POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

MULTIVARIATE SPATIO-TEMPORAL ANOMALY DETECTION IN A MOBILE NETWORK

Relatore: Prof. Pier Luca Lanzi

Tesi di Laurea di:
Gianluca Goffredi, matricola 784244

Anno Accademico 2014-2015

Contents

Abstract

Data mining is becoming increasingly important in a world where data is created and consumed at extraordinary rates. Of the many applications of data mining, anomaly detection is a particularly interesting one that aims at finding unusual patterns or data points, usually generated by rare negative events. In particular, anomaly detection applied to mobile networks can be used to improve the quality of the offered services, with benefits for end-users and service providers alike. The goal of this thesis is to present an effective strategy for spatio-temporal anomaly detection in a mobile network. To do so, we studied the performance of existing state-of-the-art algorithms applied to the mobile network domain. As a solution to the challenges of anomaly detection in this domain, we present a new algorithm, the Multivariate Spatio-Temporal Anomaly Detector using Fisher's method (MuSTF), an extension of the Fast Subset Scan framework and the STCOD algorithm. We applied MuSTF to real-world data collected from a region-wide mobile network over a month, and proved to be an improvement of current state-of-the-art algorithms in detecting drops in the quality of services, especially being able to detect anomalies with very little delay from the moment of their appearance.

Sommario

Le tecniche di data mining stanno assumendo un'importanza sempre più grande in un mondo dove grandi quantità di dati sono create e utilizzate con una velocità straordinaria. Tra le molte applicazioni del data mining una particolarmente interessante è l'anomaly detection, ovvero il riconoscimento di dati inusuali, in genere indicazione di eventi rari, spesso negativi. In particolare, l'anomaly detection applicata alle reti mobili può essere usata per migliorare la qualità dei servizi offerti, con benefici sia per gli utenti sia per le compagnie che forniscono i servizi. Lo scopo di questa tesi è di presentare una strategia efficace per l'anomaly detection spazio-temporale applicata alle reti mobili. Per fare ciò, abbiamo analizzato la performance di algoritmi stato dell'arte nell'anomaly detection applicati a reti mobili. Per affrontare i problemi che sorgono nell'applicare anomaly detection in questo campo, abbiamo sviluppato un nuovo algoritmo, Multivariate Spatio-Temporal Anomaly Detector using Fisher's method (MuSTF), estendendo gli algoritmi pre-esistenti Fast Subset Scan e STCOD. Abbiamo applicato MuSTF a dati di QoS raccolti dalla rete mobile del Piemonte per un mese. MuSTF ha dimostrato di riconoscere cali nella qualità dei servizi con migliore precisione rispetto allo stato dell'arte, in particolare riconoscendo le anomalie con un ritardo particolarmente basso rispetto al loro insorgere.

Ringraziamenti

Vorrei innanzitutto ringraziare i miei genitori e tutta la mia famiglia per il sostegno economico, ma soprattutto morale e affettivo che mi ha sempre mostrato in questi anni di università, dal primo all'ultimo giorno di studio.

Ringrazio tutti i miei amici per i momenti di svago e di sfogo in tutti questi anni. Mi scuso per non essere stato molto presente durante la scrittura di questa tesi!

Ringrazio il Prof. Lanzi per la sua guida e i suoi consigli nello scrivere questo documento, nonchè per le sue utilissime lezioni di Data Mining.

Vorrei infine ringraziare i colleghi di Bip, in particolare Daniele, per avermi seguito e aiutato in questo periodo donandomi il loro tempo prezioso.

List of Figures

Chapter 1

Introduction

Data mining is assuming an increasingly large importance in a world where data is created and consumed at extraordinary rates [20]. As data storage grows exponentially, the task of manually analyzing datasets becomes impossible for any specialist. Techniques for the automated analysis of data are coming under the spotlight, as companies strive to use the data they accumulated over the years to improve their business performance. Of the many applications of data mining, anomaly detection is a particularly interesting one that aims at finding unusual patterns or data points. Depending on the domain of application, anomaly detection can help finding various kinds of rare, usually negative, events: in computer networks, Intrusion Detection Systems are implemented in many security software suites to detect malicious activities by malware or external attackers [24]; in medical science, anomaly detection can be used to detect disease outbreaks, allowing the deployment of early countermeasures [50]; signs of abnormal weather can be used to predict violent precipitation events in localized areas [43]; in industrial machinery, using anomaly detection can be used to predict a forthcoming fault, in order to schedule maintenance more efficiently [37].

The rarity of anomalous events is the biggest obstacle in studying their behavior with traditional data mining techniques. This led to the creation of specialized anomaly detection algorithms.

Mobile networks

In this thesis we study anomaly detection applied to data collected from a mobile telecommunication network. Mobile networks are ubiquitous in today's world and are a critical infrastructure for communication, be it for work or leisure activities. Services based on these networks range from

phone calls, SMS, mails, to Internet browsing, economic transactions, video streaming and much more. It should not come as a surprise that the quality of the mobile network infrastructure is a critical element for companies and users alike. Monitoring the network for drops in the quality of offered services can be beneficial for service provider, that can solve or prevent failures in the network to increase customer retention.s Anomaly detection in a mobile network faces several challenges: mobile networks cover vast areas and operate constantly over the course of time; data collected in such networks is highly dependent on the spatial and temporal dimension. Another challenge is posed by the heterogeneity of the measurements used to monitor the quality of the different services. Furthermore, data in a mobile network is subject to seasonality effects over the course of days, weeks and months. Finally, the anomalies must be found as soon as possible to solve the issue before it impacts negatively on the services provided to the users.

Previous work

The characteristics of anomalies change depending on the domain of application of anomaly detection. As such, a wide range of approaches have been developed for specific applications. In supervised applications, data labeled as normal or anomalous is available. Various classification algorithms have been developed for supervised anomaly detection. Hawkins developed an approach based on a replicator feed-forward neural network [39], trained on normal data, that tries to minimize the error between the output and input data, and considers the error in replicating a new instance as its anomaly score; another approach by Davy [9] makes use of one-class Support Vector Machines to separate normal and anomalous data in a feature space; Mahoney and Chan developed an algorithm based on association rules to find instances that do not satisfy frequent rules; Wong developed a Bayesian network [50] modeling historical information to detect disease outbreaks; the Isolation Forest algorithm by Liu [15] uses random forest to isolate anomalies that appear in shallow branches of the trees.

Since labeling anomalies in a dataset is usually a very costly process, many unsupervised anomaly detection algorithms were developed. Nearest Neighbor techniques, such as Local Outlier Factor [25], consider anomalous instances that are distant from the rest of the data. Other approaches follow a reverse nearest neighbor technique: an example is ODIN [49] that analyzes a directional graph of the neighborhood relationship to compute the number of reverse neighbors instead of the direct ones to find anomalies.

Clustering algorithms find the clusters that best fit the data and iden-

tify as anomalous the points that are not members of any cluster, or whose membership to a cluster is low. Clustering anomaly detection algorithms are similar to standard data mining clustering algorithms: some minor variations include using robust statistics like the medoid instead of the mean [36]. Another approach uses fuzzy versions of k-mean to compute memberships of instances to different clusters [12]; another technique by He [54] called CBLOF is able to differentiate between large clusters of normal data and small clusters of similar anomalies.

Statistical algorithms detect anomalies based on the computation of the likelihood of an instance being generated by a certain distribution. Parzen windows [33] compares the log likelihood of the analyzed instance to the log likelihood of a sample generated by the assumed distribution of the data. Gaussian Mixture Models can be used to detect anomalies in data whose distribution can be represented as the sum of multiple different Gaussian distributions [23]. Several other algorithms based on information theory have been developed: an example is LSA [55], that measures the changes in entropy of a candidate anomalous set of instances while swapping instances in and out. Another algorithm presented by Wang [28] is based on the size of a compressed text before and after the removal of anomalous lines.

Spatio-temporal data is a more recent field of interest in anomaly detection. Temporal anomaly detection has been studied for quite some time, and many algorithms for analysis of time series have been developed. An example is CUSUM [32], an algorithm that compares the cumulative sum of a value over time against a fixed threshold. For network applications, dTrend [46] computes statistics on a graph to evaluate traffic trend and find anomalous behavior in a time interval. Spatial anomaly detection is a less explored application. Chawla proposed a new factor for measuring anomalies in space, SLOM [38] using a nearest neighbor approach over non-spatial data followed by the computation of spatial statistics. Another adaptation of nearest neighbor techniques is AvgDiff [53], that considers the weighted difference in measurements between neighbors, using as weight the spatial vicinity of two measurements.

Other approaches exist to study datasets with both spatial and temporal characteristics. STOUT [51] analyses a dataset by separating spatial and temporal information from measurement data and constructing matrices that describe the spatial and temporal neighborhood of each value. A novel neural network approach has been implemented in the Cortical Learning Algorithm [30], that can memorize spatial and temporal context of an instance using data structures inspired by the connections between neurons in the human brain. STCOD (Spatio-Temporal Correlation-based Outlier

Detector) [27], is an algorithm developed for analysis of a sensor network, that uses the correlation between past and current windows of values and a majority voting strategy between neighbors to distinguish outliers from anomalies. The Fast Subset Scan family of algorithms, developed by Neill [8], are anomaly detectors that exploit a property called Linear Time Subset Scan to search and find in linear time the most anomalous sets of instances in a temporal interval.

Our approach

The goal of this work is to present an effective strategy for spatio-temporal anomaly detection in a mobile network. To achieve this, we first applied state-of-the-art algorithms to data collected from a mobile network; we then developed an algorithm of our own, Multivariate Spatio-Temporal Anomaly Detector using Fisher's method (MuSTF), to provide an improved solution of the problem. MuSTF extends approaches that were successful in other domains to apply them to a mobile network. In particular, it extends the Fast Subset Scan approach with weighting and voting strategies to make it more robust to outliers. MuSTF presents all the characteristics needed to analyze a mobile network: firstly, it analyzes measurements both in spatial and temporal context; secondly, it is able to distinguish seasonal effects from changes in the trend of a feature; thirdly, it can analyze multiple features, even if different in type; finally, it can operate on-line, providing timely results for fast intervention in real-world application.

Structure of the thesis

In chapter 2 we give an overview of the task of anomaly detection and describe state-of-the-art algorithms in the field, with particular attention to algorithm for anomaly detection in spatio-temporal domains.

In chapter 3 we examine the problem of multivariate spatio-temporal detection and the challenges it introduces. We show the performance of current state-of-the-art algorithms when applied to a real-world dataset.

In chapter 4 we present MuSTF, our algorithm that solves the issues of multivariate spatio-temporal anomaly detection in a mobile network. We describe in detail the procedure of the algorithm.

In chapter 5 we provide the pseudo code of MuSTF, discussing some implementation details.

In chapter 6 we show the performance of MuSTF on a real-world dataset, and we compare its performance to the one of state-of-the-art algorithms. We show that MuSTF improves detection time and accuracy with respect

to the existing algorithms. We also give a short summary of possible future improvements to MuSTF.

Chapter 2

State-of-the-Art in Anomaly Detection

In this chapter, we introduce the state-of-the-art of anomaly detection, with particular regard to the techniques related to the domain of our problem. As a first necessary step, we give a definition of the concept of anomaly and the characteristics it can assume. We then proceed to discuss the process of anomaly detection, its applications and the challenges that it faces. An overview of algorithms and strategies follows to provide a reference to the more general approaches currently employed in the field. We then focus our attention to the specialized spatial and temporal algorithms that were designed to work in the domain of our interest. Finally, we discuss the applicability of the presented algorithms to our problem.

2.1 Concept of Anomaly

Before introducing the problem of anomaly detection, it is crucial to give a definition that clarifies what is an anomaly. Unfortunately, there isn't a single well established definition for the concept of anomaly. Many authors consider the concept of anomaly to be the same as that of outlier. A well accepted definition from Hawkins defines an outlier as "an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism"[18]. Other authors instead try to differentiate the two concepts by considering anomalies as outliers with an explanation that can point to an event of interest [44], or outliers that can provide critical, actionable information [48]. We will follow the latter approach, and consider anomalies as outliers that point to a phenomenon of

our interest. In the rest of this work, we call normal data those instances of a dataset that are neither outliers nor anomalies.

2.1.1 Characterization of anomalies

Depending on the application and on the available data, anomalies can assume different forms, each requiring a different approach to be detected. A categorization of anomalies can help in the choice of the algorithm to use.

A first characterization of an anomaly is in the number of attributes that need to be considered:

- Univariate anomaly: when the anomalous behavior is detectable by checking values from a single attribute. Even when multiple attributes show anomalous behavior, as long as the anomaly can be characterized with a single attribute it is still treatable with an univariate model.
- Multivariate anomaly: when the anomalous behavior can only be detected by a combination of values from different attributes.

A second distinction is in the number of instances which show anomalous behavior:

- Point anomalies: individual instances that can be considered anomalies with respect to the rest of the data. This is the simplest kind of anomaly.
- Contextual anomalies: individual instances that can be considered anomalies with respect to a subset of the data which shares a context with them. In this case the attributes in the data are divided in:
  - Contextual attributes: the set of variables that provide only information about the context of the instance, not on its behavior (eg. latitude and longitude in instances with geographical information).
  - Behavioral attributes: the set of variables that provide information on the behavior of the attribute, regardless of context.
- Collective anomalies: group of instances that shows anomalous behavior, when the individual instances are not anomalies by themselves. An example of this particular kind of anomaly can be a pattern of instructions executed by a program, where the individual instructions are normal but the ordering is not.

2.2 Anomaly Detection

Given the definition of anomaly, the anomaly detection process is composed of three successive tasks:

1. Outlier Detection: the process of identifying outliers present in the data.

2. Outlier Description: the process of enriching an outlier with information about its nature.

3. Anomaly Identification: the process of identifying those outliers which are of interest to our problem.

While anomaly detection as a process entails all three activities, the latter two are rarely explored in research. In fact, the information detailing the nature of an outlier and how to interpret that information as interesting or not is heavily dependent on the field of application and the goal that needs to be achieved. For this reason, there is no general solution for outlier description and anomaly identification, which are rarely automated even in an ad hoc way. For example, in fraud detection, algorithms are used to raise an alarm on a suspicious situation, but it's up to the analyst to decide if it is really an anomaly after further investigation. In the following sections we also focus only on the first task, outlier or anomaly detection.

2.2.1 Challenges in anomaly detection

Anomaly detection is a complex problem in data mining with several distinguishing characteristics from the standard data mining problems.

The first challenge encountered in anomaly detection is conceptual: it is very hard to define a clear boundary between normal behavior and an anomalous one. Even when considering a simple definition of outlierness such as the low probability of occurrence of an event, the threshold to use to consider a probability low is application dependent. This problem becomes even more complicated when we try to define a boundary between outliers and anomalies we are interested in. The ambiguity of the concept of outliers and anomalies is one of the main issues when trying to make general algorithms.

A second challenge is the massive disproportion in the amount of anomalies available with respect to normal data. Using common data mining models with both normal and anomalous data would be highly skewed in favor of

normal data, while using anomalies in high proportion would distort the results and cause overfit. Furthermore, obtaining correctly labeled anomalies requires manual intervention, thus being a very expensive procedure.

Another challenge is presented by the dynamic nature of anomalies, which may change over time. This can be due to several factors: in some application fields, such as network attacks, anomalies are crafted by an attacker which is interested in hindering the detection as much as possible; in this situation, the attacker can try to hide the anomaly in such a way that it appears similar to normal behavior. Even without malicious intent, any change in the application assumption or new kind of anomaly can prevent us to use past behavior to estimate current one. These kind of previously unknown anomalies are called novelties [48].

Another characteristic of anomalies is that they usually impact negatively on the functioning of a system. Balancing false positives while avoiding to misclassify true anomalies is critical. This element requires anomaly detection algorithms to be evaluated with metrics such as precision and recall rather that accuracy [4].

2.2.2 Applications of anomaly detection

Anomaly detection is widely used in many scenarios for different disciplines. Here are some cases of use:

- Intrusion Detection: one of the most common uses of anomaly detection is in network security. IDS use anomaly detection algorithms to detect suspicious behavior on-line. This can be either host-based, detecting anomalous patterns in system calls, or network-based, for remote breaches in security. In particular, misuse detection deals with recognizing previously known attacks, while novelty detection deals with recognizing new unknown attacks.

- Fraud Detection: refers to systems employed by commercial organizations such as banks, insurance companies, etc, to find out fraud attempts. A typical usage consists in comparing profiles of user common behavior to detect identity thefts.

- Medical Science: there are many examples of anomaly detection in the health-care field. Malfunctioning of medical instrumentation can be detected by finding anomalous readings. Outbreak of a disease can also be predicted from patient data.

- Industrial: wear is a common problem in heavily industrialized companies. Anomaly detection can be used to use historical behavior to predict faults ahead of time, avoiding economic losses and safety breaches.

- Security: anomaly detection algorithms have been applied to analysis of video data from security camera to detect anomalous behavior in crowd movements.

- Military: several algorithms have been used in military application, such as discovery and plotting of enemy route and targeting systems.

- Text Analysis: with the rise of social networks, many companies are interested in studying change in trends and anomalous behavior of users for commercial purposes.

2.2.3 Categories of anomaly detection algorithms

As the panorama of applications is quite complex, many different algorithms were developed.

A first major distinction between algorithms is the requirement of label for its functioning.

- Supervised algorithms: require training data where each instance is labeled as normal or anomalous. This kind of techniques is rarely used because the need of labeled anomalous instance is a huge barrier, and it is not suited to capture novelties. Furthermore, if labeled instances are present this problem is not particularly different from standard data mining. We will not examine this category of anomaly detection in this document.

- Semi-supervised algorithms: require training data labeled as normal. The need of normal data is still a requirement not met by many applications, but they are still used in situation where modeling the normal data can be used to detect anomalies.

- Unsupervised algorithms: algorithms that do not require data labeled as normal or anomalous. These techniques can be applied in most situations.

Another difference between algorithms is the kind of output they generate:

- Labeling algorithms: output a binary label that classifies a tested instance as an anomaly or not, or a set of retrieved outliers.

- Scoring algorithms: compute an outlierness or anomaly score, which is a numerical value, and flag as anomalies the data with the highest scores. The computation is usually done over all training instances or the most likely candidate for outliers.

The labeling approach gives a crisp result and doesn't need external analysis and comparison between instances, however it doesn't give much information on the nature of outliers. For this reason scoring algorithms are usually preferred and the most common in literature.

The biggest element that drives the decision of which algorithm to use in an application is the ability to detect anomalies in the domain of interest. This includes the kind of data attributes (numerical, categorical, etc) that the algorithm needs to process, but also the structure of the data (high dimensionality, network based, etc) and the kind of anomalies we expect to find.

2.3 Overview of Anomaly Detection Algorithms

To better understand the variety of approaches applied on the field, the following algorithms are grouped in six major categories, depending on the main strategy adopted. Note that this categorization is only for clarity of exposition, since hybrid algorithms exist using concepts from many categories at once.

2.3.1 Classification algorithms

Classification algorithms are based on the construction of a classifier model using available normal data. After deriving the model, it is applied to new instances to check how much they are represented from the learned model. If the new instance is badly represented by the model, it is assumed that it was generated by a different process, and it is flagged as an anomaly.

Classification based algorithms are mostly semi-supervised, which makes them hard to apply in many situations. Furthermore, they are very sensitive to changes in the normal behavior of the system that would require relearning the model. Nevertheless, the learned model can be a useful instrument for outlier description, and is usually fast, making these techniques useful when applicable.

Replicator Neural Network

Hawkins presented a semi-supervised classification based algorithm based on a neural network [39]. The RNN is a feed-forward multi-layered perceptron network, which tries to replicate the input data as the output of the network. The weights of the RNN are updated at every training instance, trying to minimize the reconstruction error over all the components of the output. The network is trained on a dataset containing normal instances, thus representing a compact model of the normal behavior of the system. To test data for anomaly detection, instances are simply inputted in the network. The average reconstruction error over all variables is used as the outlierness score for each instance. The biggest shortcoming of the RNN is its inability to treat categorical data without transforming it in multiple binary attributes, which still causes a raise in computational complexity.

One-class Support Vector Machines

Support Vector Machines are a widely used method to find anomalies in feature space [9]. Instead of dividing instances by class like traditional SVM algorithms, one-class SVM consider all instances as members of the normal class, and find the boundary that best fits the given data, dividing it from the origin. After the training is finished, the testing simply involves classifying an instance as normal or anomalous depending on which side of the found boundary they are mapped. SVM assume, like most semi-supervised algorithms, that the training data is composed only of normal data, however robust variants of SVM exist that can work with training data containing anomalies, thus making robust one-class SVM unsupervised algorithms. SVM algorithms are very accurate in their result, but they are also computationally intensive, especially in their robust implementation.

LERAD

Mahoney and Chan developed an algorithm using rules to detect novelties on-line [24]. The algorithm randomly couples instances from the training set and generates rules of the form $[if\, A_1 = 10 and A_2 = 20 ... then A_3 = 100]$. For each generated rule, every possible value for the consequent is memorized. During testing phase, for each instance, rule whose antecedent matches the instance are searched; if they exist, the algorithm checks if the value of the consequent in the new instance was already memorized in those rules. If not, the new instance is flagged as an anomaly. To avoid overfitting and reduce false positives, poor quality rules are pruned after training phase,

first keeping only a subset of rules that cover the training sample, then by performing a validation test with a different training set of normal data to remove rules that generate false positives. Rules provide a very user friendly tool to understand the nature of an anomaly. The downside of LERAD is that it is not suited to attributes with many possible values, which makes learning rules hard. This limits the application of LERAD only to some categorical domain.

WSARE

Wong presented another algorithm based on networks [50]. A Bayesian network is built using historical data related to the problem and its environment. The Bayesian network represents the baseline distribution we expect from the data. During test phase, we generate rules of one or two components from the current data, then compare the rules with the baseline distribution using Fisher's Exact Test, which is used to test the likelihood two items were generated by the same distribution. A high outlierness score is assigned to rules that do not match the distribution found in the baseline. WSARE requires historical data from several periods of time before being deployed, however if that data is available is a very accurate algorithm, and its rules are helpful in outlier description.

iForest

This algorithm proposed by Liu [15] exploits the property of anomalies of being isolated from the rest of the data to identify them by randomly partitioning data using binary decision trees. Because of the aforementioned property, anomalies tend to be isolated in leaf much sooner than normal data, thus allowing to stop the partitioning before it processes most of the data. The process of partitioning is repeated many times, then for each instance the average shortest path from the root is calculated and used as the outlier factor for that instance. iForest doesn't need huge amounts of data to work, therefore subsampling is used for each tree to avoid swamping (false positives due to closeness of normal and anomaly) and masking (anomaly hiding other anomalies).

2.3.2 Nearest Neighbor algorithms

Nearest Neighbor approaches are widely popular in anomaly detection. For each instance a measure is computed considering the neighborhood of the datum. In particular, two kinds of measure can be identified: distance from

its neighbors and density of the neighborhood. The biggest advantage of nearest neighbor techniques is being able to work unsupervised and being able to detect local outliers. However, the discovery of nearest neighbors can be an intensive task and is usually quadratic.

Nearest neighbor techniques assume that distance is a meaningful measure of similarity between two instances. This requires the selection of an appropriate distance measurement for the application; furthermore, even with an appropriate distance chosen, if data is too sparse (e.g. in high dimensional spaces) distance and density tend to lose significance.

Before introducing each algorithm, two common concepts shared by most techniques are presented:

- *k-neighborhood*(p): set of the k instances that are the nearest to a point p
- *k-distance*(p): distance from point p to the kth-neighbor in order of distance

Local Outlier Factor

The LOF was introduced by Breunig et al. [25] as a measure of the difference in density between the instance and its local neighborhood. Consider the computation of the LOF for a point p. The algorithm computes the reachable distance to each neighbor as:

$$\textit{reach-dist}_k(p,o) = max(\textit{k-distance}(o), d(p,o))$$

Then, compute the local reachability density:

$$lrd_k(p) = 1/(\frac{\sum_{o \in \textit{k-neighborhood}(p)} \textit{reach-dist}_k(p,o)}{|\textit{k-neighborhood}(p)|})$$

Finally, the LOF is the average of the ratio between the LRD of the neighbors' and the instance:

$$LOF_k(p) = \frac{\sum_{o \in \textit{k-neighborhood}(p)} \frac{lrd_k(o)}{lrd_k(p)}}{|\textit{k-neighborhood}(p)|}$$

LOF is a very common density-based metric and the scoring factor used by many anomaly detection algorithms.

LOCI

LOCI [40] is a nearest neighbor algorithm based on a metric called Multi-granularity Deviation Factor (MDEF). MDEF is the relative deviation of its local neighborhood density from the average local neighborhood density in a r radius. The lower bound for MDEF is 0, meaning that the density around the instance is the same of the neighbors. When MDEF is higher than three standard deviations, the instance is flagged as an anomaly. In addition to the anomaly score, LOCI is able to compute a LOCI plot for each oulier, providing a visual description of the area surrounding it and the deviation of values in the neighborhood.

ODIN

ODIN [49] is an example of a reverse nearest neighbor technique. ODIN creates a directed graph with instances as nodes and edges connecting each node to its k-neighbors. The number of nodes that have an instance as a k-neighbor is the inverse of its outlierness score. This algorithm exploit the asymmetricality of the k-neighbor relation to find which points are distant from others without calculating the actual distance. Thanks to this ODIN is an algorithm with low computational complexity. The downside of using a count as the measure of outlierness is that it requires a rather large value for k and a big dataset, otherwise the count would not be significantly different between normal and anomalous data.

ORCA

ORCA is a successful distance based algorithm introduced by Bay [41]. Each instance is considered in random order. LOCI keeps track of the nearest neighbors for each point. When an instance is evaluated and its anomaly score is found lower than a cutoff (the value of the lowest anomaly thus far), every other instance that has the currently evaluated element as close neighbor is removed from the dataset, because it can't be an anomaly. The more data is processed the more outliers are found, and the cutoff becomes higher, thus pruning is more efficient. ORCA is able to overcome the computational complexity limitation of nearest neighbor algorithms, and is near-linear. However, the pruning strategy of ORCA is only effective if the algorithm can find anomalies and update its cutoff value. If the dataset contains only a few anomalies ORCA is not able to work efficiently and is near-quadratic.

PINN

An algorithm using concepts from spectral analysis to reduce the computational complexity of searching for the k-neighbors [45]. The data is projected with a random projection to a reduced dimensional space. For each projected instance, its neighbors in the projected space are computed. These are not considered directly neighbors of the instance, but used as candidates to prune other points. The true k-neighbors are searched in the original space only within the candidate neighbors. Afterwards LOF or another nearest neighbor metric can be used to score instances.

2.3.3 Clustering algorithms

Clustering algorithms assume that normal data is grouped in clusters while anomalous points are isolated. These algorithms consist in applying a cluster algorithm and searching for outliers in points which were not assigned a cluster or that show low membership to their cluster. Some of these algorithms can also recognize correctly the presence of clusters of anomalies.

Clustering techniques are similar to nearest neighbor techniques, sharing some measurements like LOF and distance and density based concepts, but instead of evaluating instances with respect to the local neighborhood they evaluate using the assigned cluster.

CBLOF

Cluster-based LOF [54] is a scoring metric derived from LOF used after applying a clustering algorithm. Depending on a user defined threshold, clusters are considered small or large. For example, one may consider cluster containing at least 90% of the data as large if only one cluster of normal data is expected. The CBLOF is calculated for each instance as the size of its assigned cluster multiplied to the distance to the cluster centroid (if the cluster is large) or the distance to the nearest large cluster (if the cluster is small). The distinction between small and large clusters allows CBLOF to distinguish clusters of anomalies, however knowledge of the domain is required to decide the correct value for the threshold between small and large.

k-medoid and SVM

k-medoid [36] is a variant of k-means that uses as centroid for each cluster the instance nearest to the mean point instead of the mean point itself. Thanks to the robustness of k-medoid, this algorithm is highly accurate

and its resulting normal cluster can be used to train a classifier for fast recognition, for example a one-class SVM.

Fuzzy c-means

A variant of classic clustering algorithm k-means, instead of crisp membership fuzzy c-means adopts fuzzy values for the membership of an instance to the clusters [12]. Membership to a cluster is inversely proportional to the distance to the cluster centroid, which is also a measure of outlierness. The main issue of FCM is its slow convergence. Depending on the starting position of the centroids the algorithm may take a long time to reach an optimum. To improve FCM performance some variants were developed. We present some of them [22]:

- SPFCM: Instead of applying fuzzy c-means on the whole dataset, it partitions it and applies it separately on each subset, then uses the average of the partial result to approximate the global one.

- rseFCM: Computes the cluster centroid only on a random sample of the data.

- GOFCM: Variant of SPFCM, uses samples that progressively grow in size, so that the algorithm uses big amounts of data only if needed.

- MSERFCM: Variant of rseFCM, uses a random sample of the data to estimate the starting point for the centroid, to maintain rseFCM speedup and the original accuracy.

2.3.4 Statistical algorithms

Statistical anomaly detection is based on measuring the likelihood that an instance was generated by a certain distribution. Two main approaches can be identified in this category: the distribution can be assumed or it can be derived. If a distribution is already assumed or given statistical algorithm are efficient and can give mathematically sound results. However real data can always be represented by a traditional distribution. Derivation of a distribution can be done to avoid relying on weak assumptions, however the computation becomes intensive. These algorithms show their limitation when processing high dimensional data, where computing a distribution is computationally expensive, and assumptions rarely hold.

Maximum Likelihood Estimation

MLE assumes a distribution for the data, and calculates the likelihood that an instance was generated by that distribution. Various statistical tests like Mahalanobis distance can be used to test the outlierness of an instance. If we don't want to assume a distribution, we can use regression to estimate the model and its parameters. Robust regression in particular can be used even in presence of anomalies in the training data [34].

Parzen windows

Parzen windows is a semi-supervised statistical algorithm [33]. A kernel function is used to approximate a distribution from data. The algorithm extracts a sample from a dataset of normal instances and we check the probability that the log likelihood of our tested instance is inferior of the log likelihood of the sample [10]. If the computed probability is lower than a threshold (represents the false alarm rate that we want to accept) then the instance is classified as an anomaly. This method can be slow, but doesn't require training time if historical data exist.

SmartSifter

A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of many different Gaussian component densities. The assumption is that the normal data is composed by a mixture of different Gaussian distributions. An example of this approach is SmartSifter, an on-line unsupervised algorithm based on GMM [23]. The algorithm uses a variant of the Expectation Maximization algorithm [2] to estimate the GMM and update its parameters at every new instance, discounting the effect of past estimation. The difference introduced in the model by each update is used as anomaly score.

Fuzzy Gaussian Mixture Model

Two algorithms based on GMM and fuzzy concepts were proposed by Ju and Liu [56]. The main idea is to use Fuzzy c-means on different mixture models instead of the full data with complex distribution. One of the following dissimilarity measures is used for the FCM:

- Probability based FGMM: applies FCM using a dissimilarity function composed by both distance and mixture weights. After clustering, the mixture weights are updated using the distance from the centroids.

- Distance based FGMM:another dissimilarity function which focuses only on the distances between points and their component centers.

2.3.5 Information Theoretical algorithms

Information Theoretical algorithms measures outlierness of a point as the complexity that it introduces in the description of the data. Several measures are available, the most common of them being entropy. These algorithms are based on the hypothesis that outliers alter significantly the information content of a dataset by adding elements not easily describable within the underlying structure. Information theory measures are a good alternative to statistics to measure the fit of an instance to the rest of the dataset without having to assume a distribution. The choice of the measure is however critical: it must be sensitive enough to detect the change in complexity induced by very few anomalies in very large datasets.

Local Search Algorithm

LSA is a fast information theoretical algorithm developed by He et al. [55]. The algorithm keeps track of a set of k candidate outliers, initialized randomly. Every instance outside the candidate outlier set is exchanged with every candidate outlier and for each exchange LSA calculates the change in entropy of the dataset. If the entropy of the dataset decreased in some of the exchanges, the instance originally outside the candidate outlier set is inserted in it permanently and the candidate for which the best possible decrease in entropy was computed is put in the normal data. The algorithm goes on until every non outlier has been tested. The algorithm is near linear in time with respect to data size and dimensionality, but it may not reach a global optimum depending on the initial candidates chosen.

Lossless Compression

Lossless compression estimates the complexity of a dataset as inverse to the size it can be compressed with [28]. The algorithm use a grammar-based compression on the whole dataset to determine the baseline. Then for each instance the compression size of the set obtained removing it is computed. The difference, absolute or relative, between the overall compressed size or the leave-one-out compressed size is used as the measure of complexity for that instance. While this algorithm can achieve accurate results, it requires the dataset to have an underlying structure suited to compression, which limits its application to fields such as text analysis.

2.3.6 Spectral Analysis algorithms

Spectral analysis is used to project data in a lower dimensional space that shows more evidently anomalous behavior. It is mostly used as a preprocessing technique before applying computationally intensive algorithms. In some cases it is also used to enable visual detection of anomalies. An issue with spectral analysis is the problem of proper representation of the characteristics of the data: sometimes the reduced dimension space can represent normal data very well, but not anomalies. The transformation adopted needs to be one that conserves outlierness.

PCA

Principal Component Analysis is usually used to find the components with highest variance as a preprocessing technique. In anomaly detection, one possible approach is to use the components with the lowest variance, which are the most sensitive to variations caused by outliers. Another variant, PCC, uses Robust PCA to find both the highest and lowest variance components and uses a threshold for each component to detect anomalies at a given false alarm rate [26]. An issue of PCA based techniques is that multivariate anomalies which can only be identified by checking many attributes may not be evident only on the principal components.

SVD

Singular Value Decomposition is another technique that derives the principal components from the algebraic decomposition of the matrix containing the data. An example of anomaly detection using SVD by Terrell et al. [21] uses the principal components with the least significance, that are the most significant to multivariate anomalies. A measure of sum of squared error on these components is used as a test for anomaly. The SVD provides very accurate results but is computationally expensive with respect to data size and dimensionality.

2.4 Anomaly Detection in Spatial and Temporal Contexts

As described in Section 2.1.1 contextual anomalies are instances that cannot be considered anomalies with respect to the entirety of the data but only in their context. Some of the algorithms presented in the previous section can detect contextual anomalies as well as point anomalies, for example most

LOF based algorithms can detect anomalous behavior with respect to the neighborhood of the data point. However, in these algorithm the concept of context is not well defined and is therefore generalized as the distance computed from all attributes.

In the following sections we describe algorithm that better capture contextual anomalies by separating contextual and behavioral information. In particular, we examine the cases where context is spatial, temporal, and finally both spatial and temporal.

2.4.1 Spatial anomaly detection

Spatial anomaly is one of the most common examples of contextual anomalies. The contextual attributes are generally two or three, usually longitude, latitude and altitude, or analogue spatial coordinates. Two laws of geography can help to understand some of the issues in handling spatial data: first, according to Tobler's First Law of Geography, "everything is related with everything else, but near things are more related than distant things "[47]. A second principle by Harvey states that geographical variables exhibit spatial heterogeneity or heteroscedasticity, that is, data in different spatial areas may have different distribution and variance [17]. For the problem of anomaly detection, these laws imply that global approaches are not suited, and local approaches are preferred.

SLOM

Chawla and Sun [38] propose a new outlier factor for spatial anomaly detection. The algorithm computes nearest neighbors considering the spatial contextual attributes of data.

A đ(p,o) dissimilarity function is computed using only behavioral attributes: the dissimilarity defines a non-spatial distance between an object p and each of its o neighbors. Then the average đ(p,o) is computed, excluding from the count of the average the neighbor with the highest distance to avoid outliers to raise the scores of neighbors. A second measure β is computed counting the number neighbors for which the đ(p,o) is more than the average minus the ones for which it is less than the average. SLOM is the product of these two measures. SLOM avoids the problem of outliers contaminating nearby normal data, however if a normal point is near multiple outliers their effect will still be considered. SLOM is therefore not optimal in applications where anomalies form clusters of their own.

AvgDiff

AvgDiff was introduced by Kou et al. [53] to further adapt nearest neighbor techniques to spatial data. AvgDiff assigns different weights to different neighbors. The weights are determined by spatial relationships such as distance and common border length. The anomaly score is obtained summing the weighted difference between behavioral attributes values of neighbor y and the instance x.

Mean and Median Algorithms

This algorithm [6] defines an attribute function that maps all non spatial attributes to a single value. For each spatial point, after computing its k nearest neighbors, the algorithm computes the neighborhood score as the average of the normalized attribute function over the neighbors. Then a comparison function is calculated as the difference between the neighborhood score and the normalized attribute function result for the instance. Finally compute the Mahalanobis distance between the comparison function and the mean value of the comparison function in the neighborhood. If the distance is over a defined threshold, the algorithms signals an anomaly. A variant of this algorithm is also proposed that uses the median of the comparison function instead of the mean to be robust to outliers during estimation of the center of the comparison function.

2.4.2 Temporal anomaly detection

Temporal context is usually represented by a single contextual attribute indicating the timestamp or the ordering of events. As with spatial anomalies, in temporal context elements near each other are assumed to have similar behavior. Furthermore, instances in time series are ordered and can have a cause-effect relationship. One of the issues of temporal anomaly detection is that the assumption of staticity of the model does not hold. In other words, the system may change behavior over time without being anomalous. This particular case of temporal detection is called trend analysis or concept drift [4].

CUSUM

CUSUM is a statistical analysis algorithm which monitors trend change [32]. CUSUM maintains a cumulative value that updates at every time slot, adding the current value of the controlled instance minus a weight, usually the mean or likelihood of the instance. If the cumulative difference within a

window of time exceeds a value, CUSUM raises an alarm. CUSUM is only able to detect univariate anomalies. It is however a very simple and fast algorithm for detection of trend change.

Contextual Collective Anomaly Framework

A framework for the detection of anomalies in distributed streams was proposed by Jiang et al. [52]. The framework is divided in three phases:

1. Dispatching: receives observations and shuffles them in different downstreams.

2. Scoring: scores instances. It is further subdivided in two different phases:

   (a) Snapshot scorer: calculates anomaly score of an instance based on current information (assuming normal distribution, variance and leave-one-out variance are used). A score for the snapshot of the stream is also computed as the sum of individual instances scores in that time slot.

   (b) Stream scorer: incorporates temporal information into scores by summing the current snapshot score with historical snapshot score. The historical information is updated at every time slot using the current score and discounting the previous one.

3. Alert: if the current score falls outside a window calculated with the median of the dataset and the minimum anomaly score registered, the stream is flagged as an anomaly.

dTrend

A statistical algorithm suited to detect anomalies in a network domain [46]. dTrend calculates probabilistic network statistics for each time slot. The algorithm learns a linear segmentation on these statistics, which is used to compute residuals for the tested instances. We can then choose a threshold to decide how much can normal data deviate from the assumed distribution: for example if we assume that the statistics follow a normal distribution, we can set a threshold for detection using a p-value of 0.05. The choice of the statistic used is critical for the computational complexity of dTrend. Note that the statistics for dTrend are based on network graphs which may not be a valid form of representation for many applications.

mWMA

mWMA is a framework proposed by Ciocarlie et al. [16] for anomaly detection in a cellular network. The hypothesis of the authors is that because of the varied nature of the variables measured in a mobile network, no single method for time series analysis can be used to detect anomalies. Instead the authors propose to use an ensemble of methods to evaluate different anomalies, and weight the methods according to weights. Weights are based on the precision of a method, according to expert knowledge of the problem, and the age of the method computation. After weighing methods, an anomaly is signaled according to the result of a majority vote of all the methods.

2.4.3 Spatio-temporal anomaly detection

When both spatial and temporal contexts are present, both need to be taken into consideration simultaneously. Using the techniques discussed previously can still yield some results, however they might miss complex anomalous situation. The following algorithms are suited to find these complex cases.

STOUT

STOUT [51] is based on tensors, a generalization of vectors. A three-order tensor with spatial, temporal and measurement dimensions is created from the dataset. The algorithm identifies a base tensor to use as a benchmark for normality using as elements the average values from a few sample bins. Before searching for outliers the algorithm enriches the tensor by multiplying it with the spatial and temporal contiguity matrices. These matrices contain respectively the spatial and temporal relationship between each two instances of the original data. This relationship is considered binary and the values of contiguity are either 0 or 1 depending on whether the two instances are within a certain distance of space or time between each other. Next, the algorithm uses CANDECOMP/PARAFAC decomposition (which decomposes the tensor into a sum of vectors) to obtain six vector containing the space, time and measurement values, one each for the baseline and the current tensor. The difference of vectors indicate the regions where it is possible to find outliers, reducing the search space. Finally STOUT perform outlier detection in the narrowed space using neighborhood dynamicity (ratio of measured distance between elements in the narrowed space and outside the space). Where the value exceeds a threshold, the cell is an anomaly. The information of the contiguity matrices manages to add spatial and temporal context to the data without excessive computations.

Cortical Learning Algorithm

Implementation of the Hierarchical Temporal Memory framework by Numenta [30], a framework based on the functioning of the neocortex in the human brain. Layers of neural networks represent information at various granularity. CLA is based on three key concepts:

- : Sparse Distributed Representation: the information is codified using only a small portion of the available pool of cells.
- : Context Recognition: Cells are grouped in columns that are trained to represent the same input. Depending on the context different cells in the columns are activated.
- Memory: Synapses between cells form and get reinforced to keep memory of previous patterns of activation and predict the next step.

The prediction method is also used as an anomaly detector: when an element of a pattern is recognized, CLA predicts the cells that should activate in the next step. If the successive element is not recognized, instead of the predicted cells the whole column is activated because no prediction is available. The activation of a whole column of cells indicates a novelty.

HTM is composed by two sections named Spatial Pooler and Temporal Pooler, the first of which models similar inputs to similar sets of columns, and the latter captures context and sequential patterns in the data. The framework and its implementations are therefore well suited to deal with contextual spatio-temporal anomalies. In addition, to reduce false positives, implementations of CLA keep track of the average error rate and consider if an event is consistent with that or significantly different [31].

STCOD

STCOD is an algorithm developed by Wang and Wu [27] for detection of outliers in sensor data. The main innovation of this algorithm is the ability to distinguish between outliers generated by faulty readings and interesting events. First, a self-detection phase is performed by every node. The similarity between the current window of time and a previous one is computed. If a node finds low similarity, it starts a collective detection phase. Each neighbor of the candidate anomaly node checks if their previous window was similar to the previous window of the candidate. If not, the neighbor votes 0, and if yes, the neighbor also performs a check on the similarity between the current windows: if it finds them similar it votes +1, if not it votes -1. After collecting all the answers, the outlier candidate is flagged as defective

if the vote yields a negative result, while if the result is positive a collective anomaly is signaled. While the algorithm presented is intended to be used in a sensor network, it can be easily adapted to any spatial context by weighting the neighbors vote using Euclidean distance. The algorithm is as such a contextual collective spatio-temporal anomaly detector. The downside of this algorithm is that while it scores anomalies with respect to temporal context (in the self-detection phase), it only labels them without providing a measure of outlierness within the spatial context.

Outstretch

An algorithm proposed by Wu [14]. The algorithm begins finding the top k spatial region with high discrepancy from their neighborhood, using a variant of the Exact-Grid algorithm [3]. If two high-discrepancy regions overlap on more than a user defined percentage of area, only the region with the highest discrepancy is chosen. Afterwards the proper Outstretch algorithm is applied to check for temporal anomalies: the algorithm verifies if an anomalous region at current time falls within a stretched area around an anomaly of the previous time period. A tree is built using these results, with sequences of anomalies over the time represented with a child-father relationship between nodes. Note that an anomaly can fall in the stretched areas of multiple anomalies and therefore have multiple fathers. Outstretch (with Exact-Grid-k) has near quadratic complexity due to the high computational cost of finding regions.

Fast Subset Scan

Fast Subset Scan is an algorithm developed by Neill et al [29], for spatio-temporal discovery of disease outbreaks. The basic FSS procedure is to find the most anomalous groups of contiguous locations within the same time frame. To find these groups, for each examined time period FSS computes a value called priority for each location, that describes the distance of the location's behavior from its expected one. The priority is not used directly as anomaly score, instead it is used to order locations from the most likely to be anomalous to the least likely. The main innovation of FSS lies in the Linear Time Subset Scanning property, thanks to which it is possible to find the most anomalous subsets of location in linear time by examining the priorities of nodes in order from the highest to the lowest.

The first FSS algorithm was univariate and suited to gaussian or poissonian counts of data. Since then Neill developed several extensions of FSS: in [8] a multivariate extension of the FSS algorithm can analyze multiple

streams of data related to the same location; [13] presents Fast General Subset Scan, a variant of FSS that does not assume any specific distribution and operates on categorical data.

Chapter 3

Analysis of Multivariate Spatio-Temporal Anomaly Detection

In this chapter, we describe the goals of our research. We first explain in detail why these goals are relevant to real-world applications. We present the issues and difficulties related to achieving our goals. We then show the performance of state-of-the-art algorithms on a real-world dataset. We give a short analysis of the results of each algorithm and their adequacy for our goals.

3.1 Research objectives

The aim of this work is to develop an algorithm or framework for anomaly detection in the multivariate spatio-temporal domain. As a reference, we remind the most important characteristics of this domain:

- Multivariate: the anomalies may be evident only when analyzing multiple attributes of the dataset, rather than a single one;
- Spatio-temporal: the anomalies may be distributed in both spatial and temporal dimensions;
- Seasonal: the time series exhibit periodic behavior.

Other than detecting anomalies in the environment described above, our goal is also to develop an algorithm that has the following properties:

- On-line: the anomalies must be detected as they appear, as soon as possible;
- Actionable: the anomaly detector must report enough data concerning the anomaly to be useful for identifying and correcting the problem;
- Novelty detection: the anomaly detector must be able to recognize anomalies that did not occur in the past;
- Scalable: the anomaly detector must be able to work on large quantities of data and terminate in reasonable time;
- Distributed: the anomaly detector must be ready to operate in a distributed environment.

A more detailed description of each of these properties follows; we also describe the difficulties that these properties add to the task of anomaly detection, and why these properties are of interest in real-world applications.

3.1.1 Multivariate analysis

Dataset composed of many correlated attributes or features are common in real world application. Traditional time series anomaly detection algorithms such as CUSUM or STCOD analyze only one feature of the dataset. While many anomalies may be detected by analyzing a single feature, other anomalies might affect multiple features without being evident in a single one. In this situation the analysis must be performed on multiple features at once. Univariate analysis also does not account for the possibility of features being correlated to each other; this information could be used, for example, to predict the behavior of a feature by analyzing other correlated features.

Extending a univariate analysis to a multivariate one can be a complex problem. Firstly, each additional feature increases the execution time of the algorithm, and in general the increase is not linear with respect to the number of added features: this is the so called "curse of dimensionality" [5]. It is therefore necessary to extend univariate approaches while keeping low time complexity, if possible linear or constant with respect to the number of features. Secondly, multivariate data is often heterogeneous; when types of features are different the analysis needs to rely on techniques that are general enough to be applicable to every data type involved. Other than the data type, different features may be differently distributed, or have different semantic (for example only positive are meaningful, only integer etc.).

In anomaly detection we also have the additional problem of identifying what is an anomaly in the domain: this is especially hard when dealing

with multivariate data, because characterizing anomalies involving multiple features may not be intuitive and in most cases visualization approaches can't be used with high dimensionality.

Finally, in multivariate data some features may not be available as frequently as others due to coming from different sources, leading to missing data concentrated in some features. The strategies for dealing with missing data in multivariate sets need to account for correlation between features.

3.1.2 Spatio-temporal analysis

We recap here some characteristics of spatio-temporal anomaly detection. The contextual nature of spatio-temporal anomalies is the most prominent characteristic of this domain of anomaly detection. When analyzing spatio-temporal data, we need to consider each value in comparison to its spatial and temporal neighborhood. Algorithms dealing with spatio-temporal datasets are well suited to parallelization since data is already spatially distributed.

3.1.3 Seasonality

Seasonal time series present an almost periodic behavior. This adds another layer of context on top of spatio-temporality, since a value may be normal or anomalous depending on the period of time it occurs. Seasonality can be dealt with in different manners. It is possible to remove the seasonal component of the data, a process called deseasonalization, and perform the analysis only on the underlying trend. Another approach is to compare to each other values related to the same period. In this work we use the method of simple average to memorize for each period the mean and standard deviation of the features and other parameters. A possible issue arising with seasonality is that the periodicity of the time series may not be known beforehand, or the series might be affected by unknown periodic phenomena on top of known ones.

3.1.4 On-line

Many anomaly detectors operate on datasets that are completely available at the start of the analysis, and cannot update their results when more data arrives without executing the whole algorithm anew. This is the case, for example, of most techniques based on nearest neighbor or STOUT. There are many situation in which this kind of analysis is not enough because we need to process streams of data and we need to analyze new values as soon as

they arrive. On-line algorithms analyze streaming data at regular intervals, usually at each new arrival of data, therefore are subject to strict limitations in time complexity. Each new value should be processed approximately in the same time to ensure that the execution of the algorithm terminates before the next arrival of data.

3.1.5 Actionable

Knowing that an anomaly occurred is usually not enough information for most applications. Providing additional information about the anomaly is required to intervene, however it is not a trivial task, since it requires the algorithm to distinguish the relevance of a piece of information. This also implies that the algorithm may not be able to know in advance what kind of information can be discarded or should be kept for later decisions, increasing the memory requirement. Knowledge of the domain can be exploited to define beforehand which kind of information is relevant.

3.1.6 Novelty detection

Novelties are hard to detect because they can't be matched against a "blacklist" of anomalies identified in the past and their behavior is entirely unexpected. To detect novelties the anomaly detector must compare the behavior of the system to a baseline normal behavior instead of searching for anomalous patterns. In unsupervised algorithm defining the baseline is a difficult task. In these applications a model of expected behavior is built using expert knowledge of the domain and deriving it from training data using robust techniques.

3.1.7 Distributed execution and scalability

Spatio-temporal data is inherently distributed in space. Algorithms that are able to exploit this property using parallelization can considerably reduce their computational time, an important optimization for achieving timely on-line results. In addition, distributed computation may allow for an early response based on local results without having to wait for the termination of the whole algorithm. Scalability is another important property of algorithms dealing with spatio-temporal datasets, usually high in dimensionality and with many records. Distribution and parallelization can improve dramatically the scalability of an algorithm.

3.2 Application of state-of-the-art techniques

We present and examine in depth state-of-the-art algorithms for spatio-temporal anomaly detection. We focused on algorithm that satisfied most of the characteristics of our problem: as such we considered algorithms able to analyze both spatial and temporal features, with low time complexity. The considered algorithms, STCOD, MFSS and STOUT were implemented and tested on our data: we detail here the procedure followed by each algorithm and analyze the results.

Structure of the data

The subject of our analysis is an unlabeled dataset containing measurements of five features that describe the quality of a mobile network. Each feature has been measured hourly over the span of a month in roughly 6500 locations distributed in the region of Piemonte. These locations are identified by a code and values of longitude and latitude. We simplified our data by aggregating locations that have the same values of longitude and latitude. We refer to these aggregated locations as nodes. Data for each node is aggregated by averaging for all features the values related to locations represented by that node.

The five features that describe the quality of the network are the behavioral attributes of our task, and can be divided into two categories:

- Three of them (Network Effectiveness Rate (NER), Call Drop Rate (CDR), Handover Rate (HO)) measure characteristics of the voice service, and are numerical values going from 0 to 100 (percentages). The distribution of these attributes is not normal and the vast majority of the instances assume the same value (0 or 100).

- Two of them (Erlang High Speed (HS) and Throughput (THP)) measure characteristics of the data downlink service, and are non negative numerical values. These values are approximately gaussianly distributed.

The dataset is not complete, as there are several missing values. Since the following algorithms cannot deal with missing values, the dataset was completed before testing. In each node, the missing values were imputed using a linear regression model computed from the non-missing values directly preceding and succeding the missing one.

3.2.1 STCOD

STCOD is an algorithm developed by Wang and Wu [27] for detection of outliers in sensor data.

STCOD procedure

In STCOD, each sensor is represented by a node in a graph, where the connections in the graph indicate the neighbors of that node. In our implementation, instead of using a graph, we consider neighbors of a node i the nodes that are the first k nearest neighbors of i, using as measure of distance the euclidean distance between the coordinates of the nodes.

The first phase of STCOD is performed by every node separately, and consists in an univariate on-line anomaly detector that uses sliding windows to decide whether the node is anomalous.

Definition 1 *Given node i and time t, given $x_i(t)$ the value of the node i at time t, given Δt the size of the sliding window; the sliding window $W_i(t)$ is defined as:*

$$W_i(t) = \{x_i(t - \Delta t + 1), x_i(t - \Delta t + 2), ..., x_i(t)\} \tag{3.1}$$

The current window $W_i(t)$ is compared with the previous window $W_i(t-1)$. These two windows are used to compute a self-similarity measure, the correlation.

Definition 2 *Given two sliding window $W_i(t_1)$ and $W_j(t_2)$; the correlation between the two windows is defined as:*

$$corr_{i(t_1),j(t_2)} = \frac{W_i(t_1) \cdot W_j(t_2)}{\|W_i(t_1)\|_2^2 + \|W_j(t_2)\|_2^2 - W_i(t_1) \cdot W_j(t_2)} \tag{3.2}$$

$$\|W_i(t)\|_2^2 = |x_i(t - \Delta t + 1)|^2 + ... + |x_i(t)|^2 \tag{3.3}$$

The correlation is compared to a threshold *thr*: nodes with $corr_{i(t),i(t-1)} < thr$ detect a change in the current behavior with respect to the past, and flag themselves as candidate anomalies at time t.

The second phase of STCOD is performed only by the candidate anomalies. It is not necessary for all the nodes to have finished the first phase. Each candidate anomaly i requests from its neighbors a collective check via a similarity voting. For the voting procedure, each neighbor n computes $corr_{n(t-1),i(t-1)}$, the correlation between the window $W_i(t-1)$ of the candidate node and the $W_n(t-1)$ of the neighbor. $corr_{n(t-1),i(t-1)}$ is then compared to the same threshold *thr*. If the similarity of the previous windows is

found higher than the threshold, then the correlation between the candidate and the neighbor at time t is computed. This correlation $corr_{n(t),i(t)}$ is compared again to threshold thr. Using this three comparison, each neighbor provides a vote as following:

$$vote_n = \begin{cases} 0, & \text{if } corr_{n(t-1),i(t-1)} < thr; \\ -1, & \text{if } corr_{n(t-1),i(t-1)} > thr \text{ and } corr_{n(t),i(t)} < thr; \\ +1, & \text{if } corr_{n(t-1),i(t-1)} > thr \text{ and } corr_{n(t),i(t)} \geq thr. \end{cases} \tag{3.4}$$

Informally, a vote of 1 signals that the node and its neighbor remained correlated even during an anomaly event; -1 signals that the node was correlated to its neighbor before the anomaly but its current anomalous behavior is isolated. Finally, 0 signals that the two neighbors were not correlated in the past, therefore the vote does not sway the result.

The answers of all the neighbor nodes are then summed by the candidate node i. If the result is negative or zero, the node is flagged as an outlier; if the result is positive, the node is flagged as an anomaly.

Results of STCOD application

The algorithm was applied separately on the features of the dataset, since STCOD is a univariate anomaly detector.

The parameters of STCOD that require manual setting are: the window size Δt, the threshold thr, the number of nearest neighbors k. In our implementation, we separated the threshold in two, the self-similarity threshold thr_{self} and the neighborhood threshold thr_{neigh}. To set the thresholds for the analysis, we estimated a small quantile of the various correlation computed by STCOD and the thresholds that correspond to those quantiles. These computations showed us that the distribution of the values of correlation between oneself and other neighbors are very different. Figures 3.1 and 3.2 show the distributions of the two correlation measures. The graph of the correlation between a node and its neighbor in the previous slot is omitted: it is very similar to figure 3.2.

Running the algorithm on the first week of the HS table of the data, a large number of anomalies are found. In every time frame at least three nodes are always found anomalous.

Since STCOD does not provide a measure of the level of anomaly, we can only measure the general condition of the network by counting the number of anomalies in each time slot. The results are shown in picture 3.3. The plot shows that peaks of anomalies are present during week 1. The time

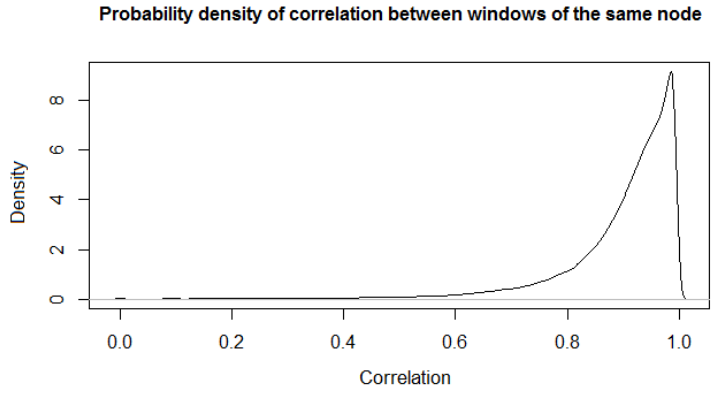

Figure 3.1: STCOD, correlation between $W_i(t)$ and $W_i(t-1)$; the graph shows high correlation between a node and its recent past.

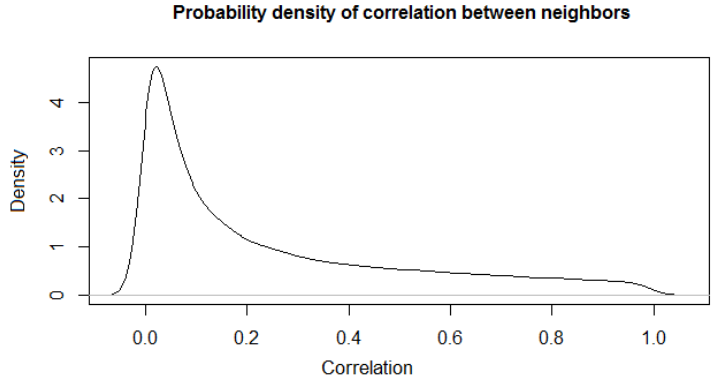

Figure 3.2: STCOD, correlation between $W_i(t)$ and $W_n(t)$; the graph shows low correlation between a node and its neighbor.

frames in which peaks are found are coherent with analysis of variance in the data.

It is also interesting to note that the number of anomalies appears to be periodic. This happens because STCOD does not make any adjustments for the fact that during periods where the absolute value of the feature is higher, the standard deviation is also higher.

We note that the algorithm only returned results as anomalies, and didn't flag any outlier.

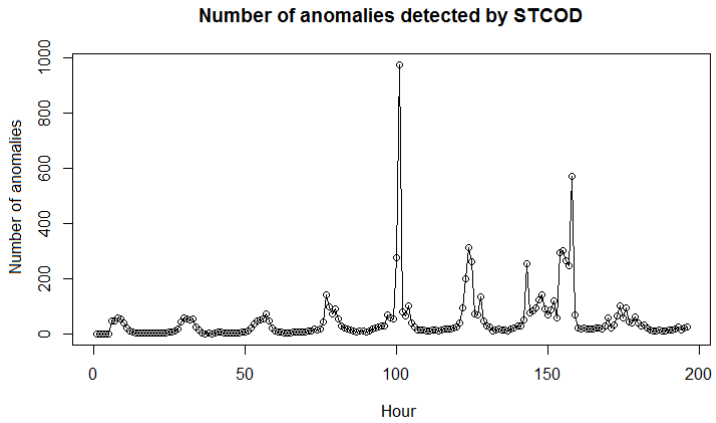

Figure 3.3: STCOD, number of anomalous nodes found over week 1; peaks correspond to high level of variance in the data; note the smaller bumps corresponding to daytime periods.

Observations on STCOD

We first observe that the correlation measure appears to be sensitive to the absolute value of the feature: as such, for periodical data, during the periods of high value, the correlation signals more anomalies. This makes STCOD unsuitable to seasonal data.

A deeper issue of STCOD lies in its underlying assumption that nodes should always behave in very similar ways. The fact that different nodes are assumed to behave similarly is particularly evident since the algorithm utilizes the same measure to compare a node to itself, and to compare a node to a neighbor. In fact, in the original formulation the same threshold is used to compare different nodes and the same node in different thresholds. As can be seen by the distribution of the values of correlation, this assumption clearly does not hold.

3.2.2 Fast Subset Scan

Fast Subset Scan is an algorithm developed by Neill et al [29], for spatio-temporal discovery of disease outbreaks. The main innovation of FSS lies in the Linear Time Subset Scanning property. Neill gave proof that, if a function satisfies the LTSS property, it is possible to find the most anomalous subsets of data in linear time instead of exploring every possible subset, which would require 2^N time for N data points.

Neill developed several variations and improvements of the first FSS al-

gorithm. In [8] a multivariate version of the same algorithm is presented, which we will refer to as MFSS. In [13] a general FSS algorithm, FGSS, which does not make assumptions on the distribution and considers seasonality is presented for categorical data. We implemented a variation of the multivariate FSS that also applies the corrections for seasonality of the FGSS algorithm.

As a first step we report here the definition of the LTSS property as described by Neill in [29].

Linear Time Subset Scan property

Let D = {R_1, R_2, ..., R_N} be a dataset containing N data records, and S be a subset of D, $S \subseteq D$. Let F(S) be the score function, a mapping from S to a real number we will call score of S. Let G(R_i) be the priority function, a mapping from R_t to a real number, the priority. We indicate with the notation $R_{(j)}$ the record in D with the jth highest priority. We can now define the LTSS property:

Definition 3 *Given a set D, the score function F and the priority function G satisfy the LTSS property if and only if:*

$$max_{S \subseteq D}\{F(S)\} = max_{j=1..N}[F(R_{(1)}, ...R_{(j)})] \tag{3.5}$$

The LTSS property allows us to only examine N subsets of records and guarantees that no other subset has higher score than them. This is accomplished simply by sorting the records by their priority and applying F to the first record, then the first two records, and so on. In case we are interested in more than the N most anomalous subsets, it is sufficient to repeat the procedure without including the records with highest priority already considered. Even in this case the procedure is optimized, since it would require examining at most N^2 subsets instead of 2^N.

FSS Procedure

The priority of each record is computed via the priority function G. The records are ordered from the one with the highest priority to the one with the lowest, and the F function is applied to the subsets composed as described in the LTSS property. We can choose to signal subsets with score higher than a threshold or choose a specific number of subsets with the highest priority.

There are several statistics that satisfy LTSS and can therefore be used as F and G; the choice of these statistics depends on the assumed distribution of the data.

We show here the procedure described in [8], however instead of the EBP statistic we use the EBG and Gaussian statistic described in [29].

For the univariate EBG, the following parameters are introduced: given a spatial location s_i in a subset S of the total locations, and given x_i, μ_i, and σ_i^2 respectively the value, historical mean and historical variance of a feature:

$$C_S = \sum_{s_i \in S} \frac{x_i \mu_i}{\sigma_i^2} \tag{3.6}$$

$$B_S = \sum_{s_i \in S} \frac{\mu_i}{\sigma_i^2} \tag{3.7}$$

$$G(s_i) = \frac{x_i}{\mu_i} \tag{3.8}$$

$$F(S) = \frac{(C_S - B_S)^2}{2B_S} \tag{3.9}$$

Another statistic, assuming Gaussian distribution (for increased counts):

$$C_S = \sum_{s_i \in S} \frac{(x_i - \mu_i)^2}{\sigma_i^2} \tag{3.10}$$

$$B_S = \sum_{s_i \in S} \frac{\sigma_i^2}{\sigma_i^2} = |S| \tag{3.11}$$

$$G(s_i) = \frac{(x_i - \mu_i)^2}{\sigma_i^2} = C_{s_i} = Z_i^2 \tag{3.12}$$

$$F(S) = \frac{1}{2}(B\, log(\frac{B}{C}) + C - B) \tag{3.13}$$

3.2.3 MFSS procedure

In [8] strategies for extending the application of these statistics to the multivariate state are presented. In this work we apply the Subset Aggregation strategy scanning over location. For each couple of nodes and features, c_{im} and b_{im} are computed with the formulas seen above for C_S and B_S, with S $= \{s_i\}$ and x_i the value of the considered m feature.

These values can be aggregated by feature, resulting in $C_i = \sum_m c_{im}$, or by location, resulting in $C_m = \sum_i c_{im}$. The same applies for B_i and B_m.

In this work, we consider the aggregation derived by using all the feature values, C_i.

The FSS search procedure is then applied by computing C and B as the sum of the Ci and then applying G and F.

In this work we modified the algorithm so that the baseline values μ and σ are computed and used depending on the hour under examination.

Results of MFSS application

We applied MFSS using both the EBG and Gaussian formulas, over the NER, CDR and HO data which can be interpreted as counts of anomalous events.

The MFSS algorithm does not require any parameter. We tested MFSS on our data to find the top 3 clusters of anomalies in each hour. Results are shown in 3.4

Using the Gaussian measure, the result of the MFSS showed a varying number of anomalous nodes anomalous. We note that the anomaly score of the resulting clusters varies over time from a minimum of 69 to a maximum of 10^14. This makes it almost impossible to determine a significant threshold to accept a report as anomaly. We instead plot the number of different nodes that appear anomalous for each hour.

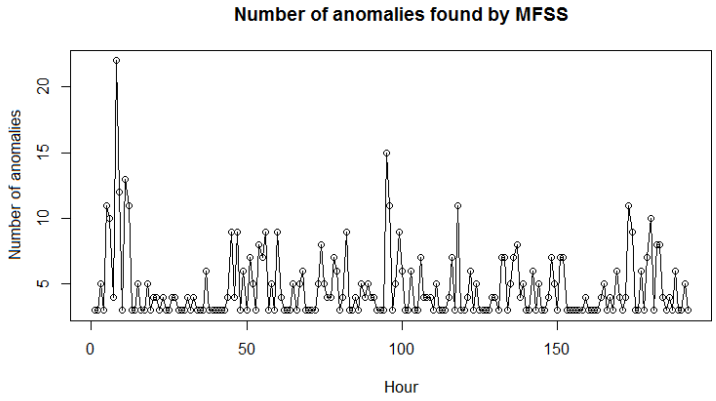

Figure 3.4: MFSS, number of unique anomalous nodes found over week 1, using NER, CDR and HS.

Using the EBG measure, all nodes are found anomalous over the whole week.

We also report the application of MFSS on the HS feature, which is normally distributed, using EBG statistic. Figure 3.5 shows the anomalous

nodes in the top 3 clusters for each hour.

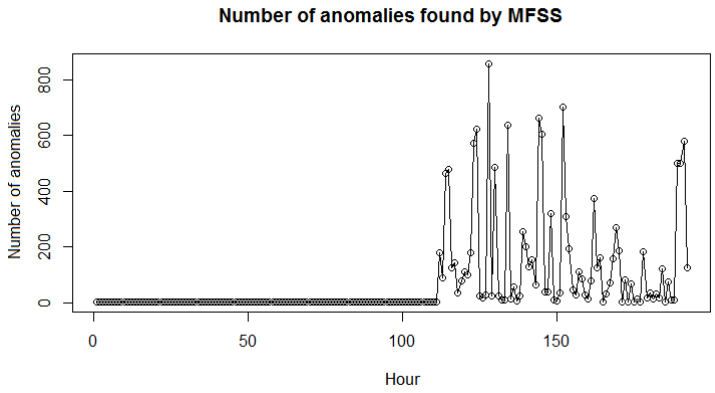

Figure 3.5: MFSS, number of unique anomalous nodes found over week 1 using HS.

Observations on MFSS

The statistics used in the FSS algorithms rely on count data. This is a problem in many application domains, including ours, where measures are continuous values and the count of anomalous events is not available.

The main issue of the FSS algorithm is that the final score has no predictable upper limit. When the results of the algorithm are the most anomalous cluster within each hour, this is not a problem, since score within an hour are comparable to each other. However, comparing scores computed in different hour is impossible because of the huge differences in scores. Furthermore since we cannot predict how high the scores can become until the end of the algorithm, we cannot define a threshold to distinguish anomalous clusters from normal ones. For example, it is unclear from figure 3.5 which clusters represent an anomaly. A possible interpretation would be that during the first part of the week there were very few anomalies compared to the second part of the week. Another interpretation would be that during the first part the found nodes were so anomalous that they were constantly the top anomalies, while the high number of different anomalies in the second part of the week simply indicates there is no distinct anomalous data. Without a clear meaning of the scores, neither interpretation can be automatically discarded. The possibility of conflicting interpretation shows that FSS requires human analysis, which makes it unfeasible for on-line applications.

3.2.4 STOUT

STOUT [51] is a multivariate spatio-temporal detector that operates off-line on tensors, multi-dimensional arrays.

STOUT procedure

In our application we provide to STOUT a tensor of three dimensions: the spatial coordinates, the considered hours and the features.

The first operation of STCOD is binning of data in slices containing equal number of hours $size_{bin}$. In our implementation slicing is done over the time dimension: therefore the dataset is divided into $\frac{numberofhours}{size_{bin}}$ bins. After binning, a base tensor representing the normal behavior of the system is computed by randomly sampling a $size_{sample}$ number of bins from the dataset. These bins are then averaged to compute the base tensor that will be compared to the others.

The spatial and time contiguity matrices that represents if two points of data are close in space or time are computed: we first compute the time distance matrix as the Manhattan distance between the timestamps of each couple of hours; the spatial distance is the euclidean distance between the the coordinates of every couple of nodes. The distance matrices are compared to two previously defined thresholds, the spatial contiguity threshold $thr_{sp,contig}$ and the time contiguity threshold $thr_{t,contig}$. The couples of nodes that are found to be less distant than $thr_{sp,contig}$ are flagged as contiguous in the spatial contiguity matrix by flagging their comparison with a 1, while other couples have their flag set to 0. The same process is repeated with the time distance matrix to generate the time contiguity matrix.

The base tensor is then decomposed using CANDECOMP/PARAFAC decomposition [19] in several rank one tensors (vectors). The CANDECOMP/PARAFAC decomposition of each bin is also computed. The difference between an analyzed bin i and the base bin is then computed with the following formula:

$$diff_{sp} = (\lambda_{base} * S_{base}) - (\lambda_i * S_i) diff_t = (\lambda_{base} * T_{base}) - (\lambda_i * T_i) \quad (3.14)$$

We now compare $diff_{sp}$ and $diff_t$ to thr_{sp} and thr_t, two thresholds for space and time respectively. Bins that have differences higher than the threshold are distant from their spatial and temporal neighbors; we score these beins with a measure called neighborhood dinamicity.

Definition 4 *The Neighborhood Dinamicity β is the measurement distance within the spatio-temporal neighborhood divided by the measurement distance outside of the neighborhood.*

Let S_i be the set of all nodes whose difference from the baseline is higher than the chosen threshold. Let T_j be the set of all hours in the current bin whose difference from the baseline is higher than the chosen threhsold. Let C_s be the set of nodes contiguous to an element of S_i; C_t is the set of contiguous hours of the element. For each node and for each hour in S_i and T_j respectively, calculate β as:

$$\beta := \frac{\sum_{s \in C_s, t \in C_t} dist(v_{st})}{\sum_{s \notin C_s, t \notin C_t} dist(v_{st})} \tag{3.15}$$

If β is over a predefined thr_β, the (i,j) node is considered anomalous during hour j.

Results of STOUT application

For STOUT, the parameters that need to be chosen are: the sample size used to compute the base tensor; the size of a bin; the thresholds in space and in time for two values to be considered contiguous, $thr_{sp,contig}$ and $thr_{t,contig}$; the thresholds in space and in time for which the distance between two contiguous values is too high, thr_{sp} and thr_t; the threshold of acceptance for an anomaly thr_β.

The STOUT algorithm was tested on the dataset after deseasonalizing it. The reason for the deseasonalization is that data with seasonality leads to wrong results depending on which bin is randomly sampled. We also used only a small subset of nodes for this tests because of high memory requirements of STOUT.

To determine the values to use for the thresholds, we find the cumulative distribution of these values and compute their 5th and 95th percentiles.

The algorithm showed varying results depending on the selected thresholds, where even small changes in the thresholds can change the results from no anomaly found to most nodes found anomalous. For some threshold values the algorithm did not terminate. In addition, even with the same parameters the resulting anomalies changed in successive iteration because of the random binning strategy.

Observations on STOUT

The main issue of STOUT is the unpredictable execution time derived from the random binning. Testing of STOUT resulted in execution times varying from seconds to minutes, and in several cases the algorithm did not terminate. We believe the cause of this issue is that STOUT assumes the bins to

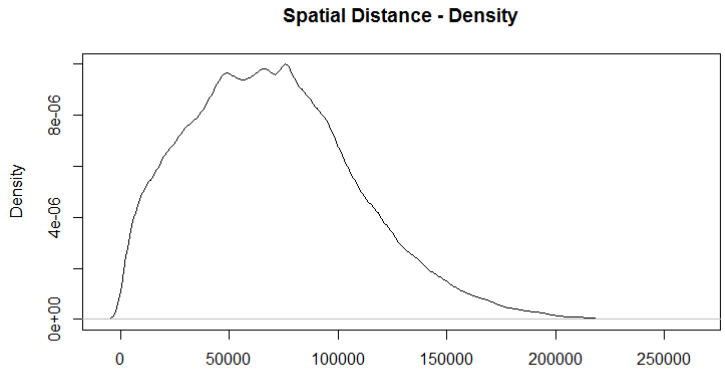

Figure 3.6: STOUT, distribution of the geospatial distance for contiguity.

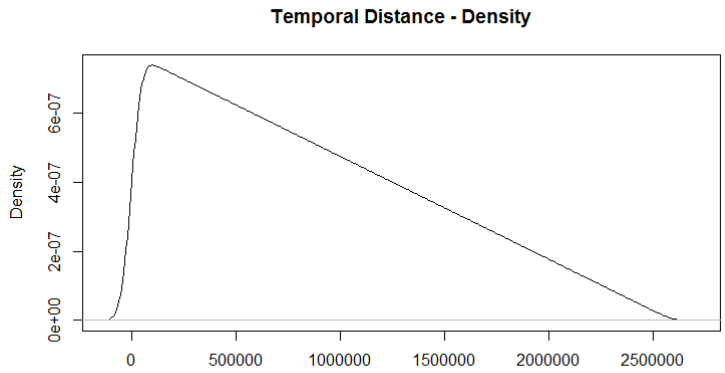

Figure 3.7: STOUT, distribution of the temporal distance for contiguity.

be selected from normal data, and data being more or less constant in time. If the bin that is randomly selected is not similar to the majority of other bins, the execution times increase significantly because of the necessity to examine more couples of location and time for anomalies. The STOUT algorithm also cannot perform well with seasonal data, since it assumes there is a single baseline bin that is valid during the entire duration of the algorithm execution.

STOUT has high requirements for memory usage, since the distance matrix over all the dataset is needed. This is unfeasible for high numbers of nodes, in our case STOUT would have required 50 TB of memory space to compute the spatial distance matrix for all the dataset.

Other minor issues of STOUT were shown: firstly, STOUT is inherently

off-line, needing the whole tensor to operate. Secondly, STOUT requires many thresholds that need to be set by the user.

3.2.5 Conclusions on state-of-the-art algorithms

The analysis of state-of-the-art algorithm showed that no algorithm can guarantee all our desired properties when applied to multivariate spatio-temporal data similar to our own.

In particular, we recap the main issues found for each algorithm:

- STCOD is able to detect major anomalies but is sensitive to seasonality of the data. Furthermore, STCOD does not provide an indication on which nodes are subject to the same anomaly, and doesn't score anomalies. Finally, it assumes that the behavior of the system is similar in all nodes and stable throughout time.

- FSS provides a valid framework for fast detection, but its statistics are not applicable to our data. Furthermore, the scores assigned by FSS are not useful to distinguish anomalies from normal results.

- STOUT is the directly applicable to multivariate spatio-temporal data, but its high memory requirement and its unpredictable execution time makes it unfeasible even for medium-scale problems. Furthermore, STOUT cannot operate on-line and is hard to tweak because of the many thresholds it uses. Finally, its random binning strategy makes it unsuitable for seasonal data and makes its results not repeatable.

Chapter 4

Multivariate Spatio-Temporal Anomaly Detection using Fisher's method

In this chapter, we describe our solution of the anomaly detection problem faced in this work. First, we provide an overview of MuSTF, the extension of the Fast Subset Scan framework we developed to solve our problem. We then focus on the most important procedures of the algorithm such as: (i) generating univariate p-values from each value in the dataset; (ii) generating a multivariate anomaly score for a node, using new score and priority functions; (iii) searching the most anomalous clusters of nodes in the dataset.

4.1 Extending Fast Subset Scan

We developed a new algorithm based on the FSS approach that adopts new priority and score functions; these functions solve the two main limitations of the FSS approach: firstly, none of the statistics used by the FSS family of algorithms is applicable to continuous data types; secondly, the priority associated to a cluster is only comparable to priorities relative to the same hour.

Our algorithm MuSTF (Multivariate Spatio-Temporal Anomaly Detector Using Fisher's method) processes a set of nodes; each node represents a different location identified by longitude and latitude, and is characterized by five time series of hourly measurements collected over the span of one

month: each series represents a feature of the network in that location.

During a preliminary training phase, the algorithm uses historical data to learn the baseline behavior of the system. The algorithm also computes the neighborhood of each node, which is the set of nodes nearest to its position, using as metric the euclidean distance between their coordinates. After the training phase, the algorithm operates on-line. Each iteration of the algorithm examines data relative to an hour, in order from the first to the last hour in the dataset. During an iteration, the algorithm executes the following steps:

- A new value arrives for each feature of a node. Using the new value and the parameters learned during the training phase, several p-values are computed as univariate anomaly scores of that feature.
- The univariate scores of each feature are combined into a single p-value using Fisher's method with weighted priorities. The result is used to compute the multivariate anomaly score of that node.
- After a score has been computed for every node, the algorithm searches for clusters of high scoring nodes in each spatial neighborhood. Each search reports clusters as anomalous if their cluster score is higher than a threshold chosen by the user.
- Nodes that are members of many anomalous clusters are considered anomalous.

The graph in figure 4.1 describes the high-level flow of the algorithm.

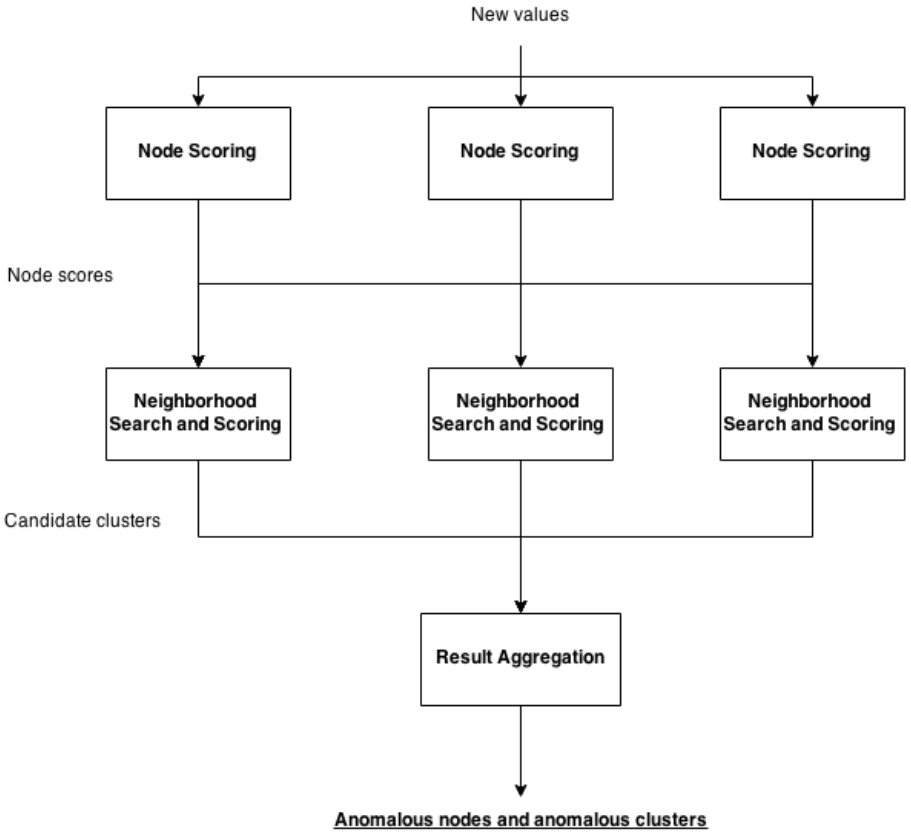

Figure 4.1: General schema of the process.

4.2 Univariate node scoring

In the FSS family of algorithms, the priority is a value representing the anomaly score of a node or feature, and is computed from the count of anomalous events and its difference from the baseline, which is the expected count of anomalous events.

In our algorithm we keep using the deviation from the baseline to compute an anomaly score, however we use the p-value of the deviation instead of the absolute value of the deviation.

Definition 5 *The p-value is the observed probability of a statistic being at least as extreme as the particular observed value when H_0 is true.*

If $Y = u(X_1, X_2, ..., X_n)$ is the statistic used to test H_0, and the observed

value $d = u(x_1, x_2, ..., x_n)$, the p-value of d is:

$$p{-}value(d) = \begin{cases} Pr(Y \geq d|H), & \textit{for measure of right tail event;} \\ Pr(Y \leq d|H), & \textit{for measure of left tail event;} \\ 2 \times \min(Pr(Y \geq d|H), Pr(Y \leq d|H)), & \textit{for measure of double tail event.} \end{cases} \tag{4.1}$$

Equivalently, if F_Y is the cumulative distribution function of Y when H_0 is true, the p-value of d is equal to $F_Y(d)$.[35]

We can compute the anomaly score of a feature by noting that it is high when the p-value is low. Formally, the anomaly score given a single p-value is the complement of the p-value.

Using the p-value has a series of benefits over the previous methodology. Firstly, the p-value is limited between 0 and 1. Secondly, the p-value has a well-defined interpretation, allowing us to understand the result. Thirdly, if the null hypothesis H_0 is valid and the assumed distribution is correct the p-value is uniformly distributed [11]; this allows us to evaluate the validity of our assumptions by checking if the distribution of p-values is uniform during training phase.

To calculate the p-values related to a value we need to know the distribution function of each feature. However in general we don't know these functions beforehand, so we can't directly compute the p-values. Furthermore, since different features will be differently distributed we need to have different ways to compute p-values for each feature. For these reasons, instead of assuming a specific distribution function for a feature, we use several methods to compute p-values according to different null hypotheses. All hypotheses assume that data is not anomalous.

4.2.1 Computing p-values

The first method used in this work is the lagged first seasonal difference, which is the deviation of the observed trend of a feature from its seasonal trend; the lag is the time interval over which the trend is computed. Consider the case of observed values x(t_{-n}, t_{-n+1}, ..., t_0) and seasonal data $\widetilde{x}$(t_{-n}, t_{-n+1}, ..., t_0). The lagged first seasonal difference, with lag i, is defined as:

$$\Delta_i = (x(t) - x(t-i)) - (\widetilde{x}(t) - \widetilde{x}(t-i)) \tag{4.2}$$

For example, Δ_1 indicates if in the last hour the feature increased or decreased faster than expected. Computing Δ_i with a small lag we measure fast changes in trend with low delay, while higher lags detect persistence

of a trend change. In our work we compute several of these measures with different lags to check for anomalies of various durations. To compute a p-value from the lagged difference, we note that its values appear normally distributed with mean 0 for all our features, allowing us to obtain the p-value by computing the z-score, the number of standard deviation from the norm of the lagged difference. The standard deviation of the lagged difference is learned during training phase.

We also use the current seasonal difference x(t) - $\widetilde{x}$(t) to measure deviation from the norm regardless of the past behavior. For features that are normally distributed, we can simply compute the z-score of the seasonal difference and compare it to its standard deviation, learned during training. A more general approach that does not assume a particular distribution is to learn the empirical cumulative distribution function of the feature during training. We can then use the empirical CDF to directly calculate the p-value according to definition 5, since for example p-value(x) = Pr(X $\leq$ x) = F_X(x), for left tail events.

Finally, we use regression to compute models of the time series. To compute this models, we apply ARIMA regression on windows of past values. We can then check the fit of a new value by comparing it to a forecast value and using the confidence intervals of the forecast to estimate the p-value of the new value.

Each resulting p-value measures the deviation of the value from the norm under different assumptions. Since we don't know how well each assumption fits our case, we estimate the validity of each method of computing p-values on every couple of feature and node by examining the distribution of p-values obtained during training phase, which should be uniform if the method assumption holds.

4.2.2 Weighted priority

In our work we use the p-values to measure the probability of occurrence of a value; the p-values should then be compared to reach a decision of a node being anomalous or not. However, directly comparing p-values would ignore the information regarding their accuracy. As in the FSS framework, the comparison between different nodes and features is done by computing a priority for each of them, which represents its anomaly score.

We now introduce the priority function that is used to generate a priority from the previously computed p-values. As a preliminary definition, we call unweighted priority P the complement of a p-value, thus P(d) = 1 - p-value(d).

Definition 6 *Let D_i be a number between 0 and 1, called the default priority of i. Let P_i be the unweighted priority we want to weigh. Let w_i be the weight, a number between 0 and 1 we use to weigh P. The weighted priority W(P_i, w_i, D_i) is:*

$$W(P_i, w_i, D_i) = \begin{cases} 1 - Y - (Y - 1 + P_i) \times \frac{|D_i - P_i|}{D_i}^{\frac{1}{w_i}}, & if\, P_i < D_i; \\ 1 - Y - (Y - 1 + P_i) \times \frac{|D_i - P_i|}{1 - D_i}^{\frac{1}{w_i}}, & else. \end{cases} \tag{4.3}$$

$$Y = [(D_i - P_i)^{\frac{1}{w_i}} + 1 - D_i] \tag{4.4}$$

Informally, the weighted priority is a function that changes the value of the unweighted priority so that it is closer to the default priority, in proportion to the given weight. An example can be seen in figure 4.2. For $w_i = 1$, the weighted priority is equal to the unweighted priority P_i, as can be seen in figure 4.3. For $w_i = 0$, the weighted priority is equal to the default priority D, as can be seen in figure 4.4.

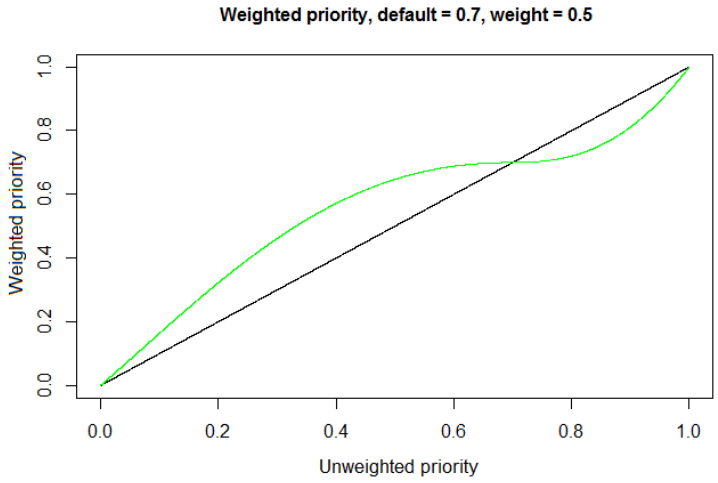

Figure 4.2: Weighted priority, weight 0.5, default 0.7; higher weight modifies more the priority towards the default

The weight of a p-value represents our confidence in using that p-value to reach a decision: if a p-value has low weight, we want it to have less influence on our decision when compared with others. The default priority represents a value of priority that does not sway our decision either towards accepting normality or anomalousness.

In the following section we describe the procedure used to estimate the weight of each method of computation of p-values.

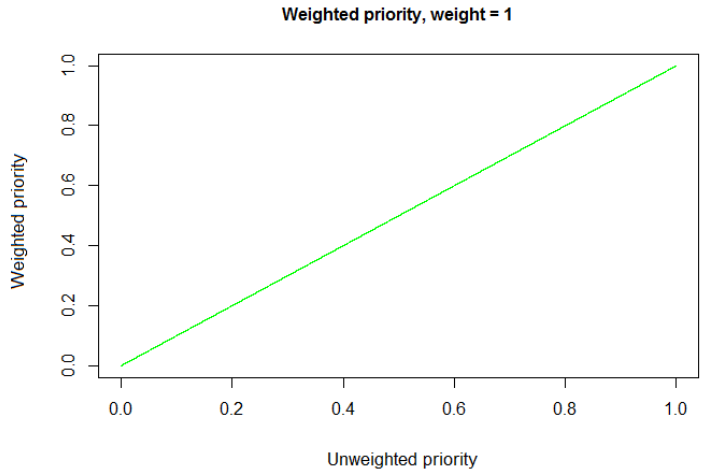

Figure 4.3: Weighted priority, weight 1, default 0.7; for max weight the priority remains unchanged

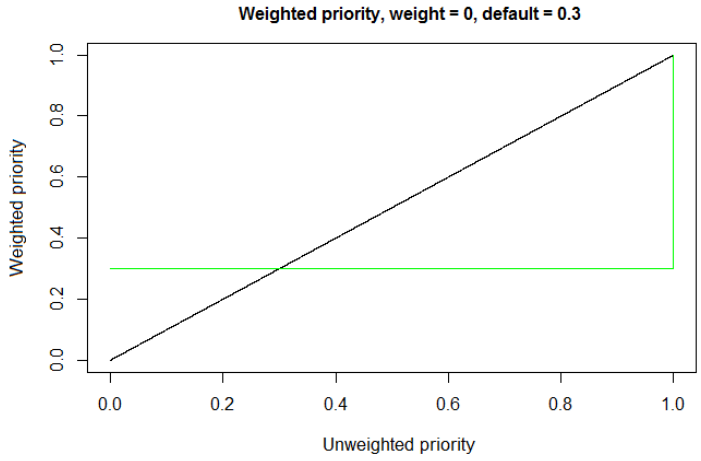

Figure 4.4: Weighted priority, weight 0, default 0.3; if no confidence is given to the priority, it's automatically brought to the exact default value, which indicates no decision

4.2.3 Weighing methods of computation of p-values

During the training phase, we compute the p-values of all features in every hour with the methods described before.

After we have all the p-values obtained by applying a method to a feature we analyze the probability density of the p-values: if the probability density is not uniform the results of a method need to be adjusted. To do so, we compute the weighted priority from all p-values using standard values for

weights and default priority. To find the optimal weight, we check how many of the computed priorities are over the anomaly threshold or not, and decrease the weight or increase to shift the distribution function to the left or right of the threshold. For example, when setting an anomaly threshold of 0.05, we expect that 5% of the p-values would be under the threshold: if we find more than 5% of p-values under the threshold, we decrease the weight of the node. We repeat this process until the error around the anomaly threshold is low. By doing this, we change the distribution of the weighted priorities so that it is approximately uniform in the critical area around the threshold. In figure 4.5 and 4.6 we show an example of the initial and final priority CDFs. After weighting the p-values we can directly compare priorities computed from different methods.

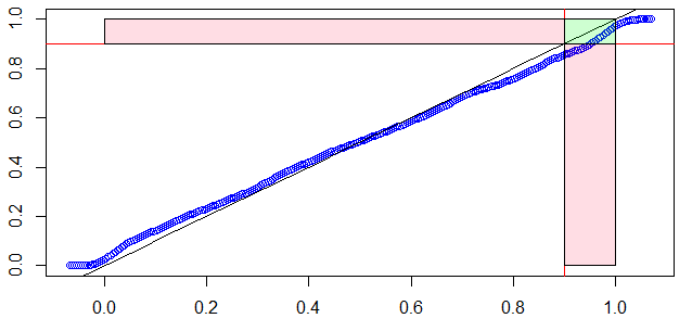

Figure 4.5: The blue points are the cumulative distribution function of the p-values. The red areas represent the false positive (on the right) and the false negatives (above), while the green area represents the true positives.

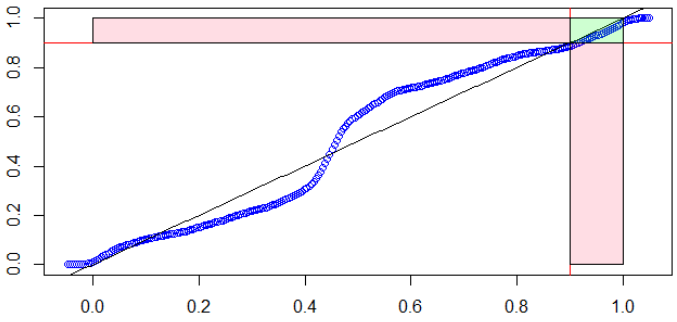

Figure 4.6: The graph shows that decreasing the weight changed the distribution function of the weighted priorities, making it almost uniform in the area of interest. Note that the area in white is not of our interest and does not influence the result of the decision process.

4.3 Multivariate node scoring

For each feature measured on a node, we have computed several p-values and weights associated to them. We also introduced the priority function that weighs p-values according to their accuracy. To compute a single score from multiple values, we use the score function, as in the FSS framework. In MFSS the multivariate score function was the sum or average of the univariate priorities. Unlike statistics based on counts however, our priorities cannot be summed or averaged meaningfully because they are based on p-values. Instead, we use a method proposed by Fisher in [1] to combine multiple p-values into one.

4.3.1 Fisher's method

Definition 7 *Let S be a set of p-values, computed independently under the same assumption H_0. Fisher's method is the function:*

$$Fisher(S) = -2 \sum_{p_i \in S} \ln(p_i) \tag{4.5}$$

The resulting value is χ^2 distributed with a number of degrees of freedom equal to $2\times|S|$.

We can compute a p-value from the result of the Fisher's method since we know its distribution. This p-value represents the probability that the null

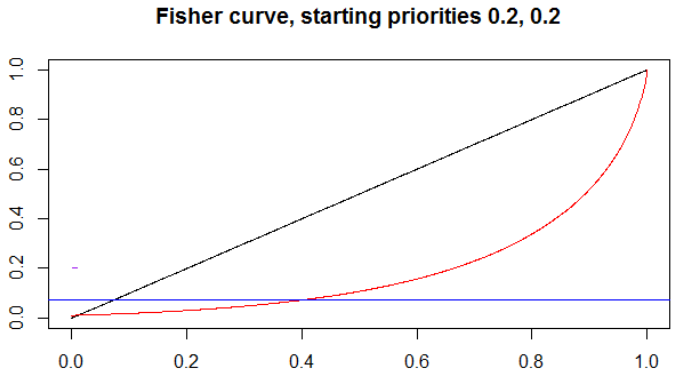

Figure 4.7: Curve of p-values computed from Fisher's method changing one of the input priorities; the remaining priorities are fixed. The blue line represents the value the Fisher's method would assume using only the fixed priorities.

hypothesis holds in all the input p-values. In figure 4.7 we see an example of result of the Fisher's method when changing one of the input p-values. As mentioned before, the p-values computed using different methods and derived from different features may not be directly comparable using the Fisher's method, since it assumes that all input p-values are identically distributed. We solved this issue by introducing weighted priorities, computed from the p-values, that approximately follow the uniform distribution in the area of our interest.

In literature weighted variants of the Fisher's method already exist, such as the weighted z-test [42] and the Lancaster method [7]. However we cannot use these methods for our application, because neither method satisfies the LTSS property. Thanks to our weighting strategy we can directly use the original Fisher's method that has the LTSS property. We now introduce our score function by integrating the Fisher's method with our weighting function.

4.3.2 Fisher's method with weighted priorities

Let S be a set of nodes or a set of features, and i $\in$ S. Let P_i be the unweighted priority $P_i = 1 - p_i$. Let w_i be the weight of the p-value of i. Let D_i be the default priority for i.

Definition 8 *The Fisher's method with weighted priorities is the function*

$\Phi(S)$, *defined as:*

$$\Phi(S) := -2 \sum_{i \in S} ln(1 - W(P_i, w_i, D_i)) \ \chi^2 \tag{4.6}$$

The resulting value $\Phi(S)$ *is* χ^2 *distributed with a number of degrees of freedom equal to* $2 \times |S|$

In the following section, we prove that the function Φ(S) satisfies the LTSS property making use of the theorems proved by Neill in [29].

4.3.3 Strong LTSS property of fisher with weighted priorities

Definition 9 *For a given data set A containing N nodes, the score function F(S) and the priority function G(Ri) satisfy the strong LTSS property if and only if, for all* $j \in [1,N]$, $\max_{S_j \in A \,:\, |S_j| = j} = F(\{R_{(1)}, R_{(2)}, ..., R_{(j)}\})$*:*

We now prove that the score function $F(S) = \Phi(S)$ satisfies the strong LTSS property with priority function $G(R_i) = W(P_i, w_i, D_i)$.

Theorem 1 *Let* $F(S) = f(X, |S|)$*, with X(S) a function of one additive sufficient statistic of subset S,* $X(S) = \sum_{R_i \in S} x(R_i)$*. If F(S) is monotonically increasing with X, then F(S) satisfies the strong LTSS property with priority function* $G(R_i) = x(R_i)$*.*

Proof 1 *Let* $x(R_i)$ *be the weighted priority* $W(P_i, w_i, D_i)$*. We consider the function* $ln(1 - x(R_i)) = ln(1 - W(P_i, w_i, D_i))$*. The natural logarithm function ln(y) is monotonically increasing with y; in our case we see that* $y = 1 - W(P_i, w_i, D_i)$*, therefore ln(y) is monotonically decreasing with* $W(P_i, w_i, D_i)$*. We also see that the function* $\Phi(S) = -2 \sum ln(y) = 2 \sum -ln(y)$ *therefore* $\Phi(S)$ *is monotonically increasing with -ln(y). Since -ln(y) is an additive sufficient statistic of subset S, depending only on record* R_i*, we can say that* $\Phi(S)$ *satisfies strong LTSS with* $G(R_i) = -ln(y)$*. Furthermore, since ln(y) is monotonically decreasing with* $W(P_i, w_i, D_i)$*, for transitivity* $G(R_i)$ *monotonically increase with* $W(P_i, w_i, D_i)$*. Therefore the given functions* $\Phi(S)$ *and* $G(R_i)$ *satisfy the strong LTSS property.*

Thanks to the LTSS property, we can now perform in linear time an exact search over a set of weighted priorities following the FSS procedure, and find any number of anomaly subsets in reasonable time.

A procedure to estimate weights for each method is defined in the previous section. Having established our score function, we can now describe a procedure for estimating the default priority.

4.3.4 Estimating the default priority

Definition 10 *Let S be a set of features or nodes, characterized by a set of priorities, a set of weights and a set of default priorities. We indicate as S_{-i} the set S without values related to node or feature i. We define as the $\bar{D}_{best}$ the set of default priorities d_i such that:*

$$\widetilde{d}_i \in \bar{D}_{best} iff \widetilde{d}_i = arg\min(d_i|\Phi(S_{-i}) - \Phi(S)), d_i \in [0,1] \tag{4.7}$$

We note that to compute the default priority d_i, we first need to compute the weighted priorities of all points other than i, which in turn would require to know all the other default priorities. It is evident that an exact solution of the problem would require a search over a space with N dimensions; this is an unfeasible solution, especially considering that we would need to compute the default priority at each iteration of the algorithm.

Instead, we compute an approximation of the default priorities using a simple iterative approach: during each iteration we estimate the default priorities one by one using the default priorities computed in the previous iteration. This approach converges to a set of low error default priorities within very few iterations.

4.3.5 Combining weighted priorities

We can combine the p-values generated by the univariate methods using the Φ function and the weights and default priorities computed as described before. The combination is per feature, combining p-values generated from the same feature using different methods; this approach generates a univariate score for each feature of the node. To compute a single value that describes the anomalous state of the node we combine all the univariate scores into one using again the Φ function. Since the priorities were already weighted when computing the univariate scores, for this phase we use them without changing their value. The result of this function is used to compute the p-value or unweighted priority of the node.

If the result indicates the presence of an anomaly, we also search within the set of univariate scores using the FSS procedure to find the most anomalous features. Note however that this task does not influence the final score of the node that is computed using all features.

The procedure from the start of MuSTF to the end of the multivariate scoring process is summarized in figure 4.8

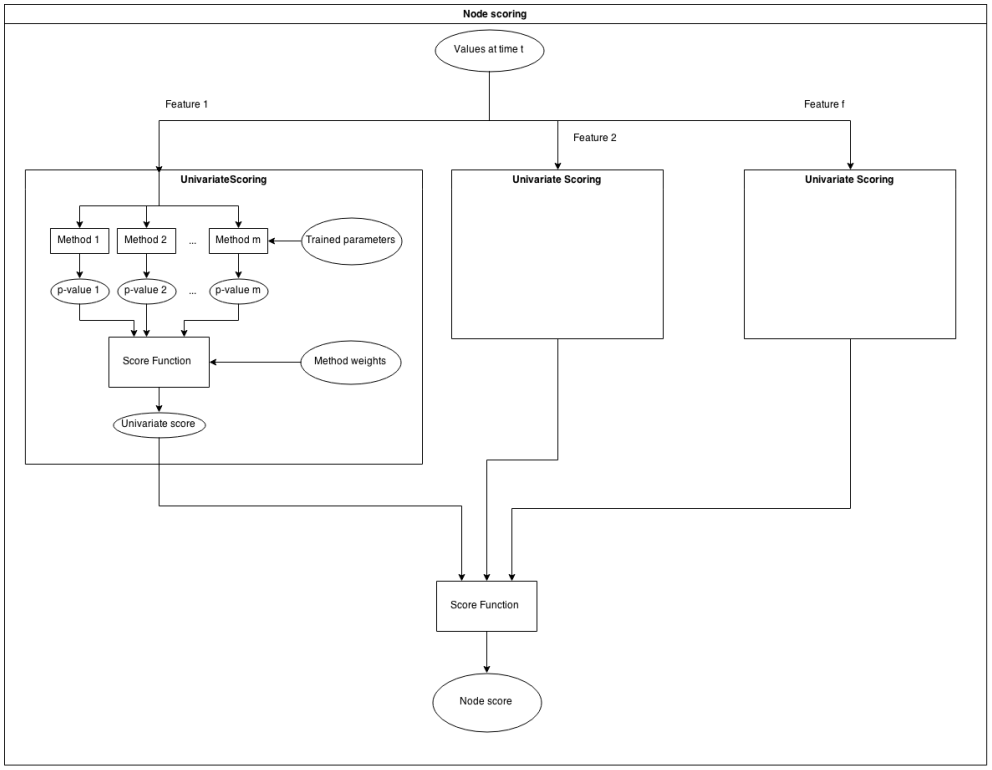

Figure 4.8: Schema of the node scoring phase of the algorithm.

4.4 Cluster scoring

After the first phase of the algorithm, we have the unweighted priority of each node that represents its level of anomaly independent of its context. During the second phase of the algorithm, nodes evaluate the state of their spatial neighborhood by searching for sets of nodes that have high priorities.

The unweighted priorities of the neighboring nodes are combined together using once again the function $Phi(S)$. The weights used in this phase of the algorithm represent the similarity between the past behavior of two neighbors. Thanks to the strong LTSS property, we can perform a scan on the weighted priorities of the nodes and find which clusters of nodes in the neighborhood present high anomaly scores. If the score of a cluster is higher than the anomaly threshold set by the user, we consider the cluster an anomaly candidate.

In the following section we describe the process of updating weights between neighbors. This update is performed after the computation of cluster scores.

4.4.1 Weighing nodes

We maintain weights for each couple(node$_i$, neighbor$_j$). Initially the weights are all equal to one. At the end of every iteration of the algorithm, the weights are updated to reflect the correlation in the behavior of a node and its neighbors. The algorithm computes the discounted cumulative difference δ between the current node and each of its neighbors. The values used to evaluate the correlation between two neighbors are windows of their unweighted priority, since we are only interested in the correlation between their anomalous states. Given a node i and its neighbor j, the windows of priorities are C_i = {P_i(t), P_i(t - 1), ..., P_i(t - m)} and C_j = {P_j(t), P_j(t - 1), ..., P_j(t - m)}. The discount λ is a parameter set by the user between 0 and 1 that is multiplied to the previous window to decrease the effect of past priorities in the weight. Formally, the discounted cumulative difference would be:

$$\delta_{i,j}(t) = \frac{\sum\limits_{0<i<m} |P_i(t-i) - P_j(t-i)| \times \lambda^i}{\sum\limits_{0<i<m} \lambda^i} \tag{4.8}$$

After computing $\delta_{i,j}$, the weight is updated using a weighted average between the previous value of the weight and $\delta_{i,j}$.

The cluster scoring phase of MuSTF is summarized in 4.9.

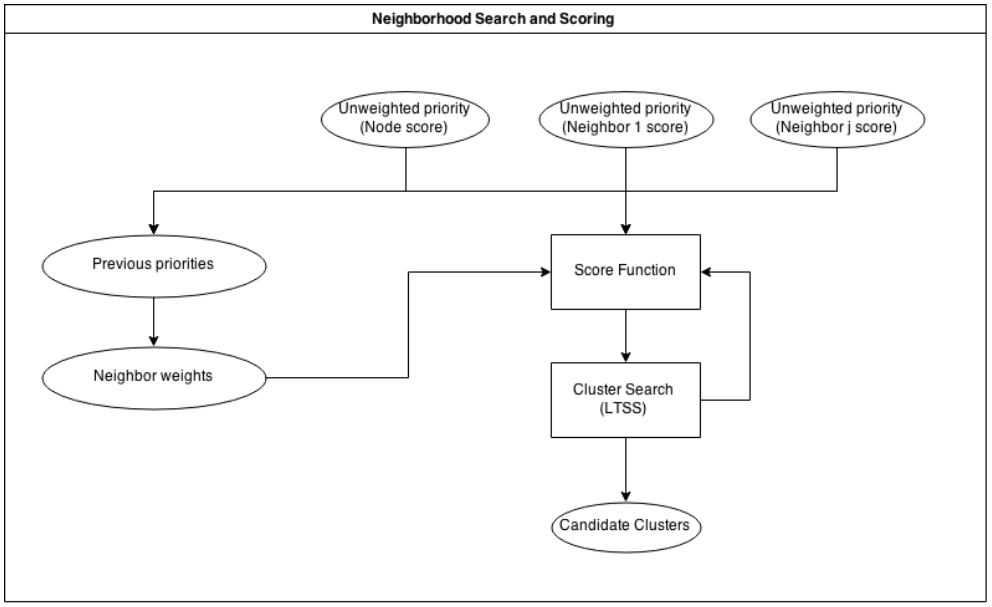

Figure 4.9: Schema of the neighborhood scoring phase of the algorithm.

4.4.2 Result aggregation

As a final step we aggregate the partial results given by the cluster scoring phase. We compute the vote count of a node $_i$ as the number of candidate clusters that contain $_i$. Since every node has k neighbors, we can consider a node anomalous if it appears in at least k_{min} candidate clusters. The k_{min} threshold is the minimum consensus between neighbors necessary for a node to be identified as anomalous, otherwise it is considered an outlier and removed from all candidate clusters. After identifying the anomalous nodes, we merge the candidate clusters that overlap by some nodes, to identify clusters bigger than the neighborhood. The resulting clusters of anomalies are the final result of our algorithm.

This final phase of the algorithm is not present in the FSS frame work, and is instead inspired by STCOD voting strategy. The aggregation phase is summarized in figure 4.10.

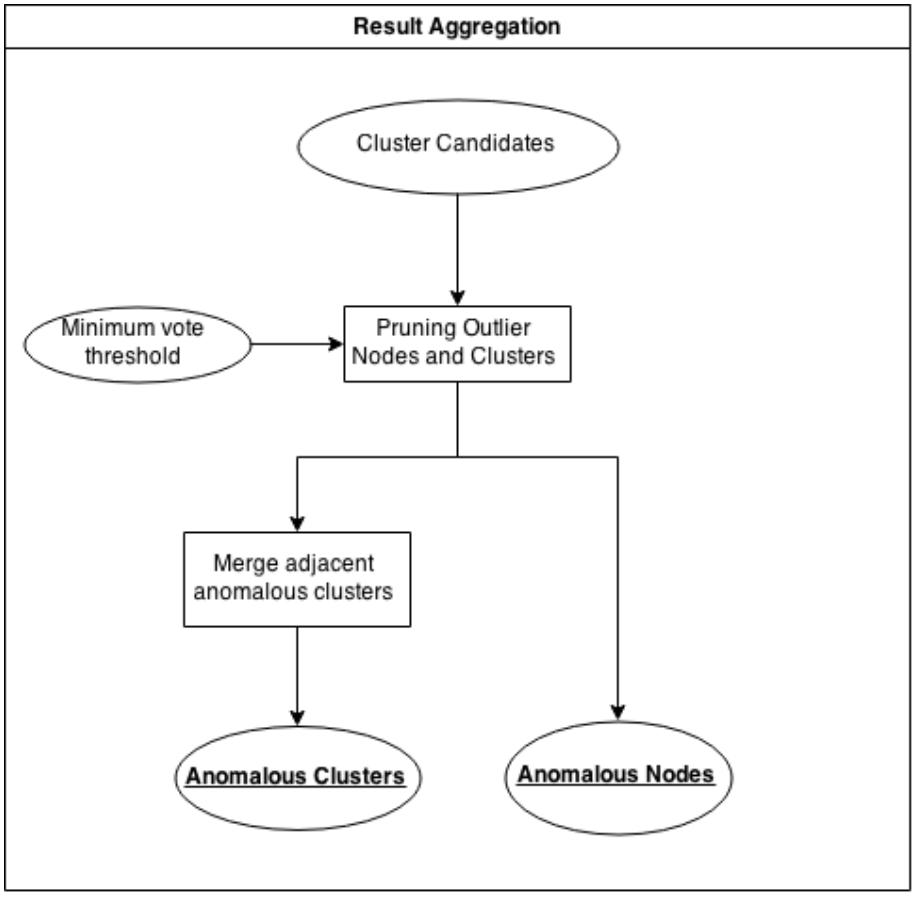

Figure 4.10: Schema of the final phase of result aggregation.

Chapter 5

MuSTF implementation

In this chapter, we provide the pseudo code of the MuSTF algorithm we developed. We first describe the notation used in the rest of the chapter. We then give the pseudo code of the training algorithm used before MuSTF. Finally, we provide the pseudo code of the various procedures of MuSTF.

5.1 Terminology and notation

The dataset used by MuSTF is a set of values identified by the hour when they were collected, the feature they measure and their node, which represents their physical location. We give here a few indications about the notation used in the pseudo code of the following sections: We indicate with D the dataset. We indicate with i $\in$ [1,N] the index of the node. Each node is characterized by two special features x_i and y_i, its spatial coordinates. We indicate with t $\in$ [1,T] the index of an hour. Note that hour is intended as an interval of time, therefore going from hour 1 to hour T at the end of the dataset. The hour of the day is instead called the period, which goes in our application from 0 to 23. We indicate with f $\in$ [1,F] the index of a feature. In our application we have five features. We indicate with m $\in$ [1,M] the index of a method used to generate a p-value from a value. We indicate with k the number of nodes in a spatial neighborhood. We indicate with $v_{i,f}(t)$ the value of feature f at hour t in node i.

We also define here the following abbreviations: $v_{i,f}$ is the set of all values assumed by a feature f in node i $v_i(t)$ is the set of all values of any feature assumed by a node i in time t v_i is the set of all values related to node i == V_i

The p-values generated is indicated by the letter p. In particular, $p_i(t)$

indicates the final p-value of the node at time t. $p_{i,f}^{(m)}(t)$ is the value of the p-value generated by method m using as input the feature f of node i at time t. $p_{i,f}(t)$ is the combination of p-values generated by any method for the f feature, the univariate node p-value. $p_i^{(m)}(t)$ is the combination of p-values generated by one method for all features, the multivariate node p-value.

We indicate priorities as P, and the priority function that generates them G(i) or G(f) depending on what they are applied (nodes or features). Note that for simple priority functions as P = G(i) = 1 - p_i we may indicate the full formula instead of using the G notation.

5.2 Training phase of MuSTF

The weights and parameters used to compute p-values on a specific node are determined during the training phase. The dataset used during training is a set of historical data related to the same nodes. The training phase of MuSTF is robust to anomalies present in the training dataset.

The first phase of the training is learning seasonal parameters: this is shown in algorithm 1. The training begins by computing the expected seasonal values for each node and feature. Seasonality is separated between ferial and festive days by a vector of booleans, indicating if each hour is in a festive day. The periodicity of the series must be defined by the user with the freq parameter. The deseasonalized dataset is computed by subtracting the expected seasonal values from the data. Then the cumulative distribution functions (CDF) for each feature are computed from the deseasonalized dataset.

Algorithm 1 Seasonal training

function SeasonTraining(D, Festive, freq)
  for all $v_{i,f} \in D$ **do**
    for hour in [1:freq] **do**
      $t_{hour} \leftarrow t \mid (t - 1) \bmod freq = hour, t \in [1,T] \wedge Festive[t] = F$
      $season_{\mu,i,f}(hour) \leftarrow median(v_{i,f}(t_{hour}))$
      $season_{\sigma,i,f}(hour) \leftarrow sd(v_{i,f}(t_{hour}))$
      $t_{hour_F} \leftarrow t \mid (t - 1) \bmod freq = hour, t \in [1,T] \wedge Festive[t] = T$
      $season\_Festive_{\mu,i,f}(hour) \leftarrow median(v_{i,f}(t_{hour_F}))$
      $season\_Festive_{\sigma,i,f}(hour) \leftarrow sd(v_{i,f}(t_{hour_F}))$
      $laggedSeries \leftarrow differences(v_{i,f}, lag, differences)$
      $diff_{\mu,i,f}(hour) \leftarrow median(laggedSeries[t_{hour}])$
      $diff_{\sigma,i,f}(hour) \leftarrow sd(laggedSeries[t_{hour}])$
    end for
    Compute $CDF_{i,f}$ from $v_{i,f}$
  end for
  return all computed $parameters_{i,f,t}$
end function

After computing parameters and CDFs, the training algorithm computes and stores p-values for each value contained in the training dataset, as seen in algorithm 2. Note that to achieve better results during the method weighting process, the anomaly_threshold parameter should be set lower than the one we want to use later in the algorithm. By doing this, we make sure that the priorities are uniform in the area around the decision threshold, and not just after it. For example, in our implementation we used an anomaly threshold of 90% during training, instead of the 95% and 99% thresholds we used in the algorithm.

Algorithm 2 P-value learning

function MethodTraining(D, anomaly_threshold, parameters)
  for all time t **do**
    for all nodes i **do**
      for all features f **do**
        for all methods m **do**
          $\text{fit}_{i,m,f} \leftarrow \text{fit} \cup \{\text{Method}_m(\text{parameters}_{i,f,t})\}$
        end for
      end for
    end for
  end for
  for all nodes i **do**
    for all features f **do**
      for all methods m **do**
        $\text{method\_weight}_{i,m,f} \leftarrow \text{MethodWeighting}(\text{fit}_{i,m,f},$ anomaly_threshold)
      end for
    end for
  end for
end function

Finally, the stored p-values are used to find the weights to apply to each method of p-value computation, as shown in algorithm 3.

Algorithm 3 Method weights learning

```
function MethodWeighting(fits, anomaly_threshold)
    P[] ← 1 - fits[]
    Compute iCDF ← inverse CDF of P
    error ← | iCDF(anomaly_threshold) - anomaly_threshold |
    weight ← 1
    best_weight ← 1
    best_error ← error
    while (error ≥ min_error ∨ iteration ≤ max_iteration) do
        P ← WeightPriority(P, weight, base)
        Compute iCDF ← inverse CDF of P
        error ← | iCDF(anomaly_threshold) - anomaly_threshold |
        if (iCDF(anomaly_threshold) ≥ anomaly_threshold) then
            weight ← max(0, weight - error)
        else
            weight ← min(1, weight + error)
        end if
        if (error ≤ best_error) then
            best_error ← error
            best_weight ← weight
        end if
    end while
    return best_weight
end function
```

The time complexity of the training phase is linear with respect to nodes, feature and hours in the data. In particular, the computation of seasonality and other parameters, including CDFs, in algorithm 1 is $O(N \times F \times T \times \log(T))$. The method weighting function 3 is $O(T \times \log(T))$. In both previous computation, the $\log(T)$ term derives from the CDF calculation that involves quicksort. The p-value learning is $O(N \times F \times T \times O(\text{methods}) + N \times F \times M \times T \times \log(T))$, $O(\text{methods})$ being the complexity of the used methods.

In general, the training phase has a time complexity of $O(N \times F \times T \times M \times \log(T))$.

5.3 MuSTF

MuSTF analyzes the dataset using the parameters learned during training.

5.3.1 Removal of NA

MuSTF can operate in dataset with missing values. One of the following strategies can be adopted:

- Ignore: when a missing value is processed, it is skipped and no p-value is generated;
- Last value: when a missing value is processed, the algorithm keeps valid the p-value generated in the previous hour;
- Average: when a missing value is processed, it is substituted with the expected seasonal value of that feature;
- Error: when a missing value is processed, stop the algorithm;
- Impute: at the beginning of the algorithm, each missing value is substituted with the average between the values that preceded it and succeeded it; this strategy is only possible if the dataset is all available at the beginning of execution.

In our tests we used the impute strategy, since we have the full dataset available. The impute strategy is shown in algorithm 4. We indicate as D_{NA} the dataset with NA, and D the dataset without NA.

The NA removal has time complexity of O(NA $\times$ T), where NA is the number of NA in the data. In the worst case (all NA) this is O(N $\times$ F $\times$ T).

Algorithm 4 NA Imputation

function NAIMPUTE(D_{NA})
  for all $v_{(i,f)}$ in D_{NA} **do** $v'_{(i,f)} \leftarrow v_{(i,f)}$
    for all $t_{NA} \mid v_{(i,f)}(t_{NA}) = NA$ **do**
      $t_b \leftarrow \max(t \mid v_{(i,f)}(t) \neq NA \wedge t < t_{NA})$
      $t_a \leftarrow \min(t \mid v_{(i,f)}(t) \neq NA \wedge t > t_{NA})$
      if $\nexists\ t_a \vee \nexists\ t_b$ **then**
        Ignore this NA
      else
        $dist_{a,b} \leftarrow t_a - t_b$
        $vdist_{a,b} \leftarrow v_{(i,f)}(t_a) - v_{(i,f)}(t_b)$
        for all NA_t in $[t_b, t_a]$ **do**
          $v'_{(i,f)}(NA\_t) \leftarrow v_{(i,f)}(t_b) + (NA\_t - t_b) \times (\frac{vdist_{a,b}}{dist_{a,b}})$
        end for
      end if
    end for
  end for
  return $D \leftarrow \bigcup v'_{(i,f)}$
end function

5.3.2 Node Scoring

In the first phase of the algorithm, each node is attributed an anomaly score independently from its neighbors. The process is shown in algorithm 5. The implementation of Fisher's method, which generates a χ^2 variable from p-values and returns their combined priority is shown in algorithm 6.

The time complexity of node scoring is O(N × F × O(methods) + N × F × 2M).

Algorithm 5 Node scoring

function NODE(v_i(t), $parameters_i$)
  for all m in [1:M] **do**
    $p_{i,f}^{(m)}(t) \leftarrow Method_m(parameters)$
  end for
  Univariate $\leftarrow$ $P_{i,f}(t)$ $\leftarrow$ FISHERCOMBINE(1 - WEIGHTPRIORITY(1 - $p_{i,f}^{(m)}(t)$, $weight_{i,f}^{(m)}$,default)), $\forall m \in [1:M]$
  Multivariate $\leftarrow$ $P_i^{(m)}(t)$ $\leftarrow$ FISHERCOMBINE(1 - WEIGHTPRIORITY(1 - $p_{i,f}^{(m)}(t)$, $weight_{i,f}^{(m)}$, default)), $\forall f \in [1:F]$
  return $P_i(t)$ $\leftarrow$ FISHERCOMBINE($P_{i,1}(t)$,$P_{i,2}(t)$, ... , $P_{i,F}(t)$)
end function

Algorithm 6 Fisher method

function FISHERCOMBINE(pvalues)
  Input: a vector of pvalues
  Output: a priority
  for all p $\in$ pvalues **do**
    log_val $\leftarrow -2 * \log(p)$
  end for
  FisherChi $\leftarrow \sum_{\forall p}$ log_val
  FisherPriority $\leftarrow$ P(FisherChi)
  return FisherPriority
end function

The Fisher's method is linear with respect to the number of input p-values.

5.3.3 Priority Weighting

The priority weighting works as described in previous chapter. The unweighted priorities are inputted, along with a weight and the default priority. The weights represents our confidence in the p-value. The function returns the weighted priorities.

The priority weight function runs in constant time.

Algorithm 7 Priority Weighting

function WEIGHTPRIORITY(priority,weight,default)
  pow ← (default - priority)$^{\frac{1}{weight}}$ + 1 - default
  dist ← pow - 1 + priority
  if priority > default **then**
    wpval ← pow - (dist * $\frac{|default-priority|}{1-default}^{\frac{1}{weight}}$)
  else
    wpval ← pow - (dist * $\frac{|default-priority|}{default}^{\frac{1}{weight}}$)
  end if
  return 1 - wpval
end function

Default priority discovery

The default priority has to be found for each priority weighting procedure: this is a lengthy process because the default priority is found by iterating and testing many values to check which has the minimal error. In algorithm 8 we show the code for a single iteration.

Instead of iterating this process at every time, we find the distribution of the values assumed by the default priority by iterating the process with random inputs for many times: we find that the values of the default priority are distributed in a gaussian curve, with mean 0.6217, and 95% of the values are within [0.584, 0.684]. This allows us to empirically reduce the interval of search for the default priority.

In fact, a further optimization can be to just use the mean value of the default priority. While not granting optimal results, the influence of the default priority is low within the found interval: the difference in the weighted priority computation when using the maximum or minimum value is less than 0.015 of the priority value.

The default priority discovery process is $O(M^2)$ for the default priority of methods priority, $O(k^2)$. Because this process is slow, it is done only once at the beginning of the algorithm or at the end of the training phase.

Algorithm 8 Default Priority Discovery

function DEFAULTDISCOVERY(priorities, weights, start, iterations, interval)
default$_i$ ← start, $\forall i \in [1 : N]$
for it ∈ [1:iterations] **do**
for all i in [1:N] **do**
wp$_i$ ← WEIGHTPRIORITY(p$_i$, w$_i$, default$_i$)
end for
for all i in [1:N] **do** best_default$_i$ ← -1 least_error$_i$ ← inf
for d **do**default$_{temp}$ ∈ Interval
fisher_without ← FISHERCOMBINE(1 - wp$_j$), j ∈ [1:N]\ i
fisher_with ← FISHERCOMBINE({1 - wp$_j$} ∪ {1 - default$_{temp}$}), j ∈ [1:N]\ i
error ← |fisher_without - fisher_with|
if error < least_error **then**
least_error$_i$ ← error
best_default$_i$ ← default$_{temp}$
end if
end for
end for
for all i in [1:N], current_error$_i$ ¿ least_error$_i$ **do**
default$_i$ ← best_default$_i$
current_error$_i$ ← least_error$_i$
end for
end for
return default
end function

5.3.4 Linear Fisher Scan

Algorithm 9 implements the search of the most anomalous subsets in the given p-value. This can be used during node scoring to identify the most anomalous subsets of features, or during cluster scoring to find the most anomalous subsets of nodes.

Algorithm 9 Linear Fisher Scan

function LINEARFISHERSCAN(priorities, weights, [default priorities])
Input: unweighted priorities, their weight, optional default prioritys values, indication of which results to report
Output: the general priority and the indices of priorities used
if ∄ default priorities **then**
default priorities ← DEFAULTDISCOVERY(priorities, weights, 1, 10)
end if
for all $p_i \in$ priorities **do**
$wp_i \leftarrow$ WEIGHTPRIORITY(p_i, w_i, b_i)
end for
priority_order ← SORTORDER(wp)
for size ∈ [1,|priorities|] **do**
group_priority$_{size}$ ← FISHERCOMBINE({1 - wp_1, ..., 1 - wp_{size}})
end for
best_size ← $\arg\max_{size}$(group_priority$_{size}$)
return nodes: priority_order[1,..., best_size], priority: group_priority$_{best_size}$
end function

function SORTORDER(vector)
Sorts the given vector, from the highest value to the lowest
return the order of the sorting of values of the vector
end function

Despite its name, LTSS is actually quasilinear since it contains quicksort, that is O(k × log(k)).

5.3.5 Cluster Scoring

In the third phase of the algorithm, each node evaluates the cluster of neighbors. We also memorize the number of times a node is found anomalous and the nodes involved in each anomaly to aggregate the results over time.

The group scoring is O(N × k × log(k)).

Algorithm 10 Cluster Scoring

function CLUSTER(priorities, neigh_weights$_i$, anomaly_threshold)
votes$_i$(t) := number of times node i is found anomalous
clusters(t) := any cluster found anomalous
for all node $i \in$ D **do** Get neighbours of i neigh_i.
anomalous_cluster $\leftarrow$ LINEARFISHERSCAN(priorities, neigh_weights$_i$, default priorities)
if anomalous_cluster.priority > anomaly_threshold **then**
clusters(t) $\leftarrow$ clusters(t) $\cup$ anomalous_cluster.nodes
for j in anomalous_cluster.nodes **do**
votes$_j$(t)++
end for
return anomalous_cluster, votes(t)
end if
end for
end function

5.3.6 Node Weighting

We update the similarity weights between nodes according to the unweighted priorities found in the previous hours.

Algorithm 11 Node weighting

function NODEWEIGHTING(priorities, neigh_weights)
Gets the memorized windows of the last unweighted priorities.
Updates the windows with the current new priority
update_rate $\leftarrow$ 0.2
for all neighbors n **do**
dissimilarity $\leftarrow \frac{\sum_{0<mem<max_{mem}} |priority(t-mem)-priority_{neighbor}(t-mem)|*discount^{mem}}{\sum_{0<mem<max_{mem}} discount^{mem}}$
neigh_weights$_n$ $\leftarrow$ neigh_weights$_n$*(1-update_rate) + (1 - dissimilarity)*update_rate
end for
return neigh_weights
end function

The node weighting time complexity is O(N $\times$ k).

5.3.7 Complete structure

We show in algorithm 12 the invocation of MuSTF, built upon the functions shown in this chapter.

Algorithm 12 MuSTF

```
function MuSTF(trained parameters, D, k, anomaly_threshold)
    Compute k neighborhoods with knn, using x and y vectors
    for time t ∈ T do
        for node i ∈ N do
            priority_i ← Node(v_i(t))
        end for
        for node i ∈ N do
            votes, clusters ← Cluster(priority, neigh_weights_i, anomaly_threshold)
            neigh_weights_i ← NodeWeighting(priorities, neigh_weights_i)
        end for
        anomaly_nodes(t) ← arg votes(t) > number
        clusters(t) ← Merge anomaly_clusters with common nodes
    end for
    return anomaly_nodes, clusters
end function
```

The complexity of the whole algorithm (not counting training) is O(N × F × M × k × log(k)) for each iteration. For the full dataset the same complexity is multiplied by T and is O(T × N × F × M × k × log(k))

Chapter 6

Experimental evaluation of MuSTF

In this chapter, we describe the performance of MuSTF and compare it to the performance of the state-of-the-art. We first describe the dataset used to evaluate the performance. We then describe the performance metrics adopted and the expected results. Finally, we present the performance and analyze it in comparison to the performance of the benchmark state-of-the-art algorithms.

6.1 Description of the dataset

The dataset used in this research contains quality of service data collected from roughly 6500 telecommunication towers in Piemonte; the dataset is composed of hourly measurements and refers to a month of service. The collected measurements are five features, two of them related to data downlink service, three of them related to voice service. In figures 6.1 through 6.5 we show examples of time series describing each feature for a single node. In figure 6.6 through 6.10 we show the average trend of each feature over all nodes. We grouped towers with the same values of longitude and latitude together in a single virtual location called node, and used the mean of the values collected in the grouped tower as the measurement of each node. The resulting dataset is composed of roughly 1000 nodes.

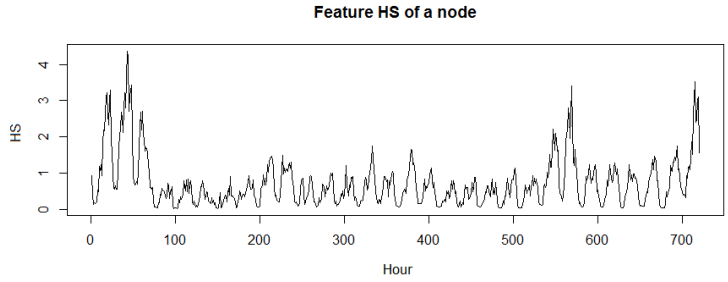

Figure 6.1: Time series representing the Erlang High Speed (HS) measurement in a node.

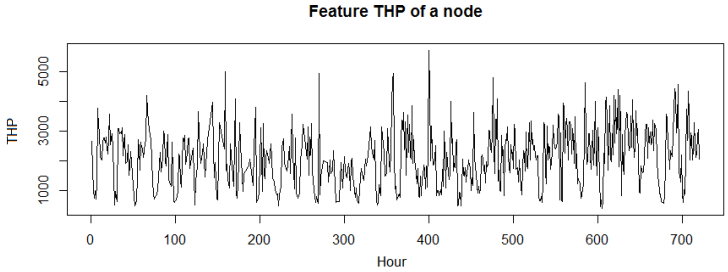

Figure 6.2: Time series representing the Throughput (THP) measurement in a node.

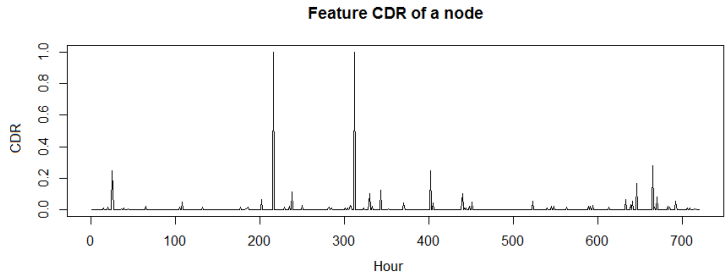

Figure 6.3: Time series representing the Call Drop Rate (CDR) measurement in a node.

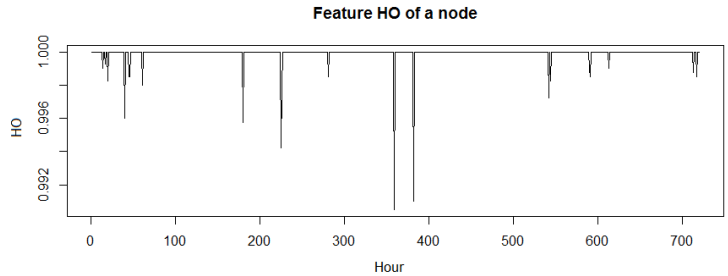

Figure 6.4: Time series representing the Handover Rate (HO) measurement in a node.

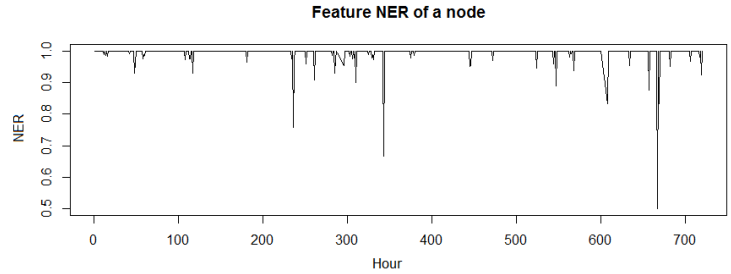

Figure 6.5: Time series representing the Network Effectiveness Rate (NER) measurement in a node.

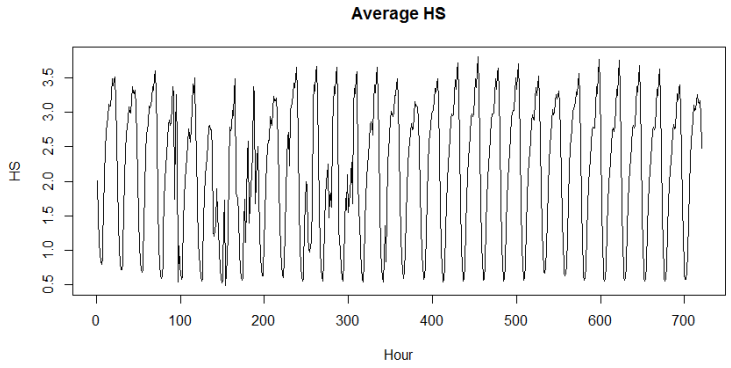

Figure 6.6: Average values of HS in the dataset for each hour.

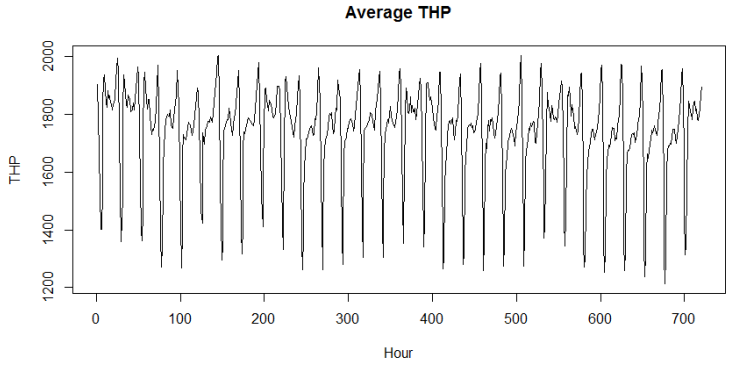

Figure 6.7: Average values of THP in the dataset for each hour.

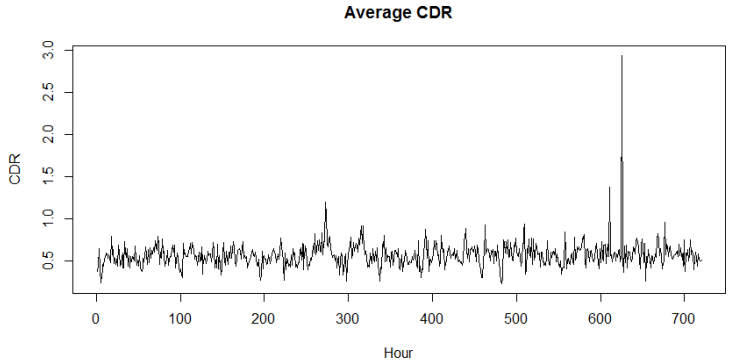

Figure 6.8: Average values of CDR in the dataset for each hour.

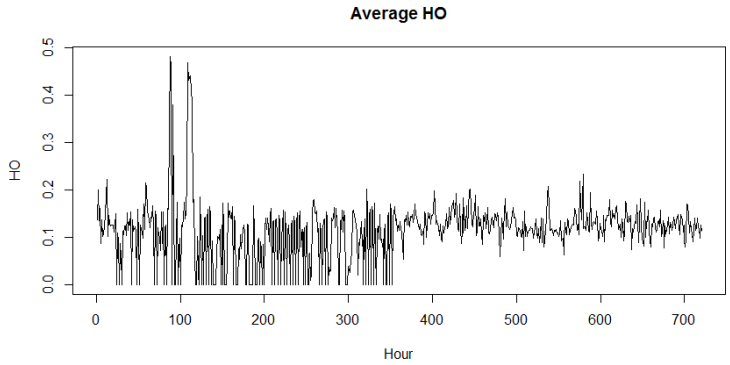

Figure 6.9: Average values of HO in the dataset for each hour.

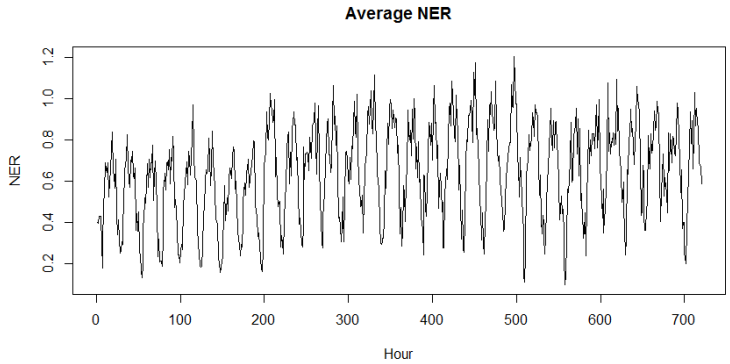

Figure 6.10: Average values of NER in the dataset for each hour.

6.2 Performance metrics

The performance metric we considered is the number of anomalous nodes reported by an algorithm in each hour. In MuSTF, we control the number of anomalous nodes using two parameters: the anomaly threshold of a cluster and the minimum vote of a node. The anomaly threshold is the minimum score over which a cluster of nodes is considered anomalous. The vote of a node is the number of anomalous clusters that contain that node. We set a minimum vote threshold to exclude from the results nodes that were only found in a small number of clusters. In STCOD, the number of anomalous nodes is determined by setting the two thresholds of self similarity and neighborhood similarity. MFSS does not have any parameters, so its results are fixed.

6.2.1 Results of MuSTF

We first executed MuSTF using an anomaly threshold of 95%. In figure 6.11 we show the number of nodes found anomalous in each hour, using a minimum vote threshold of 1. In figure 6.12 we show the same results but with a minimum vote threshold of 20.

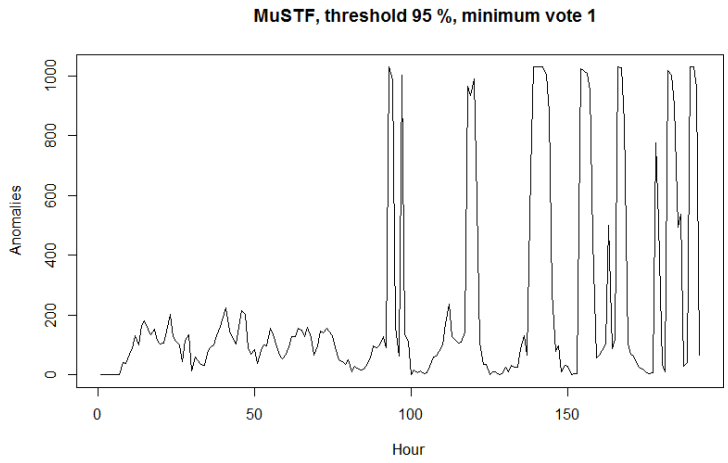

Figure 6.11: Number of anomalies detected by MuSTF with anomaly threshold 95% and minimum vote threshold 1.

In figure 6.13 and figure 6.14 we show the performance of MuSTF with anomaly threshold 99%, and a minimum vote threshold of 1 and 20 respectively.

By comparing the performance of MuSTF we notice that increasing the

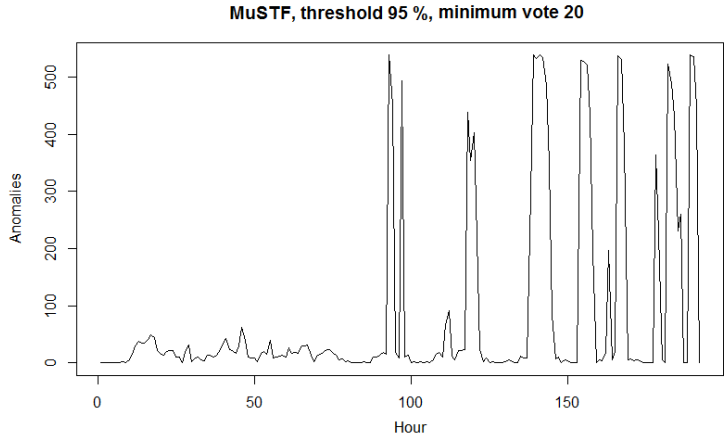

Figure 6.12: Number of anomalies detected by MuSTF with anomaly threshold 95% and minimum vote threshold 20.

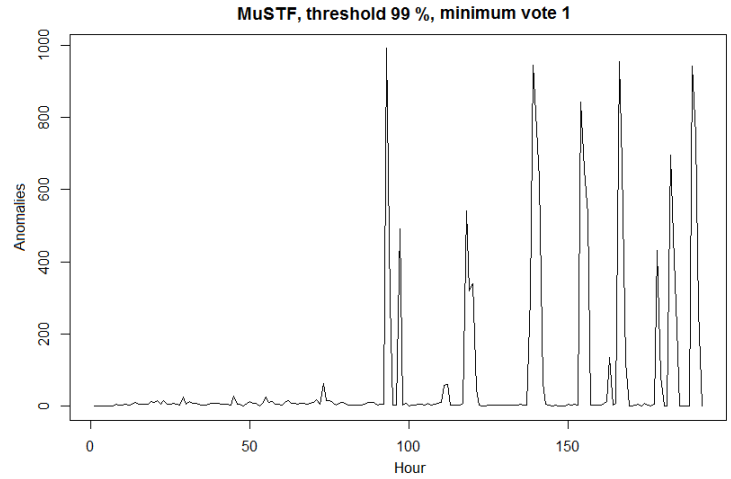

Figure 6.13: Number of anomalies detected by MuSTF with anomaly threshold 99% and minimum vote threshold 1.

value of the minimum vote threshold reduces the number of reported anomalous nodes but in both configuration the same peaks indicating anomalies are still clearly distinguishable. The effects caused by changing the anomaly threshold are more prominent. Firstly, the number of anomalies reduce drastically in most hours, but during the peaks of anomalous events the number of nodes remains the same. We note that a previously undistinguishable anomaly (November 4th, 00:00) can be seen in the results. Secondly, the length of detected anomalous events (the width of the peaks in the fig-

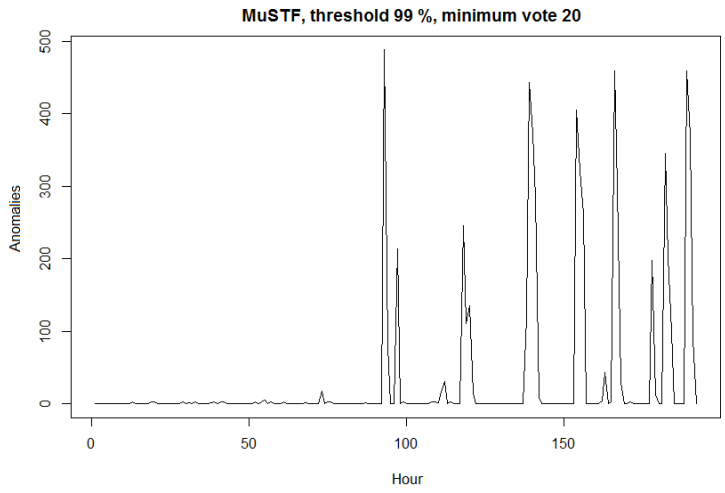

Figure 6.14: Number of anomalies detected by MuSTF with anomaly threshold 99% and minimum vote threshold 20.

ures) reduces when using a high anomaly threshold. The explanation of the decreased count of anomalies in the 99% case with respect to the 95% is obvious: we are now accepting less cluster as anomalous and vote decreases. The decrease in the length of detected events can be explained by examining the methods used to discover anomalies. The most effective method in our data was the lagged difference Δ_i. In general, the lagged difference is only able to discover anomalies events as long as their lag. Furthermore, the detection power of Δ_i decreases for high values of i. For example, Δ_1 is optimal to discover sudden anomalies, and is therefore very sensitive to the beginning of an anomalous events. However Δ_1 can only detect anomalies in the same hour they start. Δ_3 and Δ_5 can detect anomalies that last for 3 and 5 hours, but with lower accuracy. In MuSTF we use Δ_1, Δ_5 and Δ_5: the maximum length of anomaly detected by these measures is 5 and the detection is higher at the beginning because of the higher accuracy of Δ_1. Other anomalies are discovered using the empirical CDF. This second method is punctual instead of window-based, therefore CDF effectiveness is constant throughout an anomalous event. However, we note that the number of false positives generated by this method during the training phase is quite high, therefore the CDF method has a weight lower than the one of the lagged differences. Because of this anomalies detected only by CDF tend to have a lower score, and when using a high anomaly threshold the most obvious anomalies are the ones detected by the Δ_i.

The highest peaks of anomalies found by MuSTF are coherent with an

analysis of the variance and mean of the features in the dataset. In figure 6.15 we show a matrix representing the votes of nodes computed by MuSTF using 95% anomaly threshold.

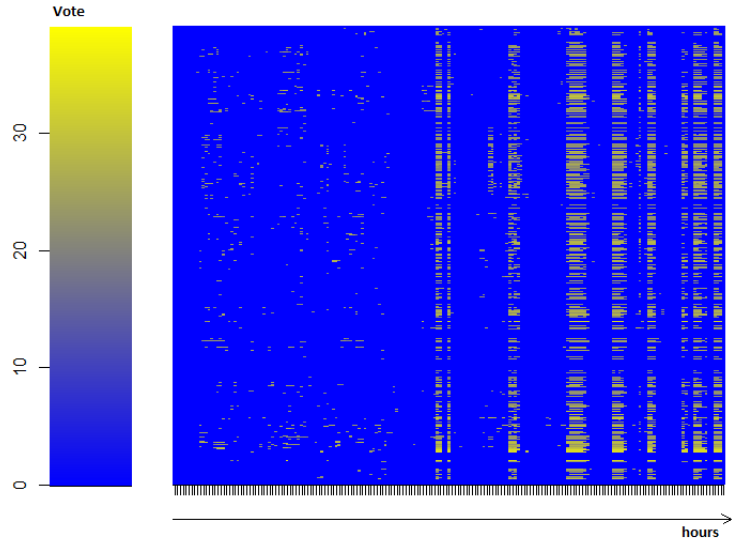

Figure 6.15: In this image each node is represented by a row and each hour by a column of the table. A yellow color indicates a high vote for a node in a hour, while blue color indicates normal behavior.

Clustering anomalies

To analyze the anomalies reported by MuSTF, we clustered the anomalies to find the basic shapes that describe the behavior of each feature during an anomalous event. Before collecting the anomalies from the dataset, we normalized them with respect to their season. An anomaly was represented by five windows, each containing the values of a feature before, during and after the anomalous event. In figure 6.16 we show two examples of anomalies represented by windows.

To cluster the anomalies we first computed a similarity matrix containing the similarity between each couple of anomalies. The similarity measure used was the correlation between the windows of the same feature. The computation of the similarity matrix is a very intensive task, requiring $O(N^2)$ time for N anomalies, thus we sampled a subset of anomalies instead of using all of them. After computing the similarity matrix, we used it in the k-medoids algorithm to find anomalies with similar shapes. To find the best number of shapes of anomalies k we iterated the clustering algorithm using different values of k and chose the value maximizing intra-cluster similarity.

After finding the k clusters of anomalies, we compute the means of their windows and plot them as the archetypal shapes of found anomalies. In figure 6.17 we show an example of archetypal shape derived by a cluster of anomalies.

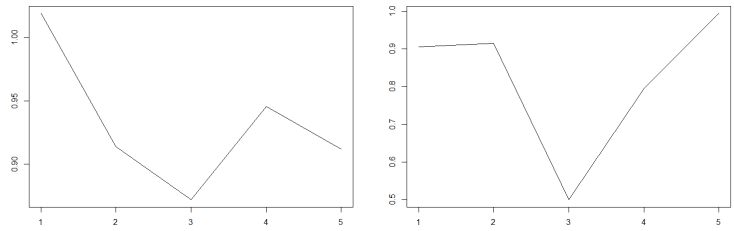

Figure 6.16: Example of two anomalous windows found in the real-world dataset. On the vertical axis the original values are represented relative to their seasonal value. For example, a value of 0.5 on the vertical axis means that the real value is half the one expected in that period.

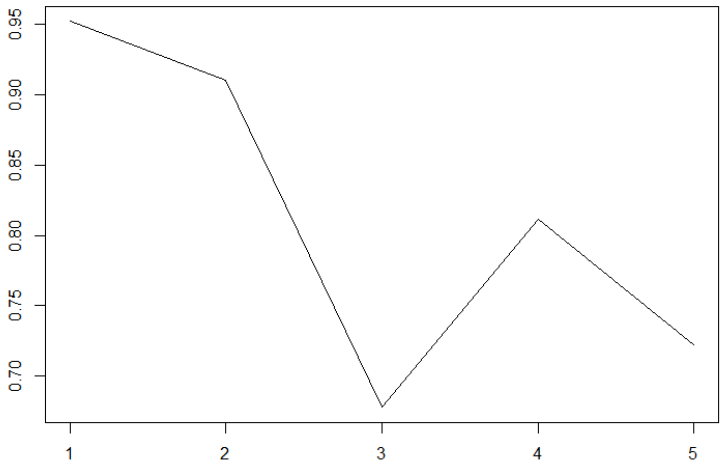

Figure 6.17: Example of archetypal shape resulting from the clustering of the anomalies in figure 6.16 along with similar others.

Analysis of archetypal anomalies

In figure 6.18 through 6.22 we show the archetypal shapes of anomalies found by clustering the anomalous windows reported by MuSTF. We note that most anomalies are particularly evident in the trend of the first feature (the HS), with some influence of the second feature (THP) and the third (CDR).

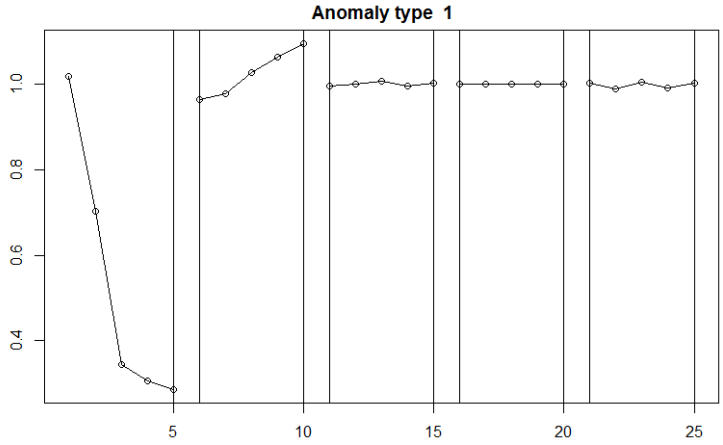

Figure 6.18: First archetypal anomaly found, characterized by a sudden decrease in the first feature value that keeps decreasing after the detection.

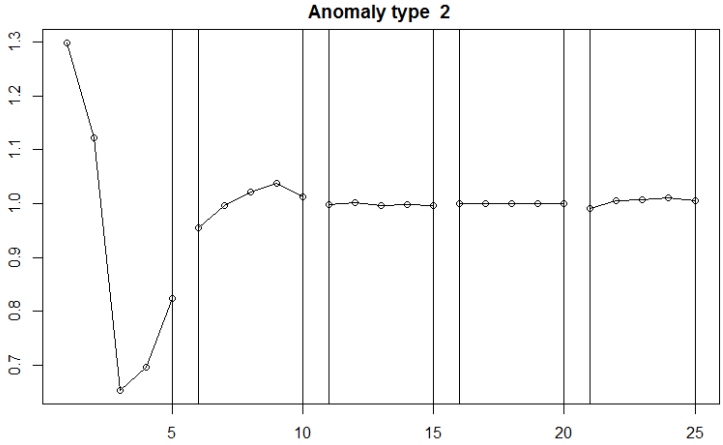

Figure 6.19: Second archetypal anomaly found. In this case the value of the first feature decreases suddenly but the starting point is a value higher than the seasonal value. After the detection it rises towards values within expectations.

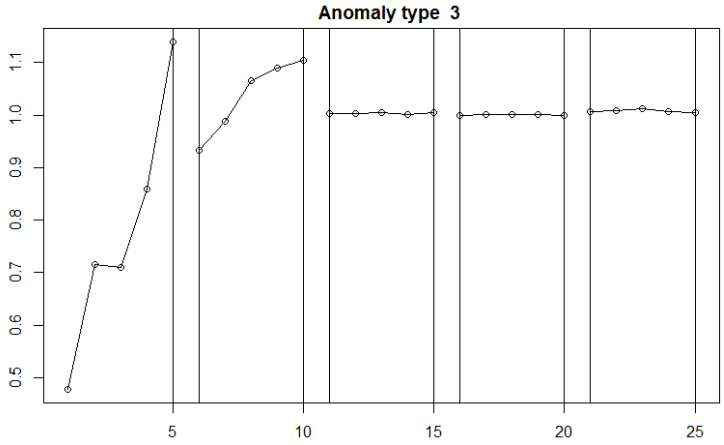

Figure 6.20: Third archetypal anomaly found. The value of the first feature increases from low values to expected levels, but is stable for a period before restarting in its increasing trend.

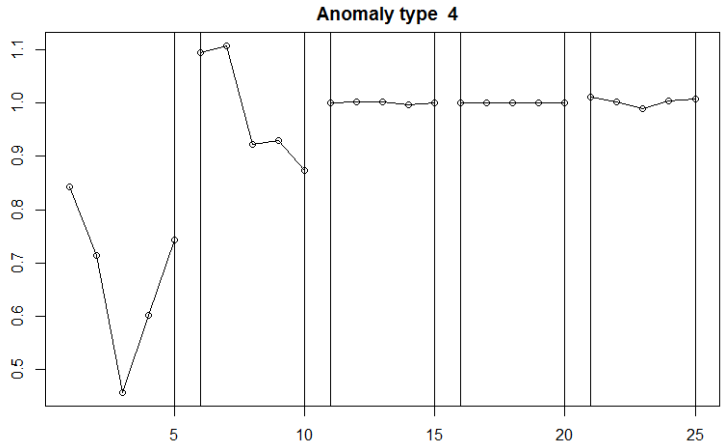

Figure 6.21: Fourth archetypal anomaly found. The first feature decreases as in the second type of anomaly, but from a lower starting point. In this case the second feature (THP) shows sign of sudden decrease after assuming high values.

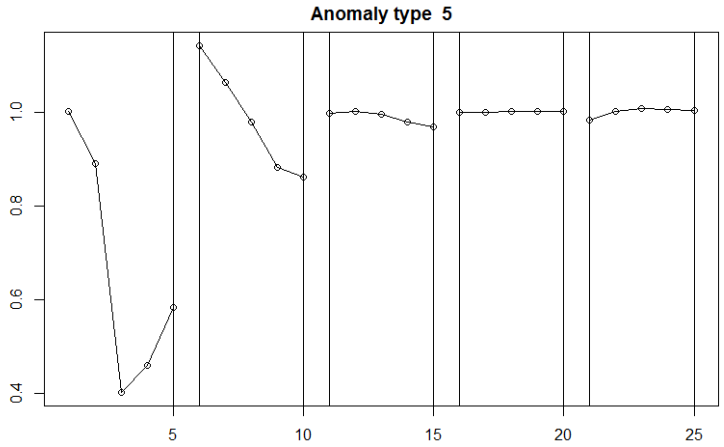

Figure 6.22: Fifth archetypal anomaly found. Similar to the fourth type but the first features starts from higher values. The same situation is seen in the second feature while the fifth feature shows no sign of anomalies. The third feature (CDR) shows signs of a decreasing trend.

Spatial clusters of anomalies

MuSTF distinguishes different groups of nodes by their spatial location and the similarity of their anomalous behavior. We show in figures 6.23 and 6.24 the clusters of anomalies relative to two hours of the dataset.

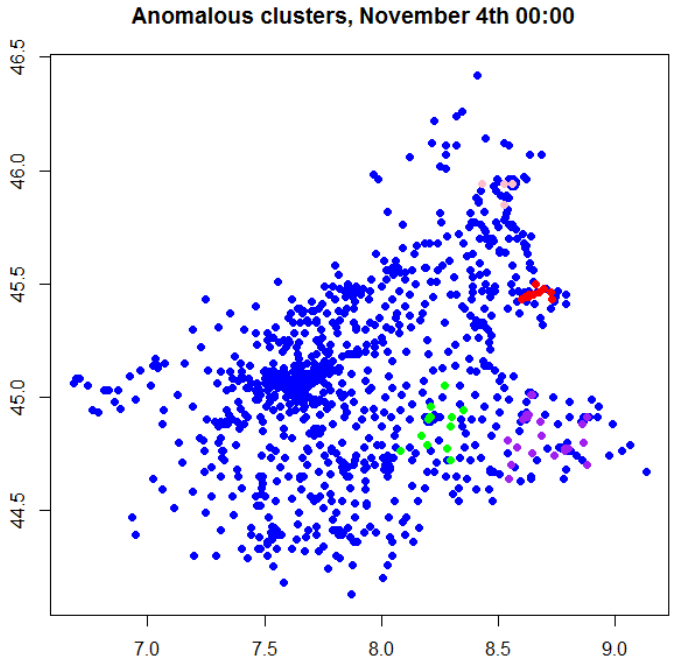

Figure 6.23: Spatial clusters of anomalies in Piemonte, November 4th 00:00

In the first example, in figure 6.23, nodes are grouped in four anomalous clusters, each identified by a different color. We note that the clusters have different number of members and different density.

In the second example, in figure 6.24, we can observe the logical division of clusters. We note that the distance between the purple cluster and the pink one is smaller than the distance between some of the nodes in the red cluster. This exemplifies that the clusters are grouped according to their similarity thanks to MuSTF neighbor weighting strategy. In this particular case the pink cluster is centered in the Turin metropolitan area while the purple clusters is outside the city, which explains the difference in the mobile network usage. We can see that in this case the correlation between an area prevails over geographical distance in the definition of the cluster's boundaries We note however that the pink cluster is likely the merge between

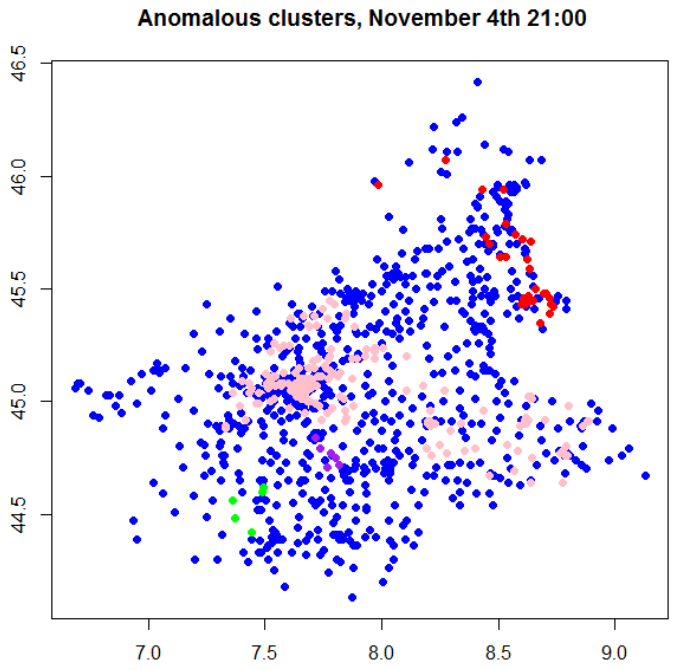

Figure 6.24: Spatial clusters of anomalies in Piemonte, November 4th 21:00

two or more different clusters. We believe this to be an issue of the merging procedure that does not analyze the correlation in-between the cluster.

6.2.2 Performance of benchmark algorithms

We compare here the performance of two benchmark algorithms: STCOD and MFSS.

The performance of MFSS is low because of the lack of evident anomaly score. We note in figure 6.25 that in the hour where MuSTF detected the highest anomaly (4th November, 20:00) there is a peak, however it is hardly distinguishable from the rest of the result. Furthermore we note that even in the most anomalous peak MFSS finds only 20 anomalous nodes. It is evident that the FSS family of algorithms is not effective in this application, even though a similar approach is used by our algorithm.

STCOD is sensitive to seasonality as can be seen by the periodic increase in the number of detected anomalies in figure 6.26. The peaks of anomaly detected by STCOD are correlated to the peaks detected by MuSTF: in fact most peaks detected by MuSTF at 99% are detected by STCOD, with some

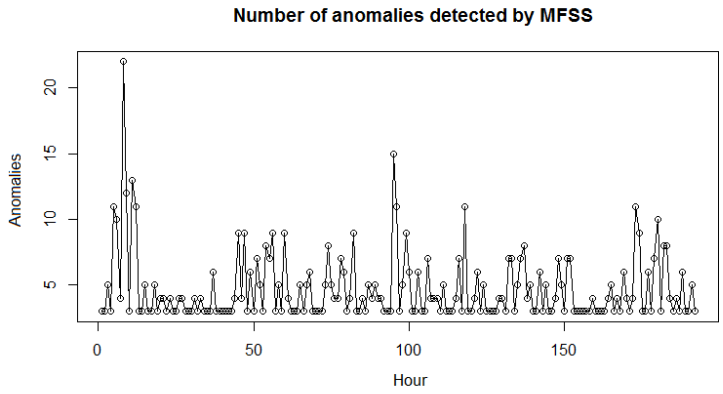

Figure 6.25: Number of anomalies detected by MFSS.

more noise in the latter case. A clear improvement in the detection lies in the timeliness of detection. STCOD detects anomalies later than MuSTF because of its larger window size.

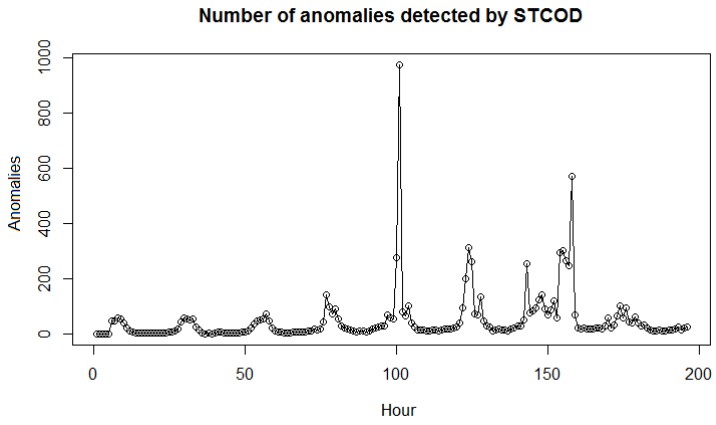

Figure 6.26: Number of anomalies detected by STCOD

6.3 Conclusions

MuSTF overcomes the challenges of multivariate spatio-temporal anomaly detection. MuSTF complexity is linear with respect to the number of nodes and the number of features it processes. Our tests demonstrated that MuSTF can operate on-line, analyzing data of an hour from the whole network in a couple of seconds.

In our experiments, MuSTF performed at least as well as state-of-the-art algorithms in detecting anomalies in the same data. Furthermore, the comparison against STCOD and MFSS shows that MuSTF is less sensitive to noise, and the hours of major anomalies are clearly distinguishable from normal hours: this result was achieved by adopting STCOD voting strategy and improving it to remove outliers from the results. Another particularly important improvement of MuSTF is the shorter delay in detection: our algorithm is able to detect the same anomaly several hours sooner than the state-of-the-art algorithms. Finally, MuSTF performance is less sensitive to small changes in the user-set parameters than other state-of-the-art algorithm, which relieves the user from having to discover the best settings to find acceptable results. We also showed how MuSTF is able to identify the spatial cluster that is involved in the anomaly, distinguishing clusters of nodes that belong to areas with different behavior even when close to each other.

Future improvements

MuSTF can be improved in several ways. Firstly, the learning process we adopted for the weights and baselines is approximate. For the priority weighting in particular, the process is only able to achieve uniformity in the area of anomalous behavior. This imprecision may lead to overestimating the combined priority of a feature or node. In our training process, setting the anomaly threshold lower than the one used in the algorithm was effective in reducing the effect of this overestimate. Finding a new strategy to transform the original p-values in an exact uniform distribution may improve further the precision of the Fisher's method results.

Secondly, the strategy used to define a neighborhood could be improved. In MuSTF the neighborhood of a node is defined at the beginning of the execution: if a node is found dissimilar from other neighbors its weight is lowered, but its neighborhood remains the same. Finding a way to update the neighborhood without an excessive burden on computation time could be an improvement in the cluster scoring phase.

Thirdly, other methods of computing p-values could be tested. In particular, the methods used in this work showed high precision on features such as THP and HS, but were less effective on features as CDR. Introducing other methods for generating p-values in our framework might improve its performance.

MuSTF can be adapted to run in a distributed manner, since both the node and cluster scores are computed separately for each node. These two

phases are also the most time intensive, so we believe that a parallelized or distributed implementation could be much faster than our own.

In this work we examined the performance of MuSTF in comparison to other state-of-the-art algorithms. While we showed that MuSTF is an improvement over the performance of these algorithms, we believe that using datasets with labeled anomalies could lead to a more accurate measurement of the improvements introduced by MuSTF. Given a labeled dataset, we also believe that a supervised approach could be adopted during the training phase to compute the precision of the specific methods in an exact way.

Bibliography

[1] Fisher R. A. *Statistical Methods for Research Workers.* 1932.

[2] N. M. Laird A. P. Dempster and D. B. Rubin. *Maximum Likelihood from Incomplete Data via the EM Algorithm.* 1977.

[3] McGregor A. Phillips J.M. Venkatasubramanian S. Zhu Z. Agarwal, D. *Spatial Scan Statistics: Approximations and Performance Study.* 2006.

[4] Charu C. Aggarwal. *Outlier Analysis.* 2013.

[5] Richard Bellman. *Dynamic Programming.* 1957.

[6] Yufeng Kou Chang-Tien Lu, Dechang Chen. *Detecting Spatial Outliers with Multiple Attributes.* 2003.

[7] Lancaster H. D. *The combination of probabilities: an application of orthonormal functions.* 1961.

[8] Huanian Zheng Daniel B. Neill, Edward McFowland III. *Fast subset scan for multivariate event detection.* 2012.

[9] Manuel Davy and Simon Godsill. *Detection Of Abrupt Spectral Changes Using Support Vector Machines: An Application To Audio Signal Segmentation.* 2002.

[10] Calvin Chow Dit-Yan Yeung. *Parzen-Window Network Intrusion Detectors.* 2002.

[11] James Adcock Duncan J. Murdoch, Yu-Ling Tsai. *P-values are random variables.* 2008.

[12] J.C. Dunn. *A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters.* 1973.

[13] Daniel B. Neill Edward McFowland III, Skyler Speakman. *Fast Generalized Subset Scan for Anomalous Pattern Detection.* 2013.

[14] Wei Liu Elizabeth Wu and Sanjay Chawla. *Spatio-temporal Outlier Detection in Precipitation Data*. 2010.

[15] Zhi-Hua Zhou Fei Tony Liu, Kai Ming Ting. *Isolation Forest*. 2008.

[16] Henning Sanneck Gabriela F. Ciocarlie, Szabolcs Novázki. Detecting anomalies in cellular networks using an ensemble method. 2013.

[17] Michael F. Goodchild. *The Validity and Usefulness of Laws in Geographic Information Science and Geography*. 2004.

[18] D. M. Hawkins. *Identification of Outliers*. 1980.

[19] F. L. Hitchcock. *The expression of a tensor or a polyadic as a sum of products*. 1927.

[20] IDC, editor. *IDC Big Data and Analytics Conference*, 2014.

[21] Zhengyuan Zhu Jeff Terrell, Lingsong Zhang. *Multivariate SVD Analyses For Network Anomaly Detection*. 2005.

[22] Lawrence O. Hall Jonathon K. Parker. *Accelerating Fuzzy-C Means Using an Estimated Subsample Size*. 2014.

[23] Graham Williams-Peter Milne Kenji Yamanishi, Jun-ichi Takeuchi. *Online Unsupervised Outlier Detection Using Finite Mixtures With Discounting Learning Algorithms*. 2004.

[24] Matthew V. Mahoney and Philip K. Chan. *Learning Rules for Anomaly Detection of Hostile Network Traffic*. 2003.

[25] Raymond T. Ng Jörg Sander Markus M. Breunig, Hans-Peter Kriegel. *LOF: Identifying Density-Based Local Outliers*. 2000.

[26] Kanoksri Sarinnapakorn Li Wu Chang Mei-Ling Shyu, Shu-Ching Chen. *A Novel Anomaly Detection Scheme Based on Principal Component Classifier*. 2003.

[27] Zhongbo Wu Min Wang. *Spatio-temporal Correlation based Outlier Detection Algorithm in Sensor Network*. 2010.

[28] Jinyun Fang Nan Wang, Jizhong Han. *An Anomaly Detection Algorithm Based on Lossless Compression*. 2012.

[29] Daniel B. Neill. *Fast subset scan for spatial pattern detection*. 2011.

[30] Numenta. *Hierarchical Temporal Memory*. 2011.

[31] Numenta. *The Science of Anomaly Detection.* 2014.

[32] E. S. Page. *Continous Inspection Scheme.* 1954.

[33] E. Parzen. *On the estimation of a probability density function and the mode.* 1962.

[34] Annick M. Leroy Peter J. Rousseeuw. *Robust Regression and Outlier Detection.* 1987.

[35] Allen T. Craig Robert V. Hogg. *Introduction to Mathematical Statistics.* 1978.

[36] Huang Chuanhe Roshan Chitrakar. *Anomaly Detection using Support Vector Machine Classification with k-Medoids Clustering.* 2012.

[37] Martin Meckesheimer Sabyasachi Basu. Automatic outlier detection for time series: an application to sensor data. 2007.

[38] Pei Sun Sanjay Chawla. *SLOM: a new measure for local spatial outliers.* 2005.

[39] Graham Williams Simon Hawkins, Hongxing He and Rohan Baxter. *Outlier Detection using Replicator Neural Network.* 2002.

[40] Phillip B. Gibbons Christos Faloutsos Spiros Papadimitriou, Hiroyuki Kitagawa. *LOCI: Fast Outlier Detection Using the Local Correlation Integral".* 2002.

[41] Mark Schwabacher Stephen D. Bay. *Mining Distance Based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule.* 2003.

[42] Lipták T. *On the combination of independent tests.* 1958.

[43] Takayuki Ito Takanobu Otsuka, Yoshitaka Torii. Anomaly detection algorithm for localized abnormal weather using low-cost wireless sensor nodes. In *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*, 2014.

[44] Henry G. Goldberg Ted E. Senator and Alex Memory. *Distinguishing the Unexplainable from the Merely Unusual: Adding Explanations to Outliers to Discover and Detect Significant Complex Rare Events.* 2013.

[45] Michael E. Houle Timothy de Vries, Sanjay Chawla. *Finding Local Anomalies in Very High Dimensional Space.* 2010.

[46] Brian Gallagher Timothy La Fond, Jennifer Neville. *Anomaly Detection in Networks with Changing Trends*. 2014.

[47] Waldo Tobler. *A computer movie simulating urban growth in the Detroit region*. 1970.

[48] Arindam Banerjee Varun Chandola and Vipin Kumar. *Anomaly Detection: A survey*. 2009.

[49] Ismo Kärkkäinen Ville Hautamäki and Pasi Fränti. *Outlier Detection Using k-Nearest Neighbour Graph*. 2004.

[50] Gregory Cooper Michael Wagner Weng-Keen Wong, Andrew Moore. *Bayesian Network Anomaly Pattern Detection for Disease Outbreaks*. 2003.

[51] Vandana P. Janeja Yanan Sun. *STOUT : Spatio-Temporal Outlier detection Using Tensors*. 2014.

[52] Jian Xu Tao Li Yexi Jiang, Chunqiu Zeng. *Real Time Contextual Collective Anomaly Detection over Multiple Data Streams*. 2014.

[53] Dechang Chen Yufeng Kou, Chang-Tien Lu. *Spatial Weighted Outlier Detection*. 2006.

[54] Shengchun Deng Zengyou He, Xiaofei Xu. *Discovering Cluster Based Local Outliers*. 2003.

[55] Shengchun Deng Zengyou He, Xiaofei Xu. *An optimization model for outlier detection in categorical data*. 2005.

[56] Honghai Liu Zhaojie Ju. *Fuzzy Gaussian Mixture Models*. 2011.