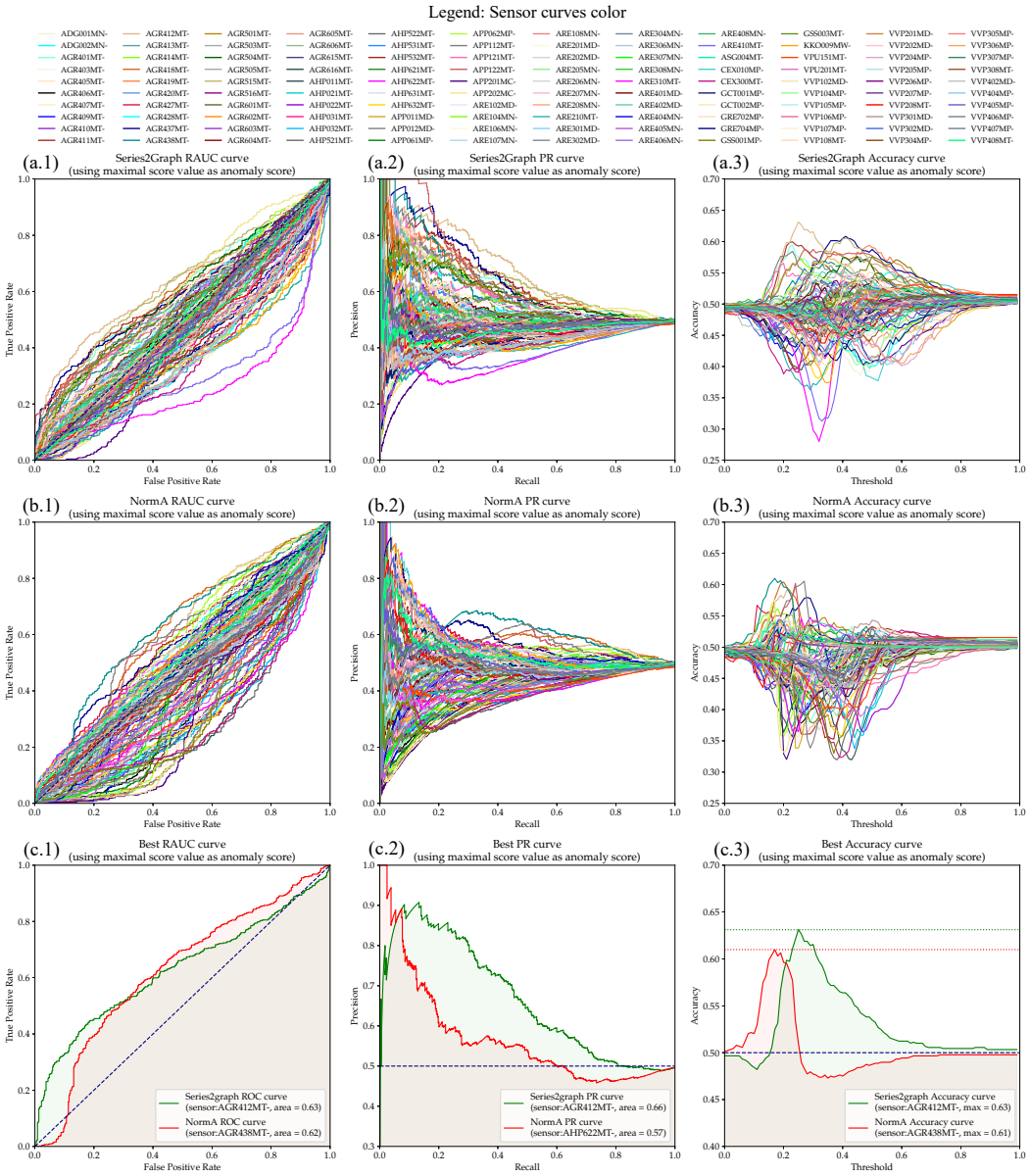
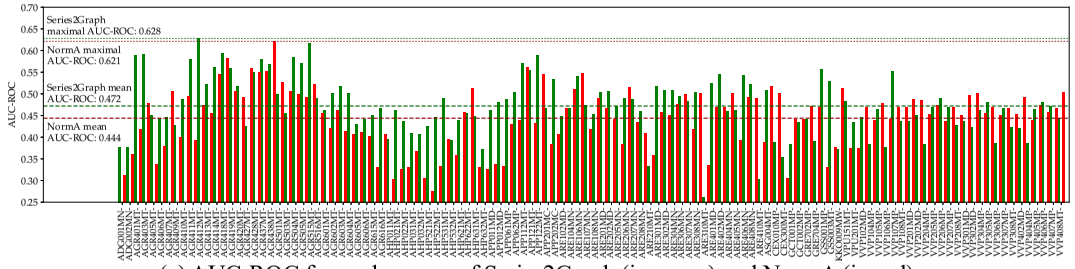
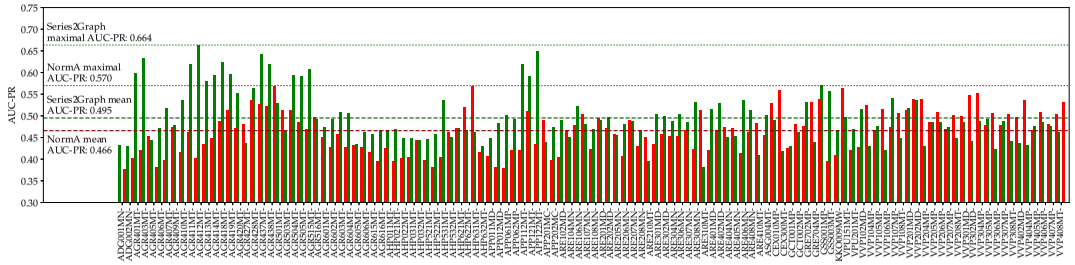


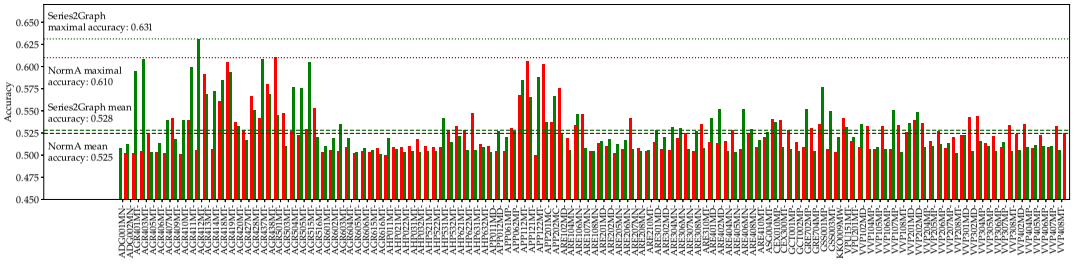
Figure 6.5: ROC, PR, and Accuracy curve for Series2Graph and NormA evaluated on each sensor separately and using the maximal anomaly score on the entire instance.



(a) AUC-ROC for each sensor of Series2Graph (in green) and NormA (in red)



(b) AUC-PR for each sensor of Series2Graph (in green) and NormA (in red)



(c) Accuracy for each sensor of Series2Graph (in green) and NormA (in red)

Figure 6.6: Detection Accuracy measures (Area Under the Curve for ROC, PR, and maximal accuracy that can be obtained with the best threshold) for Series2Graph and NormA evaluated on each sensor separately and using the maximal anomaly score on the entire instance.

This first analysis of NormA and Series2Graph anomaly scores distributions confirms that it is difficult to discriminate instances from the vibration classes and instances from the normal class for the vast majority of sensors. However, we observe that, for few sensors, we can discriminate some of the vibration class instances from normal class instances. Therefore, we now formally evaluate the accuracy of NormA and Series2Graph. For that purpose, we use the following three accuracy measures:

- **ROC curve** and **ROC-AUC**: We compute the true positive rate (i.e., the number of vibration instances correctly classified as belonging to the vibration class divided by the total number of instances in the vibration class) and the false positive rate (the number of normal instances being wrongly classified as belonging to the vibration class divided by the number of instances in the normal class) for 100 different thresholds between the minimum and the maximum value of all scores. From the latter, we obtain a curve called the ROC curve. We then use the area under this curve as an accuracy measure. This measure is called ROC-AUC.
- **Precision-Recall curve** and **PR-AUC**: We use the Precision and the Recall as accuracy measures. We compute the Precision and the Recall for a set of 100 different thresholds between the minimum and the maximum values of all the scores. We thus obtain a curve called the PR curve. We then use the area under this curve as an accuracy measure. This measure is called PR-AUC.
- **Classification accuracy curve**: We define the classification accuracy as the number of correctly classified instances among all the instances. We thus use a threshold and assign to the vibration class all the instances with a score higher than this threshold. We compute the classification accuracy for 100 different thresholds between the minimum and maximum scores of all the scores. We obtain a curve, and we can extract the maximal value on the curve. This maximal value corresponds to the maximal classification accuracy achievable with a threshold.

These three accuracy measures are depicted in Figure 6.5 and Figure 6.6. Figure 6.5(a) depicts the Series2Graph ROC curve (Figure 6.5(a.1)), PR curve (Figure 6.5(a.2)), and accuracy curve (Figure 6.5(a.3)) for each sensor. Similarly, Figure 6.5(b) depicts the NormA ROC curve (Figure 6.5(b.1)), PR curve (Figure 6.5(b.2)), and accuracy curve (Figure 6.5(b.3)) for each sensor. Figure 6.5(c) depicts, for Series2Graph (in green) and Norma (in red), the sensors providing the best ROC curve (Figure 6.5(c.1)), the best PR curve (Figure 6.5(c.2)) and the best accuracy curve (Figure 6.5(c.3)).

We observe that, as inferred from Figure 6.3 and Figure 6.4, Series2Graph and NormA are not accurate in detecting vibration using most of the sensors. This inaccuracy for both two methods is expected as most of the sensors are not related to the vibrating pumps. However, some sensors provide relatively higher accuracy. For example, the highest accuracy is obtained with sensor $AGR412MT-$ using series2Graph for both ROC-AUC (0.63), PR-AUC (0.66), and maximal achievable accuracy (0.63). For NormA, the highest accuracy is obtained with different sensors based on what accuracy measure we use. For example, NormA maximal ROC-AUC (0.62) and accuracy (0.61) are obtained using sensor $AGR438MT-$, whereas the maximal PR-AUC (0.57) is achieved with sensor $AHP622MT-$.

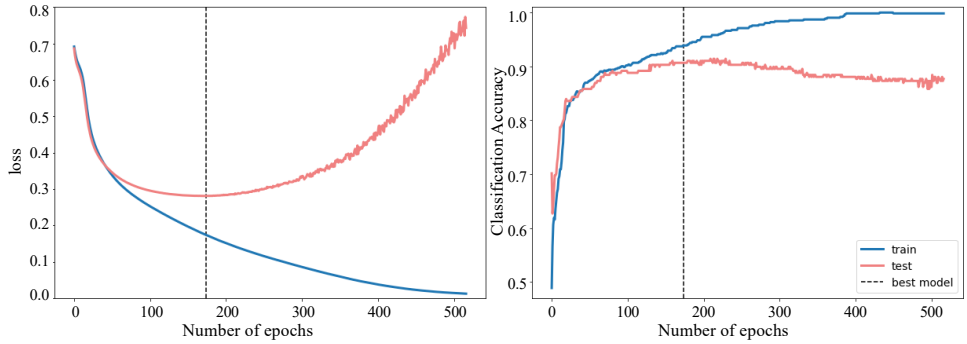


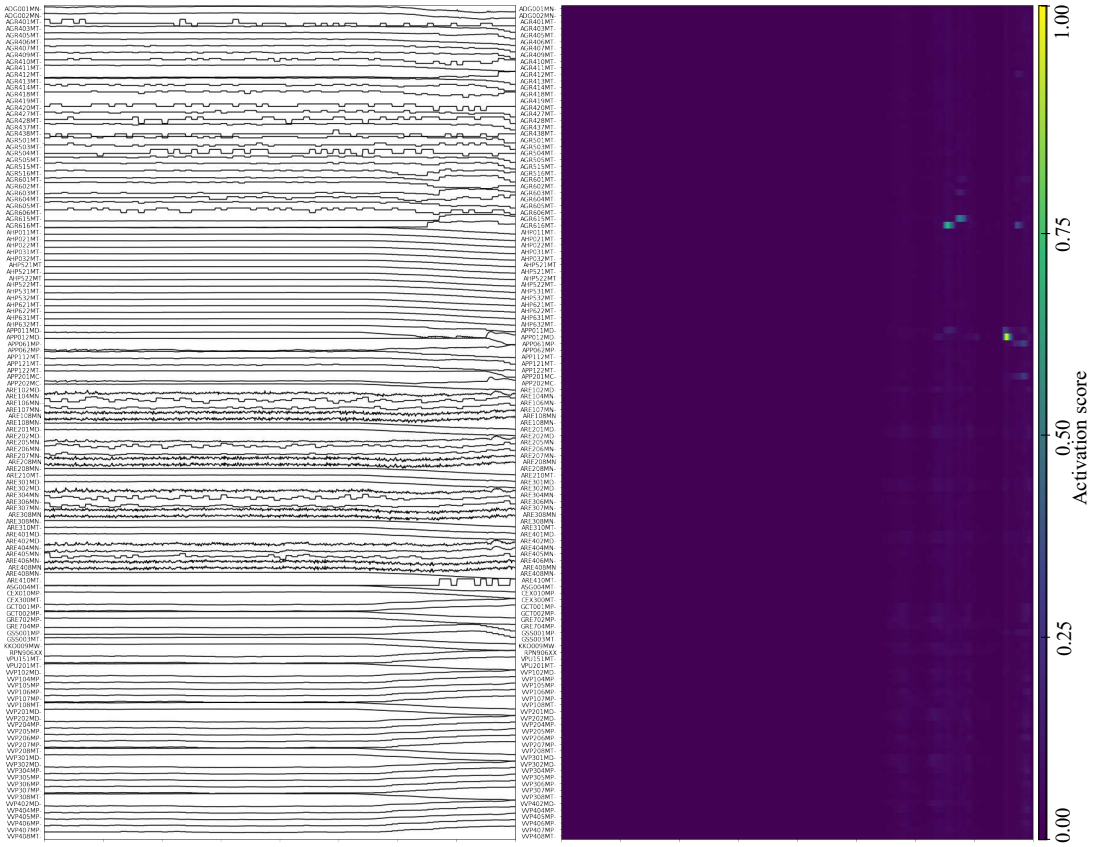
Figure 6.7: Training phase of dCNN. Loss and accuracy evolution when the number of epoch are increasing.

Overall, we observe that Series2Graph is more accurate than NormA, but both methods are not performing a high accuracy. Figure 6.6 summarizes the accuracy measures obtained using Series2Graph (in green) and NormA (in red) for every sensor. Moreover, Figure 6.6(c) demonstrates that two groups of sensors are providing slightly better accuracy than the rest of the sensors. The first group is the feed-water pump-turbine lubrification and control fluid system (AGR), and the second group is the turbine-driven feed-water pump system (APP). These two groups correspond to systems very close and dependent on the vibrating pump and are confirmed by the expert to be physically (i.e., containing functional information of the pumps such as sealing temperatures and outlet pump flow) and structurally consistent with the vibration detection task.

6.2.1.2 Supervised Detection

We now evaluate vibration detection using a supervised model. We use the dCNN (defined in Chapter 5) as our supervised model in this section. We train the dCNN model (using 5 convolutional layers with $(64, 128, 256, 256, 256)$ filters respectively; a kernel size of 3 and a padding of 2) on 70% of each class ($T^N_{\mathcal{M}}$ and $T^A_{\mathcal{M}}$) and we use the 30% left as a validation set. We set the batch size to 8 instances. We stop the training phase when the loss on the validation set does not reduce for the last 100 epochs. In total, we limit the training phase to 1000 epochs.

Figure 6.7 depicts the evolution of the loss and the classification accuracy (number of instances correctly classified divided by the total number of instances) as the number of epochs increases. Overall, dCNN achieve 0.91 of accuracy on the training set and 0.89 of accuracy on the validation set (whereas NormA and Series2Graph provide an accuracy of 0.63 and 0.61, respectively). We thus confirm that a supervised model is significantly more accurate than an unsupervised model for the vibration detection case. The latter underlines the benefits that can bring labels to model accuracy. As the dCNN accuracy is high, we can now use dCAM to identify discriminant subsequences (i.e., possible vibration precursors). Finally, the overall training step and dCAM computation require a full day in execution time on a GPU NVIDIA V100.



(a) Example of multivariate time series T from the vibration class $T^A_{\mathcal{M}}$.

(b) $dCAM_{T^A_{\mathcal{M}}}(T)$: Dimension-wise Class Activation Map of T for the vibration class $T^A_{\mathcal{M}}$.

Figure 6.8: Example of dimension-wise Class Activation Map (dCAM) for one multivariate data series instance from the vibration class (in blue: the non-activated parts of the data series, in yellow: the activated, and thus discriminant, subsequences of the data series).

6.2.2 Precursors identification Quantitative Evaluation

We concluded from the previous sections that supervised approaches outperform unsupervised method in the context of our use case. We now evaluate the relevance of the subsequences identified by our proposed approach dCAM. We thus compute dCAM for every instance belonging to the vibration class. Figure 6.8 illustrates an example of the dCAM (as depicted in Figure 6.8(b)) for a given vibration class instance (as depicted in Figure 6.8(a)). In the latter figure, the blue section in the dCAM corresponds to sections of the data series that the model (dCNN) did not consider useful to classify that data series as a vibration. On the contrary, the yellow sections are the subsequences of the data series that are considered the most discriminant and thus might explain why the vibration occurred. The subsequences highlighted in yellow can thus be interpreted as potential precursors of the vibration.

We now conduct further analysis on dCAM results across the entire vibration class. We conduct the following three analysis:

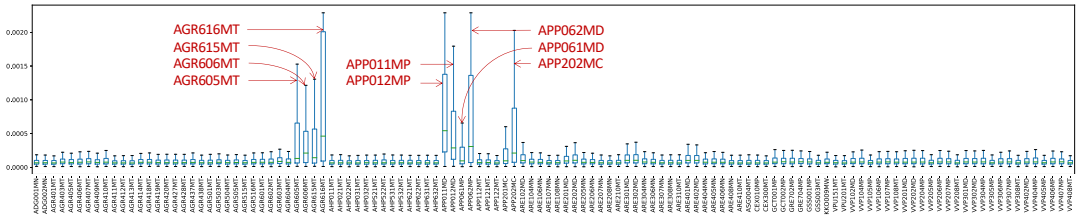


Figure 6.9: Aggregated activation score for dCAM per sensors for every timestamps. In red: the sensors names that are overall highly activated and that possibly contains one or several precursors.

- We first start by measuring the consistency of the detection as regards the structural information (i.e., are the sensors closely related to the vibrating system?). To minimize possible miss of information, we have artificially selected many sensors that are not all directly related to the vibrating pump. Nevertheless, we expect that the sensors that the model uses to classify instances are closely related to the vibrating pump. We thus verify this expectation. We also compare the structural consistency of dCAM with the temporal consistency of NormA and Series2Graph.
- We then continue by measuring the consistency of the detection (or activation) regarding temporality (i.e., are the subsequences detected close in time to the vibration?). Based on the experts' opinions, the possible precursors are more likely to be close to the vibration. We also compare the temporal consistency of dCAM with the temporal consistency of NormA and Series2Graph.

6.2.2.1 Structural consistency

We first measure the average activation score per sensor for every timestamp. Figure 6.9 depicts activation score box plot for each sensor when using dCAM. We observe that the activation scores returned by dCAM vary significantly between different sensors. We can easily distinguish nine sensors out of the 120. These sensors correspond to temperature measurements inside the feed-water pumps (sealing temperatures noted AGR605MT, AGR615MT for TPA1, and AGR606MT, AGR616MT for TPA2) and the outlet pump flow and pressures (noted APP011MD, APP061MP for TPA1, and APP012MD, APP062MP for TPA2). As highlighted in Figure 6.1(a), AGR and APP are sub-systems directly connected to the vibrating pump. Moreover, pressure and flow can directly influence the pump efficiency, with low-efficiency areas known to be conducive to vibration events. Thus, the sensors highlighted by our proposed approach dCAM are very consistent with the knowledge from experts and the functional structure of the plant.

Moreover, we can compare Figure 6.9 with Figure 6.6(c) and compare the activated sensors by dCAM with the sensors that provided the best accuracy for NormA and Series2Graph. We first observe that the same group of sensors is highlighted by NormA, Series2Graph and dCAM (AGR and APP). This confirms that both unsupervised and supervised methods are consistent. However, dCAM selects a smaller group of sensors (despite the total number of sensors) that is confirmed by the expert to be potentially related to the

occurrence of vibrations.

6.2.2.2 Temporal consistency

We then measure the evolution of the average activation score (for all sensors) in time obtained by dCAM (Figure 6.10(B)). Figure 6.10(B) depicts the quantiles values for each timestamp. The solid red line is the median, while every dotted grey line corresponds to the 20-quantiles. We first observe that the average activation score is higher when the vibration occurs (red vertical line in Figure 6.10). As it is unlikely to find precursors one hour before the vibration, we can thus confirm that dCAM results are consistent regarding temporality. We then observe the average anomaly score for some specific sensors (highlighted in red in Figure 6.9) that are the most activated across all vibration instances. We observe that, on average, all these sensors see their activation increases approximately 10 minutes before the vibrations. We thus explore in the following section the activated subsequences for these nine sensors.

Moreover, Figure 6.10(A) depicts the average score across all sensors for NormA (Figure 6.10(A.1)) and Series2Graph (Figure 6.10(A.2)). We observe that, similarly to dCAM, the score increases on average when we get closer to the vibration. Overall, we confirm that both unsupervised (NormA and Series2Graph) and supervised (dCAM) results are consistent regarding temporality.

6.2.3 Precursor identification Qualitative Evaluation

We now analyze the results returned by dCAM and discuss the information that the knowledge experts can gain from it. We mainly focus our analysis on the nine most activated sensors (i.e., sensors depicted in Figure 6.10(b)). We cluster (computed with the usual k-means using Euclidean distance) the 15 minutes long subsequences with the highest activation score for a vibration instance. The centroids of these clusters are thus representing the different shape categories within each sensor detected by dCAM. Therefore we can limit our analysis on this reduced (but relevant) set of centroids. Figure 6.11 depicts the 15 minutes long subsequences clusters in the nine sensors mentioned above. For each cluster, the time distribution histogram is displayed below. We first notice that the majority of the subsequences (for all clusters) happened while the vibration is detected (such as, for instance, the cluster depicted in Figure 6.11(b.2)). As understood by the experts, these subsequences correspond to a specific action (such as an increase or decrease of the water flow through the feed-water pump to either increase or decrease the power generated by the plant) that could lead to vibrations but that are not avoidable. Thus dCAM first permits the expert to confirm and visualize which subsequences are directly correlated to the vibration. Then, several subsequences detected by dCAM are anterior to the vibration (such as Figure 6.11(a.2),(a.3),(b.1),(c.1),(c.2),(e.1),(g.3)). These subsequences could correspond to precursors of the vibration and would require to be carefully inspected by the experts. For instance, knowledge experts conclude that clusters such as in Figure 6.11(e.1.1) correspond to unusual variations of the sealing temperature of the pump and lead them to investigate in detail specific examples. Overall, our proposed approach dCAM permits the experts to

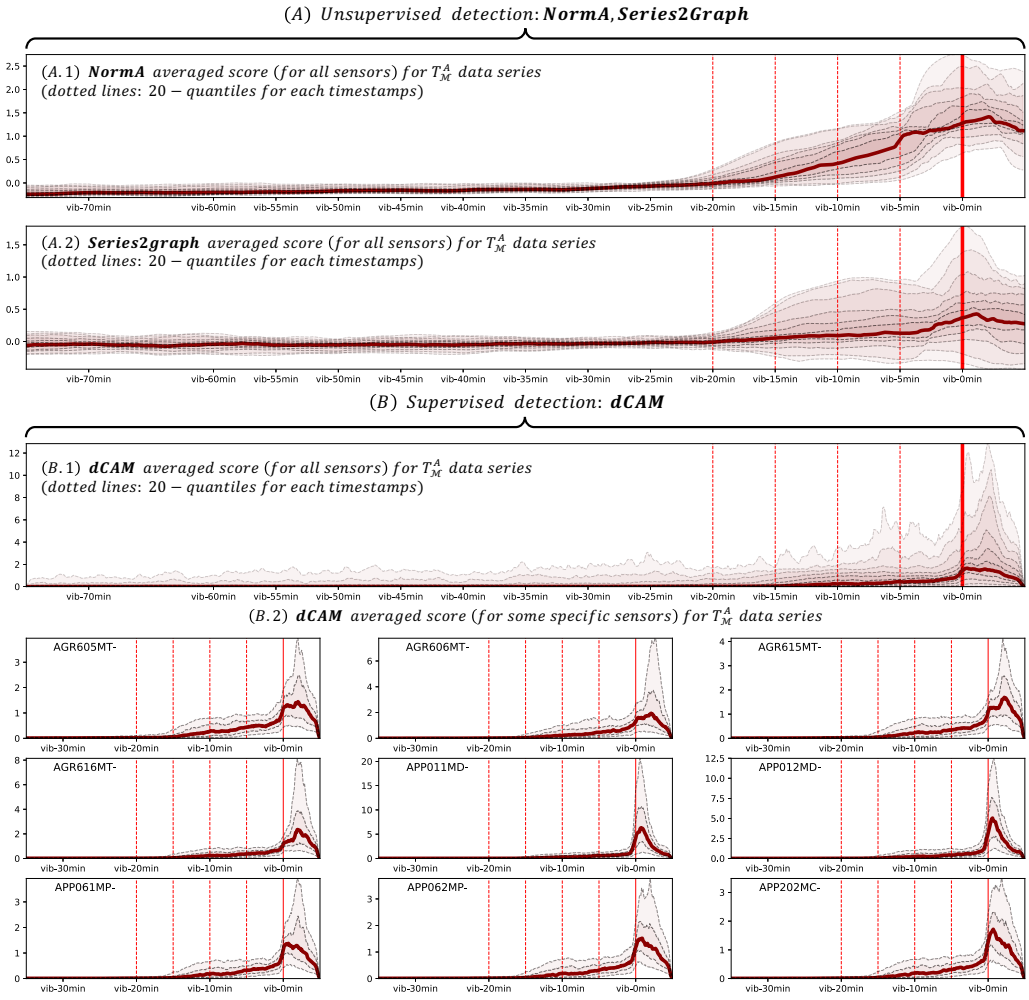


Figure 6.10: Aggregated NormA (A.1), Series2Graph (A.2) and dCAM activation score for all sensors (B.1). Aggregated dCAM activation scores for some specific sensors (B.2). Red shades correspond to quantile interval (0.05-0.95,0.10-0.90,0.15-0.85, etc for (A and B.1) and only 0.3-0.7, 0.4-0.6 for (B.2)). The solid red line is the median values for each timestamp.

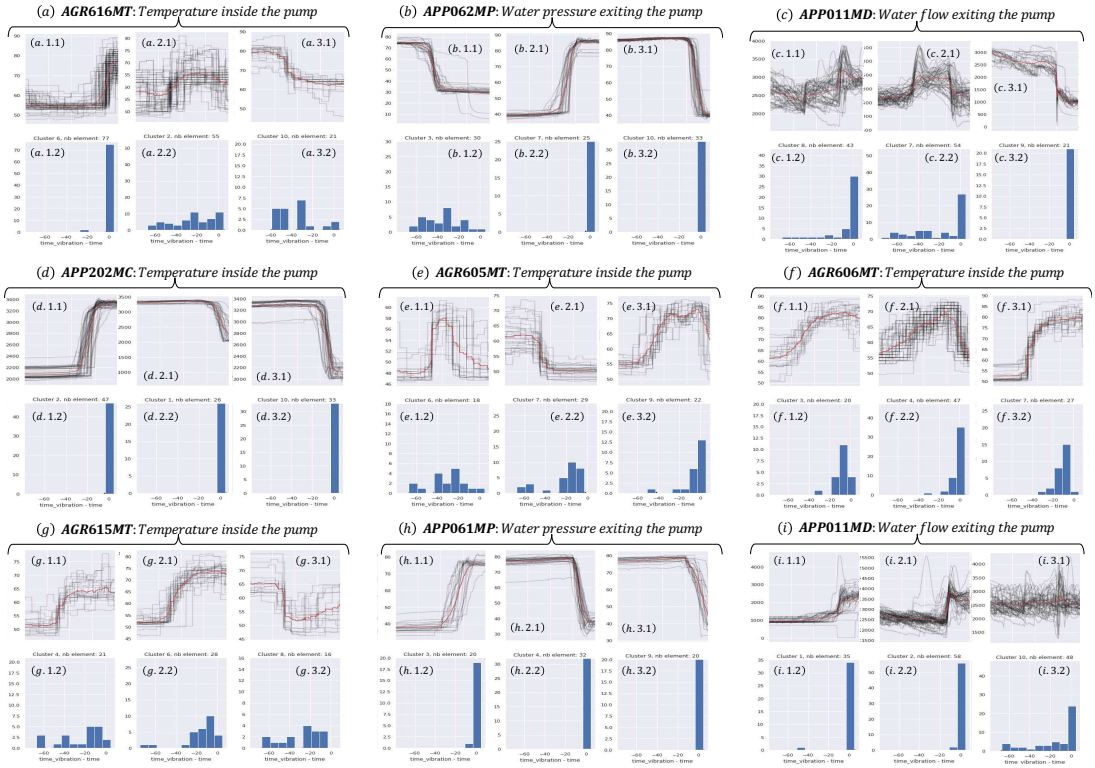


Figure 6.11: Subsequences clusters (**.1) (and their time distribution compared to the vibration timestamps (**.2)) detected as precursors of vibration by dCAM for the 9 most activated sensors.

build a dictionary of patterns related to a targeted anomaly. This dictionary can be used to raise alarms and help the experts to avoid critical situations. Thus the investigation by the expert could be significantly reduced.

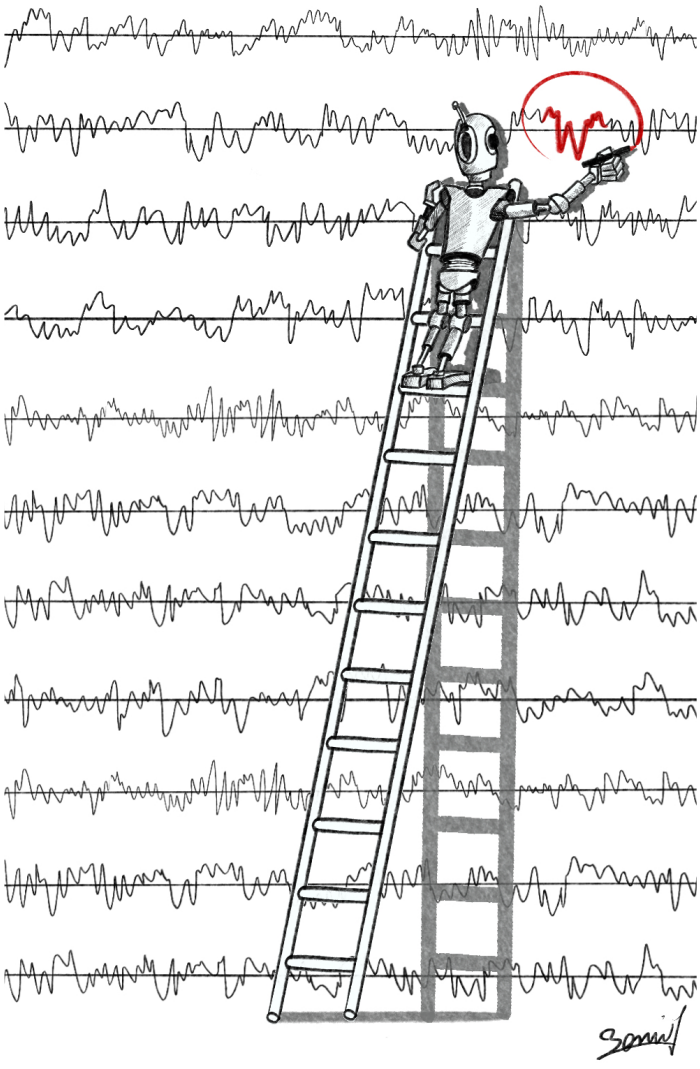
6.3 Summary and Conclusion

In this chapter, we described a real-industrial use case. The latter is related to the detection of unwanted vibrations in turbine-driven feed-water pump systems inside French nuclear power plants. We explored two possible ways: (i) the unsupervised detection of vibration, (ii) the supervised classification of vibrations, and the detection of their precursors. For the first case, we applied the unsupervised methods NormA and Series2Graph defined in Chapter 3. We evaluated the accuracy of these two methods using labels provided by the experts. For the second case, we applied the supervised method dCNN/dCAM defined in Chapter 5. We used the labels provided by the experts to train our model dCNN, and use dCAM to detect discriminant subsequences (and possible precursors of the vibration) within every dimension. This chapter demonstrated that supervised method dCNN/dCAM are significantly more accurate than unsupervised methods, even though NormA and Series2Graph provide interesting results consistent with expert knowledge. We then compared the results

obtain by dCAM with the expert knowledge. Finally, we confirmed with the experts the validity and the accuracy of our proposed approach dCNN/dCAM. As future work, we plan to explore in details the differences between unsupervised methods (such as Series2Graph) and dCNN by extracting the subsequences that are highlighted by dCAM and not by the unsupervised model.

Chapter 7

Conclusions and Perspectives



7.1 Contributions Summary

Data series is a very usual data type. Many scientific applications have now generated extensive collections of data series that are required to be analyzed. Moreover, anomaly detection is a crucial task for most scientific and industrial applications. This thesis tackled this task with special care to constraints related to large collections of data series.

At first, we described in Chapter 2 the related works proposed in the literature, which can be used to detect abnormal subsequences either in an unsupervised, semi-supervised and supervised manner. After discussing their limitations, we presented in Chapter 3 two new approaches, NormA and Series2Graph, that aim to detect on an unsupervised manner subsequences anomalies in data series. We described in detail the computational steps of these algorithms. The first method, NormA, extracts subsequences representing the data series' recurrent behaviors and uses the distance to these subsequences as the anomaly score. The second method, Series2Graph, aims to embed the data series into a directed graph, in which, the nodes and the edges encode are used to detect abnormal subsequences. Theoretically and empirically, we underlined the limitation of the discord-based approaches and density-based approaches for large data series that can contain several similar anomalies. We then empirically demonstrated our two developed approaches' superiority in terms of anomaly detection accuracy and execution time. Finally, we developed two user interfaces that depict the computational steps of NormA and Series2Graph, respectively.

In a second phase, we tackled in Chapter 4 the specific case of unsupervised subsequence anomaly detection for streaming data series. We first analyzed the limitations of state-of-the-art approaches, including NormA and Series2Graph, when applied to data streams. We then introduced SAND, a novel unsupervised approach for subsequence anomaly detection in streaming sequences. SAND is based on a set representation of the subsequences in a data stream (inspired from NormA). We proposed a user interface implementation of our approach that simplifies the usage and facilitates the comprehension of SAND. Finally, we conducted an experimental analysis of several synthetic and real datasets. The synthetic datasets are specifically built in order to simulate changes in normality. These experiments demonstrated the benefits of our approach in terms of efficiency and accuracy.

In a third step, we study in Chapter 5 the specific case of supervised anomaly detection with the final purpose of identifying precursors. Thus, we tackled this task as data series classification task in the general case. Even though data series classification using deep learning has attracted much attention, existing techniques for explaining the classification decisions fail in multivariate data series. Moreover, simple extensions, such as cCNN, do not meet the usual CNN-based architectures' performances. That is why we introduced a novel approach, dCAM, based on convolutional neural networks, enabling us to detect the discriminant subsequences within individual dimensions of a multivariate data series. In order to apply dCAM, we provided a transformation to the network architecture that we call dCNN (for the usual CNN architecture). Such extension can be applied to any other convolutional-based architecture with a Global Average Pooling (GAP) layer. We thus also introduced dResNet and dInceptionTime (for the ResNet and the InceptionTime architectures). The experimental evaluation with synthetic and real datasets demonstrated our approach's benefits and superiority in discriminant feature discovery and classification

explanation for multivariate time series. Moreover, we evaluated the influences between several important parameters to give the user some keys to assess the accuracy of dCNN and dCAM.

Finally, we described in Chapter 6 a real-industrial use case. It is related to detecting unwanted vibration in turbine-driven feed-water pump systems inside EDF French nuclear power plants. We explored two possible tasks: (i) the unsupervised detection of vibrations, (ii) the supervised classification of vibrations, and the detection of their precursors. For the first objective, we applied the unsupervised methods NormA and Series2Graph we developed. We evaluated the accuracy of these two algorithms using labels provided by the experts. For the second task, we applied the supervised method dCNN/dCAM. We used the labels provided by the experts to train our model dCNN, and used dCAM to detect discriminant subsequences (and possible precursors of the vibration) within every dimension. We demonstrated that the supervised method dCNN/dCAM is significantly more accurate than unsupervised methods, even though NormA and Series2Graph provided interesting results consistent with expert knowledge. We then briefly interpreted the results obtained by dCAM with respect to the expert knowledge. Finally, we confirmed with the experts the validity, practical, usefulness, and accuracy of our proposed approach dCNN/dCAM.

7.2 Open Research Directions

Our work raises new research questions and opens up several interesting research directions, which are discussed in the following sections.

7.2.1 Series2Graph and NormA for Multivariate Data Series

As described in Chapter 3, NormA and Series2Graph are proposed for univariate time series only. A natural research direction is thus to provide modifications to these two methods such that multivariate anomalies could be found. This objective requires to describe formally what a multivariate abnormal subsequence is in the specific case of unsupervised detection. For NormA, we need to define a proper similarity measure between multivariate subsequences, which remains the main challenge to upgrade NormA for multivariate data series. For Series2Graph, the challenge is different. A first research direction would be to find a new projection function that can map multivariate data series into a two-dimensional space. If such a method exists, the following steps would remain unchanged. A trivial solution would be to consider the multivariate subsequences as a vector. However, such representation would erase the structure of a multivariate data series.

7.2.2 Series2Graph for Streaming Data Series

We proposed a new method, inspired by NormA, in Chapter 4. However, a streaming method using a subsequence graph, as for Series2Graph, is still an open problem. As defined in Chapter 3, Series2Graph relies on a projection step that uses PCA transformation. The

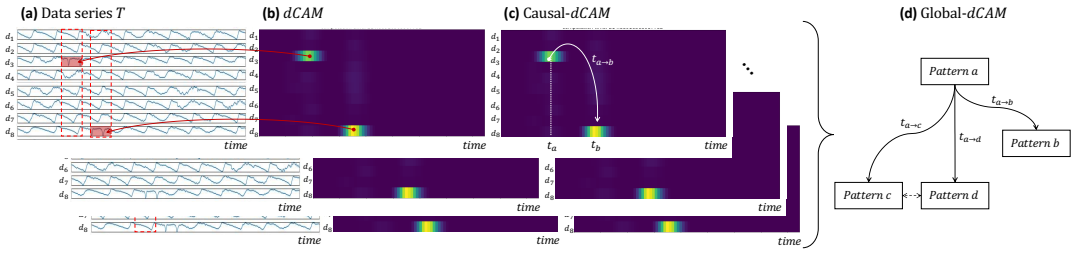


Figure 7.1: Local explanation with (b) dCAM over (a) a data series T. Open-research directions aims to obtain (c) a causal-dCAM and (d) a global explanation aggregating all the dCAM over the entire dataset.

first challenge would be to change this projection such that old subsequences are discarded incrementally, and new subsequences corresponding to a new behavior would change the projection while minimizing the throughput.

7.2.3 Distributed implementations

As the number of data series collected by several scientific applications increases, the execution time is a serious criterion that needs to be studied. In this work, we addressed the latter, and we proposed approaches that can scale for large databases. However, to benefit from all the available core and CPUs, distributed implementations of our proposed method remain an open research direction. For example, a recent research paper has already proposed a distributed implementation of Series2Graph [100]. However, interesting future works would be to propose a distributed implementation for NormA and SAND.

7.2.4 Non-linear projection for Series2Graph first step

As described in Chapter 3, Series2Graph relies on a linear projection step, which uses PCA to reduce the subsequence length to 2 dimensions. This linear projection is sufficient on our dataset corpus used in the experimental section. However, Series2Graph is not performing well on some specific datasets (such as the multi-normality data series introduced in Chapter 4). Changing the projection can be a solution to improve Series2Graph accuracy. For instance, studying the impact and the benefit of non-linear projections (such as AutoEncoders latent space, or t-SNE-based embedding [80]) on Series2Graph accuracy and execution time is an interesting open-research direction.

7.2.5 Optimization of Memory and Execution Time Complexity for dCAM

In Chapter 5, we proposed dCAM, which uses a data structure that duplicates several time dimensions of the multivariate data series input. Thus, this duplication implies a more sig-

nificant memory usage and execution time than the usual convolutional-based architecture. This drawback is, therefore, a significant research direction. At first sight, memory usage can be solved by replacing information duplication with pointers. Moreover, the information duplication might create duplicated operations. Investigating how such duplicated operations can be removed is a challenging open-research direction.

7.2.6 From local to global explanation with dCAM

In Chapter 5, we proposed dCAM, a dimension-wise Class Activation Map that overcomes the limitation of the usual Class Activation Map and provides a multivariate data series that highlights, in each dimension, the discriminant features (or subsequences). Thus, for each multivariate data series input, we obtain an explanation emphasizing the reasons for the classification as a given class. Nevertheless, the explanation brought by dCAM is local (i.e., corresponding to one input only). As illustrated in Figure 7.1(d), a global explanation would require aggregating all the dCAMs computed over the entire dataset and retrieving global discriminant features. In Chapter 6, we introduced a pipeline that clustered all the subsequences highlighted by dCAMs and extracted centroids. These are typical patterns that can globally explain and describe a class and can be seen as Shapelets [123]. Bridging the local explanation provided by dCAM to a global Shapelet-like explanation is a challenging open research direction.

7.2.7 Adding temporal explanation to dCAM

For the specific multivariate data series classification explanation, discriminant features can be correlated or co-occurrent of specific patterns in different dimensions. We proposed dCAM in Chapter 5 to detect such discriminant features. However, as illustrated in Figure 7.1(c), discriminant features can also be related to causal links between two patterns occurring one after the other in one or several dimensions. Such a link could be theoretically identified with CNN-based approaches using kernels of considerable lengths (such as Inception Time [52]). However, the resulting dCAM, as proposed in Chapter 5, does not provide any causal link or, in general, any time-dependent information. Thus, this is an important open-research direction.

7.2.8 Applicability to non-continuous data series

As described in the different experimental evaluations and industrial use cases, we only applied our methods to continuous data series. However, because of the input type (such as boolean sensors) or discretization processes, the data series to analyze could be non-continuous. This discontinuity brings new challenges. For instance, distances functions defined in Chapter 2 might not be relevant for such type of data series. Therefore, adapting our proposed approaches to non-continuous data series is an open-research direction.

7.2.9 Exploration of other industrial use cases

We illustrated the applicability and interest of our developments through an industrial application in Chapter 6. However, as mentioned in the introduction, EDF experts are studying other related use cases in which our proposed approaches could be used. Thus, we plan to evaluate and validate our proposed approaches over other industrial applications.

Bibliography

[1] D. Abboud, M. Elbadaoui, W.A. Smith, and R.B. Randall. "Advanced bearing diagnostics: A comparative study of two powerful approaches". In: *Mechanical Systems and Signal Processing* 114 (2019), pp. 604–627 (cit. on pp. 5, 57).

[2] Ali Abdul-Aziz, Mark R Woike, Nikunj C Oza, Bryan L Matthews, and John D lekki. "Rotor health monitoring combining spin tests and data-driven anomaly detection methods". In: *Structural Health Monitoring* (2012) (cit. on pp. 82, 113).

[3] Jérôme Antoni and Pietro Borghesani. "A statistical methodology for the design of condition indicators". In: *Mechanical Systems and Signal Processing* 114 (2019), pp. 290–327 (cit. on pp. 5, 57).

[4] Anthony Bagnall, Richard L. Cole, Themis Palpanas, and Kostas Zoumpatianos. "Data Series Management (Dagstuhl Seminar 19282)". In: *Dagstuhl Reports* 9.7 (2019). Ed. by Anthony Bagnall, Richard L. Cole, Themis Palpanas, and Konstantinos Zoumpatianos, pp. 24–39 (cit. on p. 2).

[5] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances". In: *Data Mining and Knowledge Discovery* 31 (Nov. 2017), 606–660 (cit. on pp. 9, 46).

[6] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. "Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles". In: *IEEE Transactions on Knowledge and Data Engineering* 27.9 (2015), pp. 2522–2535 (cit. on pp. 9, 46, 47).

[7] Anthony Bagnall, Luke M Davis, Jon Hills, and Jason Lines. "Transformation Based Ensembles for Time Series Classification." In: *SDM*. Vol. 12. SIAM. 2012, pp. 307–318 (cit. on p. 47).

[8] V. Barnet and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, Inc., 1994 (cit. on pp. 2, 7, 59).

[9] I. V. Bezsudnov and A. A. Snarskii. "From the time series to the complex networks: The parametric natural visibility graph". In: *Physica A Statistical Mechanics and its Applications* 414 (2014), pp. 53–60 (cit. on p. 37).

[10] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. "A Review on Outlier/Anomaly Detection in Time Series Data". In: *ACM Comput. Surv.* 54.3 (Apr. 2021) (cit. on p. 19).

[11] Paul Boniol. "Unsupervised Subsequence Anomaly Detection in Large Sequences". In: *Proceedings of the VLDB 2020 PhD Workshop co-located with the 46th International Conference on Very Large Databases (VLDB 2020), ONLINE, August 31 - September 4, 2020*. Ed. by Ziawasch Abedjan and Katja Hose. Vol. 2652. CEUR Workshop Proceedings. CEUR-WS.org, 2020 (cit. on p. 57).

[12] Paul Boniol and Themis Palpanas. “Series2Graph: Graph-Based Subsequence Anomaly Detection for Time Series”. In: *Proc. VLDB Endow.* 13.12 (July 2020), 1821–1834 (cit. on p. 68).

[13] Paul Boniol, Michele Linardi, Federico Roncallo, and Themis Palpanas. “Automated Anomaly Detection in Large Sequences”. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 1834–1837 (cit. on p. 58).

[14] Paul Boniol, Themis Palpanas, Mohammed Meftah, and Emmanuel Remy. “GraphAn: Graph-Based Subsequence Anomaly Detection”. In: *Proc. VLDB Endow.* 13.12 (Aug. 2020), 2941–2944 (cit. on p. 93).

[15] Paul Boniol, Michele Linardi, Federico Roncallo, and Themis Palpanas. “SAD: An Unsupervised System for Subsequence Anomaly Detection”. In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 1778–1781 (cit. on p. 93).

[16] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. “SAND in Action: Subsequence Anomaly Detection for Streams”. In: *Proc. VLDB Endow.* 14.12 (2021), pp. 2867–2870 (cit. on p. 119).

[17] Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. “SAND: Streaming Subsequence Anomaly Detection”. In: *Proc. VLDB Endow.* 14.10 (2021), pp. 1717–1729 (cit. on p. 99).

[18] Paul Boniol, Michele Linardi, Federico Roncallo, Themis Palpanas, Mohammed Meftah, and Emmanuel Remy. “Unsupervised and scalable subsequence anomaly detection in large data series”. In: *The VLDB Journal* (2021) (cit. on pp. 58, 102).

[19] Elizabeth Bradley and Holger Kantz. “Nonlinear time-series analysis revisited”. In: *Chaos* 25.9, 097610 (2015), p. 097610 (cit. on p. 38).

[20] Markus M. 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*. SIGMOD ’00. New York, NY, USA: ACM, 2000, pp. 93–104 (cit. on p. 27).

[21] Peter G. Bryant. “On the Minimum Description Length (MDL) Principle for Hierarchical Classifications”. In: *Data Science, Classification, and Related Methods*. Ed. by Chikio Hayashi, Keiji Yajima, Hans-Hermann Bock, Noboru Ohsumi, Yutaka Tanaka, and Yasumasa Baba. Tokyo: Springer Japan, 1998, pp. 182–186 (cit. on p. 63).

[22] Yingyi Bu, Oscar Tat-Wing Leung, Ada Wai-Chee Fu, Eamonn J. Keogh, Jian Pei, and Sam Meshkin. “WAT: Finding Top-K Discords in Time Series Database”. In: *SDM*. 2007 (cit. on p. 31).

[23] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection: A Survey”. In: *ACM Comput. Surv.* 41.3 (July 2009) (cit. on p. 19).

[24] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection for Discrete Sequences: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* 24.5 (2012), pp. 823–839 (cit. on p. 40).

[25] Huanhuan Chen, Fengzhen Tang, Peter Tino, and Xin Yao. "Model-Based Kernel for Efficient Time Series Analysis". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. Chicago, Illinois, USA: Association for Computing Machinery, 2013, 392–400 (cit. on p. 9).

[26] Bill Chiu, Eamonn Keogh, and Stefano Lonardi. "Probabilistic Discovery of Time Series Motifs". In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '03. Washington, D.C.: Association for Computing Machinery, 2003, 493–498 (cit. on p. 24).

[27] Christophe Croux, Sarah Gelper, and Koen Mahieu. "Robust control chart for time series data". In: *Katholieke Universiteit Leuven, Open Access publications from Katholieke Universiteit Leuven* 38 (Jan. 2010) (cit. on p. 40).

[28] Zhicheng Cui, Wenlin Chen, and Yixin Chen. "Multi-Scale Convolutional Neural Networks for Time Series Classification". In: *CoRR* abs/1603.06995 (2016) (cit. on pp. 9, 46, 50).

[29] *Dash documentation* (cit. on pp. 93, 119).

[30] Hoang Anh Dau, Anthony J. Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ratanamahatana, and Eamonn J. Keogh. "The UCR time series archive". In: *IEEE/CAA Journal of Automatica Sinica* 6 (2019), pp. 1293–1305 (cit. on pp. 9, 46, 47, 114, 132).

[31] Jesse Davis and Mark Goadrich. "The Relationship between Precision-Recall and ROC Curves". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. New York, NY, USA: Association for Computing Machinery, 2006, 233–240 (cit. on p. 133).

[32] J.-P Eckmann, S. Oliffson Kamphorst, and D Ruelle. "Recurrence Plots of Dynamical Systems". In: *Europhysics Letters (EPL)* 4.9 (1987), pp. 973–977 (cit. on p. 39).

[33] Philippe Esling and Carlos Agon. "Time-series data mining". In: *ACM Computing Surveys* 45.1 (2012), p. 12 (cit. on pp. 9, 46).

[34] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. "Deep learning for time series classification: a review". In: *Data Mining and Knowledge Discovery* 33 (2019), pp. 917–963 (cit. on pp. 9, 46, 48–52, 134).

[35] Pavel Filonov, Andrey Lavrentyev, and Artem Vorontsov. "Multivariate Industrial Time Series with Cyber-Attack Simulation: Fault Detection Using an LSTM-based Predictive Data Model". In: *CoRR* abs/1612.06676 (2016) (cit. on p. 42).

[36] Ada Wai-chee Fu, Oscar Tat-Wing Leung, Eamonn Keogh, and Jessica Lin. "Finding Time Series Discords Based on Haar Transform". In: *Proceedings of the Second International Conference on Advanced Data Mining and Applications*. ADMA'06. Xi'an, China: Springer-Verlag, 2006, 31–41 (cit. on p. 31).

[37] Yixin Gao, S. Vedula, Carol E. Reiley, N. Ahmidi, B. Varadarajan, Henry C. Lin, L. Tao, L. Zappella, B. Béjar, D. Yuh, C. C. Chen, R. Vidal, S. Khudanpur, and Gregory Hager. "JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS) : A Surgical Activity Dataset for Human Motion Modeling". In: 2014 (cit. on pp. 140, 142).

[38] Zhong-Ke Gao and Ningde Jin. "Complex network from time series based on phase space reconstruction". In: *Chaos (Woodbury, N.Y.)* 19 (Sept. 2009), p. 033137 (cit. on p. 39).

[39] Zhong-Ke Gao, Michael Small, and Juergen Kurths. "Complex network analysis of time series". In: *EPL (Europhysics Letters)* 116 (Dec. 2016), p. 50001 (cit. on p. 39).

[40] Ary L. Goldberger, Luis A. Nunes Amaral, L Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and Harry Eugene Stanley. "PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals." In: *Circulation* 101.23 (2000), pp. 215–220 (cit. on pp. 28, 82, 113, 118).

[41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680 (cit. on p. 43).

[42] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. "Data Augmentation for Time Series Classification using Convolutional Neural Networks". In: *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*. 2016 (cit. on pp. 9, 50).

[43] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. "Clustering Data Streams: Theory and Practice". In: *IEEE Trans. on Knowl. and Data Eng.* 15.3 (Mar. 2003), 515–528 (cit. on p. 26).

[44] Medina Hadjem, Farid Naït-Abdesselam, and Ashfaq Khokhar. "ST-segment and T-wave anomalies prediction in an ECG data using RUSBoost". In: *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*. 2016, pp. 1–6 (cit. on pp. 5, 57).

[45] N. Halko, P. Martinsson, and J. Tropp. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Rev.* 53 (2011), pp. 217–288 (cit. on p. 80).

[46] Sahand Hariri, Matias Carrasco Kind, and Robert J. Brunner. "Extended Isolation Forest". In: *IEEE Transactions on Knowledge and Data Engineering* 33.4 (2021), pp. 1479–1489 (cit. on p. 30).

[47] Douglas M Hawkins. *Identification of Outliers*. English. Springer Netherlands, 1980 (cit. on p. 17).

[48] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. "Support vector machines". In: *IEEE Intelligent Systems and their Applications* 13.4 (1998), pp. 18–28 (cit. on p. 40).

[49] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. "Classification of time series by shapelet transformation". In: *Data Mining and Knowledge Discovery* 28.4 (2014), pp. 851–881 (cit. on p. 47).

[50] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780 (cit. on p. 42).

[51] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. "Evaluating Surgical Skills from Kinematic Data Using Convolutional Neural Networks". In: *MICCAI*. 2018 (cit. on pp. 9, 134, 142, 143).

[52] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre Alain Muller, and François Petitjean. "InceptionTime: finding AlexNet for time series classification". In: *Data Mining and Knowledge Discovery* 34 (Sept. 2020), 1936–1962 (cit. on pp. 9, 50, 127, 134, 169).

[53] F. Itakura. "Minimum prediction residual principle applied to speech recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1 (1975), pp. 67–72 (cit. on p. 25).

[54] Holger Kantz and Thomas Schreiber. *Nonlinear Time Series Analysis*. 2nd ed. Cambridge University Press, 2003 (cit. on p. 38).

[55] Holger Kantz and Thomas Schreiber. *Nonlinear Time Series Analysis*. New York, NY, USA: Cambridge University Press, 2003 (cit. on p. 72).

[56] Matthew B. Kennel, Reggie Brown, and Henry D. I. Abarbanel. "Determining embedding dimension for phase-space reconstruction using a geometrical construction". In: *Phys. Rev. A* 45 (1992), pp. 3403–3411 (cit. on p. 38).

[57] E. Keogh, J. Lin, and A. Fu. "HOT SAX: efficiently finding the most unusual time series subsequence". In: *Fifth IEEE International Conference on Data Mining (ICDM'05)*. 2005, 8 pp.– (cit. on p. 118).

[58] Eamonn Keogh and Jessica Lin. "Clustering of time-series subsequences is meaningless: implications for previous and future research". In: *KAIS* 8.2 (Aug. 2004), pp. 154–177 (cit. on p. 62).

[59] Eamonn Keogh, Stefano Lonardi, and Chotirat Ann Ratanamahatana. "Towards Parameter-free Data Mining". In: *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '04. New York, NY, USA: ACM, 2004, pp. 206–215 (cit. on p. 73).

[60] Eamonn J. Keogh, Stefano Lonardi, Chotirat Ratanamahatana, Li Wei, Sanghee Lee, and John C. Handley. "Compression-based data mining of sequential data". In: *Data Mining and Knowledge Discovery* 14 (2006), pp. 99–129 (cit. on p. 31).

[61] H. S. Kim, R. Eykholt, and J. D. Salas. "Nonlinear dynamics, delay times, and embedding windows". In: *Physica D Nonlinear Phenomena* 127 (Mar. 1999), pp. 48–60 (cit. on p. 38).

[62] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015 (cit. on p. 49).

[63] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105 (cit. on p. 48).

[64] Dafne van Kuppevelt, Vincent van Hees, and Christiaan Meijer. *PAMAP2 dataset preprocessed v0.3.0*. July 2017 (cit. on p. 7).

[65] Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuño. "From time series to complex networks: The visibility graph". In: *Proceedings of the National Academy of Science* 105.13 (2008), pp. 4972–4975 (cit. on pp. 36, 39).

[66] Balaji Lakshminarayanan, Daniel M. Roy, and Yee Whye Teh. "Mondrian Forests: Efficient Online Random Forests". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, 3140–3148 (cit. on p. 113).

[67] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 50).

[68] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". English (US). In: *Nature* 521.7553 (May 2015), pp. 436–444 (cit. on p. 48).

[69] Dan Li, Dacheng Chen, Jonathan Goh, and See-Kiong Ng. "Anomaly Detection with Generative Adversarial Networks for Multivariate Time Series". In: *CoRR* abs/1809.04758 (2018) (cit. on pp. 44, 45).

[70] Xiaosheng Li and Jessica Lin. "Linear Time Motif Discovery in Time Series". In: *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM. 2019, pp. 136–144 (cit. on p. 62).

[71] Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn J. Keogh. "Matrix profile goes MAD: variable-length motif and discord discovery in data series". In: *Data Mining and Knowledge Discovery* 34 (2020), pp. 1022–1071 (cit. on p. 35).

[72] Jason Lines and Anthony Bagnall. "Time series classification with ensembles of elastic distance measures". In: *Data Mining and Knowledge Discovery* 29.3 (2014), pp. 565–592 (cit. on p. 47).

[73] Jason Lines, Sarah Taylor, and Anthony Bagnall. "HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles for Time Series Classification". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 1041–1046 (cit. on pp. 9, 47).

[74] Jason Lines, Luke M Davis, Jon Hills, and Anthony Bagnall. "A shapelet transform for time series classification". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 289–297 (cit. on p. 47).

[75] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation Forest". In: *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*. ICDM '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 413–422 (cit. on pp. 29, 113).

[76] Yubao Liu, Xiuwei Chen, and Fei Wang. "Efficient Detection of Discords for Time Series Stream". In: *Advances in Data and Web Management* (2009), pp. 629–634 (cit. on p. 31).

[77] Wei Luo and Marcus Gallagher. "Faster and Parameter-Free Discord Search in Quasi-Periodic Time Series". In: *Proceedings of the 15th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*. PAKDD'11. Shenzhen, China: Springer-Verlag, 2011, 135–148 (cit. on p. 31).

[78] Bartolo Luque, Lucas Lacasa, Fernando Ballesteros, and Jordi Luque. "Horizontal visibility graphs: exact results for random time series." In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 80 4 Pt 2 (2009), p. 046103 (cit. on p. 37).

[79] Haoran Ma, Benyamin Ghojogh, Maria N. Samad, Dongyu Zheng, and Mark Crowley. "Isolation Mondrian Forest for Batch and Online Anomaly Detection". In: *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2020, pp. 3051–3058 (cit. on pp. 8, 30, 99).

[80] Laurens van der Maaten and Geoffrey E. Hinton. "Visualizing High-Dimensional Data Using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605 (cit. on p. 168).

[81] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. "Long Short Term Memory Networks for Anomaly Detection in Time Series". In: (2015) (cit. on p. 42).

[82] Pierre-François Marteau, Saeid Soheily-Khah, and Nicolas Béchet. *Hybrid Isolation Forest - Application to Intrusion Detection*. May 10, 2017. (Visited on 11/05/2020) (cit. on p. 30).

[83] G.B. Moody and R.G. Mark. "The impact of the MIT-BIH Arrhythmia Database". In: *IEEE Engineering in Medicine and Biology Magazine* 20.3 (2001), pp. 45–50 (cit. on pp. 28, 82, 113, 118).

[84] Abdullah Mueen, Eamonn Keogh, and Neal Young. "Logical-shapelets: an expressive primitive for time series classification". In: *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2011, pp. 1154–1162 (cit. on p. 47).

[85] Abdullah Mueen, Yan Zhu, Michael Yeh, Kaveh Kamgar, Krishnamurthy Viswanathan, Chetan Gupta, and Eamonn Keogh. *The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance*. 2017 (cit. on p. 34).

[86] Abdullah Al Mueen, Eamonn J. Keogh, Qiang Zhu, Sydney S. Cash, and M. Brandon Westover. "Exact Discovery of Time Series Motifs". In: *Proceedings of the SIAM International Conference on Data Mining. SIAM International Conference on Data Mining* 2009 (2009), pp. 473–484 (cit. on p. 24).

[87] Gyoung S. Na, Donghyun Kim, and Hwanjo Yu. "DILOF: Effective and Memory Efficient Local Outlier Detection in Data Streams". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '18. London, United Kingdom: Association for Computing Machinery, 2018, 1993–2002 (cit. on p. 30).

[88] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. USA: Omnipress, 2010, pp. 807–814 (cit. on p. 47).

[89] N. H. Packard, J. P. Crutchfield, J. D. Farmer, and R. S. Shaw. "Geometry from a Time Series". In: *Phys. Rev. Lett.* 45 (1980), pp. 712–716 (cit. on p. 38).

[90] Themis Palpanas. "Data Series Management: The Road to Big Sequence Analytics". In: *SIGMOD Rec.* 44.2 (Aug. 2015), 47–52 (cit. on p. 2).

[91] Themis Palpanas and Volker Beckmann. "Report on the First and Second Interdisciplinary Time Series Analysis Workshop (ITISA)". In: *SIGMOD Rec.* 48.3 (Dec. 2019), 36–40 (cit. on pp. 2, 7).

[92] John Paparrizos and Luis Gravano. "Fast and Accurate Time-Series Clustering". In: *ACM Trans. Database Syst.* 42.2 (June 2017) (cit. on pp. 26, 27).

[93] John Paparrizos and Luis Gravano. "K-Shape: Efficient and Accurate Clustering of Time Series". In: *SIGMOD Rec.* 45.1 (June 2016), 69–76 (cit. on pp. 8, 26, 27, 114).

[94] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *NeurIPS*. Vol. 32. 2019 (cit. on p. 132).

[95] Dragoljub Pokrajac, Aleksandar Lazarevic, and Longin Jan Latecki. "Incremental Local Outlier Detection for Data Streams". In: *2007 IEEE Symposium on Computational Intelligence and Data Mining*. 2007, pp. 504–515 (cit. on p. 30).

[96] T. Rakthanmanon, E. J. Keogh, S. Lonardi, and S. Evans. "Time Series Epenthesis: Clustering Time Series Streams Requires Ignoring Some Data". In: *2011 IEEE 11th International Conference on Data Mining*. 2011, pp. 547–556 (cit. on pp. 62, 63).

[97] J. Rissanen. "Modeling by shortest data description". In: *Automatica* 14.5 (1978), pp. 465–471 (cit. on p. 63).

[98] H. Sakoe and S. Chiba. "Dynamic programming algorithm optimization for spoken word recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49 (cit. on p. 25).

[99] Patrick Schäfer. "The BOSS is Concerned with Time Series Classification in the Presence of Noise". In: *Data Mining and Knowledge Discovery* 29.6 (Nov. 2015), 1505–1530 (cit. on p. 47).

[100] Johannes Schneider, Phillip Wenig, and Thorsten Papenbrock. "Distributed detection of sequential anomalies in univariate time series". In: *VLDB Journal* 30.4 (2021), pp. 579–602 (cit. on p. 168).

[101] Bernhard Schölkopf, Robert C Williamson, Alex J. Smola, John Shawe-Taylor, and John C. Platt. "Support Vector Method for Novelty Detection". In: *Advances in Neural Information Processing Systems 12*. Ed. by S. A. Solla, T. K. Leen, and K. Müller. MIT Press, 2000, pp. 582–588 (cit. on p. 40).

[102] David W. Scott. *Multivariate Density Estimation. Theory, Practice, and Visualization.* Wiley, 1992 (cit. on pp. 75, 113).

[103] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, Crystal Chen, and Susan Frankenstein. "GrammarViz 3.0: Interactive Discovery of Variable-Length Time Series Patterns". In: *ACM Trans. Knowl. Discov. Data* 12.1 (Feb. 2018) (cit. on pp. 82, 118).

[104] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, Crystal Chen, and Susan Frankenstein. "Time series anomaly discovery with grammar-based compression". In: *EDBT*. 2015 (cit. on pp. 5, 31–33, 57).

[105] Joan Serrà, Santiago Pascual, and Alexandros Karatzoglou. "Towards a universal neural network encoder for time series". In: *CCIA*. 2018 (cit. on p. 50).

[106] Jin Shieh and Eamonn J. Keogh. "iSAX: disk-aware mining and indexing of massive time series datasets". In: *Data Mining and Knowledge Discovery* 19 (2009), pp. 24–57 (cit. on p. 63).

[107] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. "Online Outlier Detection in Sensor Data Using Non-Parametric Models". In: *Proceedings of the 32nd International Conference on Very Large Data Bases*. VLDB '06. Seoul, Korea: VLDB Endowment, 2006, 187–198 (cit. on pp. 2, 7).

[108] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9 (cit. on p. 50).

[109] Jian Tang, Zhixiang Chen, Ada Wai-Chee Fu, and David Wai-Lok Cheung. "Enhancing Effectiveness of Outlier Detections for Low Density Patterns". In: *PAKDD*. 2002, pp. 535–548 (cit. on p. 30).

[110] P. Tanisaro and G. Heidemann. "Time Series Classification Using Time Warping Invariant Echo State Networks". In: *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2016, pp. 831–836 (cit. on p. 50).

[111] David MJ Tax and Robert PW Duin. "Support vector data description". In: *Machine learning* 54.1 (2004), pp. 45–66 (cit. on p. 40).

[112] Jingyuan Wang, Ze Wang, Jianfeng Li, and Junjie Wu. "Multilevel Wavelet Decomposition Network for Interpretable Time Series Analysis". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '18. London, United Kingdom: Association for Computing Machinery, 2018, 2437–2446 (cit. on p. 9).

[113] Jun Wang, Arvind Balasubramanian, Luis Mojica de La Vega, Jordan R. Green, Ashok Samal, and B. Prabhakaran. "Word Recognition from Continuous Articulatory Movement Time-series Data using Symbolic Representations". In: *SLPAT*. 2013 (cit. on p. 24).

[114] Zhiguang Wang, Weizhong Yan, and Tim Oates. "Time series classification from scratch with deep neural networks: A strong baseline". In: *2017 International Joint Conference on Neural Networks (IJCNN)* (2017), pp. 1578–1585 (cit. on pp. 9, 46, 48–50, 52, 127, 134).

[115] CW Whitney, DJ Gottlieb, S Redline, RG Norman, RR Dodge, E Shahar, S Surovec, and FJ Nieto. "Reliability of scoring respiratory disturbance indices and sleep staging". In: *Sleep* 21.7 (1998), pp. 749–57 (cit. on p. 24).

[116] Frank Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83 (cit. on p. 91).

[117] Qin Wu, Xingqin Qi, Eddie Fuller, and Cun-Quan Zhang. "Follow the Leader: A Centrality Guided Clustering and Its Application to Social Network Analysis". In: *The Scientific World Journal* (2013) (cit. on p. 64).

[118] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. "Empirical Evaluation of Rectified Activations in Convolutional Network". In: *CoRR* abs/1505.00853 (2015) (cit. on p. 47).

[119] Qiang Yang and Xindong Wu. "10 Challenging Problems in Data Mining Research". In: *International Journal of Information Technology and Decision Making (IJITDM)* 05.04 (2006), pp. 597–604 (cit. on pp. 9, 46).

[120] Dragomir Yankov, Eamonn Keogh, and Umaa Rebbapragada. "Disk Aware Discord Discovery: Finding Unusual Time Series in Terabyte Sized Datasets". In: *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. 2007, pp. 381–390 (cit. on pp. 5, 31, 32).

[121] Dragomir Yankov, Eamonn Keogh, and Umaa Rebbapragada. "Disk Aware Discord Discovery: Finding Unusual Time Series in Terabyte Sized Datasets". In: *Seventh IEEE International Conference on Data Mining (ICDM 2007)*. 2007, pp. 381–390 (cit. on p. 31).

[122] Dragomir Yankov, Eamonn Keogh, Jose Medina, Bill Chiu, and Victor Zordan. "Detecting Time Series Motifs under Uniform Scaling". In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '07. New York, NY, USA: Association for Computing Machinery, 2007, 844–853 (cit. on p. 24).

[123] Lexiang Ye and Eamonn Keogh. "Time series shapelets: a new primitive for data mining". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 947–956 (cit. on pp. 47, 169).

[124] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. "Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 1317–1322 (cit. on pp. 2, 5, 7, 8, 31, 33, 35, 57, 84, 99).

[125] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V. Chawla. "A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (2019), pp. 1409–1416 (cit. on p. 9).

[126] Bendong Zhao, Huan zhang Lu, Shangfeng Chen, Junliang Liu, and Dongya Wu. "Convolutional neural networks for time series classification". In: *Journal of Systems Engineering and Electronics* 28 (2017), pp. 162–169 (cit. on pp. 9, 51).

[127] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J. Leon Zhao. "Time Series Classification Using Multi-Channels Deep Convolutional Neural Networks". In: *WAIM*. 2014 (cit. on pp. 9, 50).

[128] Bolei Zhou, A. Khosla, À. Lapedriza, A. Oliva, and A. Torralba. "Learning Deep Features for Discriminative Localization". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2921–2929 (cit. on pp. 9, 52).

[129] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk, and Eamonn Keogh. "Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 739–748 (cit. on pp. 33, 67).

[130] Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Kaveh Kamgar, and Eamonn Keogh. "Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds". In: *2018 IEEE International Conference on Data Mining (ICDM)*. 2018, pp. 837–846 (cit. on p. 35).

[131] Zachary Zimmerman, Kaveh Kamgar, Nader Shakibay Senobari, Brian Crites, Gareth Funning, Philip Brisk, and Eamonn Keogh. "Matrix Profile XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond". In: *Proceedings of the ACM Symposium on Cloud Computing*. SoCC '19. Santa Cruz, CA, USA: Association for Computing Machinery, 2019, 74–86 (cit. on p. 35).

[132] Kostas Zoumpatianos and Themis Palpanas. "Data Series Management: Fulfilling the Need for Big Sequence Analytics". In: *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 2018, pp. 1677–1678 (cit. on p. 2).