FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Anomaly Detection on Data Streams from Vehicular Networks

Eduardo Dantas Barreto Rodrigues

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Internal Supervisor: Prof. Dr. Ana Aguiar

External Supervisor: Dr. Daniel Moura

July 25, 2018

Resumo

Redes veiculares são caracterizadas pela alta mobilidade dos nós que apenas estão ativos quando estão em movimento, o que causa que a rede seja imprevisível e em constante mudança. Num cenário tao dinámico, torna-se crucial conseguir detetar anomalias na rede. A Veniam opera uma rede veicular instalada em autocarros e veículos municipais em cidades como o Porto, Nova York e Singapura. A rede garante uma conexão fiável através de redes heterogéneas como LTE, Wi-Fi e DSRC, que conectam os veículos à Internet e a outros dispositivos espalhados pela cidade. Ao longo do tempo, os nós enviam dados para a Cloud através de tecnologias em tempo real ou tecnologias tolerantes a atraso, aumentando a dinamica da rede.

O objetivo desta dissertação é propor e implementar um método para detetar anomalias numa rede veicular real, através da análise de fluxos de dados que chegam dos veículos até à Cloud. Inicialmente, os fluxos de dados foram explorados com o objetivo de caracterizar os dados disponíveis, de forma a selecionar os casos de uso pretendidos. Os *datasets* escolhidos foram submetidos a vários tipos de algoritmos de deteção de anomalias tais como previsão de séries temporais, abordagem estatística ou deteção de outliers baseado em densidade de amostras e os seus *trade-offs* foram avaliados. A solução proposta juntou os modelos que melhor se adequaram às características dos dados e foi composta por duas fases: uma fase de triagem seguida de uma fase de classificação baseada no método dos vizinhos mais próximos (Nearest Neighbours).

A performance do método foi avaliada pelas suas curvas ROC, i.e. o *trade-off* entre a taxa de verdadeiros positivos e a taxa de falsos positivos, quando foi submetido a *datasets* com anomalias artificiais provenientes de diferentes fontes de dados, tanto sob tecnologias em tempo real como sob tecnologias tolerantes a atraso. Este algoritmo foi capaz de atingir um TPR de 100% e um FPR de 0.4%.

Abstract

Vehicular networks are characterized by high mobility nodes that are only active when the vehicle is moving what causes the network to be unpredictable and in constant change. In such a dynamic scenario, detecting anomalies in the network is a challenging but crucial task. Veniam operates a vehicular network deployed in buses and municipally vehicles in different cities such as Porto, New York and Singapore. The network ensures reliable connectivity over heterogeneous networks such as LTE, Wi-Fi and DSRC, that connects the vehicles to the Internet and to other devices spread throughout the city. Over time, nodes send data to the cloud either by real time technologies or delay tolerant ones, increasing the network's dynamic.

The aim of this dissertation is to propose and implement a method for detecting anomalies in a real-world vehicular network by means of an analysis of the data streams that come from the vehicles to the cloud. First, the network's data streams were explored in order to characterize the available data so that target use cases could be selected. The chosen datasets were submitted to different anomaly detection algorithms such as time series forecasting, statistical approaches and density-based outlier detection and their trade-offs were evaluated. The proposed solution gathered the models that best fitted the data characteristics and it comprised two stages: a lightweight screening step followed by a Nearest Neighbor classification.

The performance of the method was evaluated by its ROC curves, i.e. the trade-off between true positives and false positives, when it was submitted to datasets with artificial anomalies from different data sources, received either by real-time or delay tolerant technologies. The algorithm was able to reach a 100% TPR with a 0.4% FPR.

Acknowledgments

I would like to express my sincere appreciation to my advisers Prof. Ana Aguiar and Dr. Daniel Moura who offered valuable support during this dissertation, giving me the best conditions to learn and improve. Without their guidance and persistent help, this dissertation would not have been possible.

I'm really thankful to Veniam for giving me the chance to pursue this project, especially to the Data Science team Nuno Coutinho and Patrícia Oliveira for all the advice and assistance.

I would also like to thank my family who has been a great support during my studies and the development of this dissertation, especially to my parents.

I am also grateful to my college friends for their support not only during this dissertation but for all these years working together.

Lastly, i would like to thank my friends from Barcelos for all the good mood during the moments of extra work.

Eduardo Rodrigues

"Engineering is done with numbers. Analysis without numbers is only an opinion."

Akin's Laws of Spacecraft Design

Contents

List of Figures

List of Tables

Abbreviations and Symbols

ANN	Artificial Neural Networks
ARIMA	Auto Regression Integrated Moving Average
ARMA	Auto Regression Moving Average
AR	Auto Regression
BN	Bayesian Network
CPT	Condition Probability Table
DAG	Direct Acyclic Graph
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DSRC	Dedicated Short-Range Communications
EM	Expectation Maximization
FP	False Positive
FPR	False Positive Rate
GMM	Gaussian Mixture Model
HDFS	Hadoop Distributed File System
IoT	Internet of Things
kNN	k Nearest Neighbor
MA	Moving Average
MSE	Mean Square Error
MLP	Multi Layer Perceptron
MMH	Maximum Marginal Hyperplane
NN	Neural Network
OBU	On-Board Unit
RNN	Random Neural Network
ROC	Receiver Operating Characteristic
RSU	Road Side Unit
SMO	Sequential Minimal Optimization
SVM	Support Vector Machines
TN	True Negative
TP	True Positive
TPR	True Positve Rate
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VANET	Vehicular Ad-hoc Network
YARN	Yet Another Resource Negotiator

Chapter 1

Introduction

1.1 Context

With the exponential growth of Internet of Things and data sources, the amount of connected devices is constantly growing. In [1], Cisco predicted that in 2020 there will be 50 Billion connected devices and in [2] it is said that in the same year, the digital universe will be higher than 40000 exabytes. Therefore, the amount of available data has been increasing to unimaginable proportions that only distributed computing infrastructures will be able to process. Gathering all this continuously changing data only makes sense if it is possible to retrieve essential information from it. Driven by this problem, data analysis and data processing are fields that have been proliferating in the past few years. Throughout several industries, sensors have been deployed in order to monitor different parameters in an automated way, otherwise impossible to control. All this generated data is fed into different algorithms trying to detect patterns, deviations or faults in order to improve the decision making.

Detecting anomalies is a big part of data analysis. In different scenarios such as predictive maintenance, fraud detection or security attacks, the systems behave differently from normal, generating anomalies that can be detected to identify these problems. In the last decades, various subjects like machine learning and outlier detection, among other statistical approaches, have been explored in order to build autonomous systems that can analyze the data and build models, either online or offline, that detect abnormal behaviour. This data processing is still being improved due to the proliferation of distributed computing technologies, such as Hadoop [3] or Spark [4]. These softwares build clusters with multiple nodes and distribute the storage and processing load across all of them, decreasing the processing time and increasing the storage availability.

Anomaly detection is widely applied to time-series analysis. In some use cases it is mandatory for these detections to happen in real-time. This way, alerts can be generated that might contribute to a faster repair of the system. Anomaly detection can also be used to detect patterns that are common before an anomaly, this way the system can also be used to predict the anomaly before it even happens. This need for faster processing led to the growth of real-time distributed processing tools such as Spark Streaming [5] or Apache Storm [6], where data is analyzed by streams instead of batches, and the model is updated at each iteration.

One of the main applications of data-streams analysis is anomaly detection in networks. Although mining data from different kinds of networks (such as computer or energy networks) has been extensively studied before, there are no real-time anomaly detection systems for vehicular networks described in the literature. Hence, this dissertation aims to create a model able to detect malfunctions or deviations in a real world vehicular network. The resulting model will then be deployed in a distributed cluster in the cloud, ingesting data and making predictions in real-time.

1.2 Motivation

Veniam operates a network where vehicles are seen as nodes and each one of them gathers data from different sensors. The network is composed of buses and municipality vehicles in cities spread across several continents like Porto, New York or Singapore. The communication between them and to the Internet is made using different interfaces that use different communication protocols. While travelling through the city, every vehicle collects data from different sources, that can be sent to the cloud either via real-time technologies or delay tolerant ones, *i.e.* they are sent when there are favorable conditions. In such a dynamic environment, where nodes' position is constantly changing and sensors are only active when the vehicle is moving, detecting anomalies in real-time is a challenging task. However, identifying malfunctions in the network or in some specific vehicle is of utmost importance to ensure a stable operation of the system. Since the monitoring cannot be made manually due to the high number of parameters caused by the high mobility, and the usage of static thresholds might be unable to adapt to changes, building an automatic and accurate anomaly detection becomes crucial.

1.3 Goals

This dissertation's goal is to build, deploy and evaluate a method to detect anomalies in vehicular networks based on data streams that come from the vehicles to the cloud either by real-time or by delay tolerant technologies. It is expected to be used as the first monitoring tool of the entire network, thus its goal is to detect any behavior that is not normal and then other tools that are more specific can be triggered to find the cause of that problem. The abovementioned method shall be able to classify the streams as well as adapt the model continuously without the need of frequent re-training. It is expected that the model adapts autonomously to seasonality changes or the addition of new fleets, where the network starts showing a new pattern. It is also expected that the system is able to deal with different time zones and cultures that affect the data pattern which are characteristics of a planetary network such as Veniam's. Lastly, the solution shall be able to detect anomalies in different data sources that are sent using different interfaces so that the algorithm is able to monitor the entire network. This dissertation is expected to achieve two goals:

- Deployment of the method and the evaluation of its accuracy.
- Deploy the algorithm over the distributed cloud infrastructure of Veniam in order be fast and able to scale

On the whole, this dissertation might contribute to the expansion of anomaly detection in network fields due to the unique environment in which Veniam operates, presenting problems that have not yet been dealt with.

1.4 Document Structure

This report is divided in 8 chapters. This Chapter presents the context and goals of this dissertation. Chapter 2 makes a theoretical introduction to Vehicular networks and to algorithms frequently used in anomaly detection. In Chapter 3 a review about anomaly detection methodologies is made. Chapter 4 presents the available data and the characteristics of the chosen dataset. Chapter 5 presents the trade-offs of several anomalies detection techniques and the chosen solution. The results achieved by the proposed solution are described in chapter 6. The implementation of the solution over the distributed cluster is presented in chapter 7. Lastly, Chapter 8 presents the a summary of the dissertation followed by the contributions made and future work.

Chapter 2

Theoretical Background

2.1 Vehicular networks

Vehicular ad hoc networks (VANETs) are a class of wireless networks that are formed between vehicles equipped with wireless interfaces that can have similar or different radio interfaces technologies [7]. Driven by this emerging network, a dedicated short-range communications (DSRC) system has been created for such type of communication. VANETs are considered as a real life ad hoc network that enables communication between nearby vehicles as well as communication among vehicles and roadside units (RSU). Thus, vehicles and RSUs are seen as the nodes of the network. The RSUs can communicate with each other originating the access network that can also be connected to the Internet. There can be three types of VANETs [7]: (a) a Vehicle-to-vehicle (V2V) wireless network that has no infrastructure support, (b) a Vehicle-to-Infrastructure (V2I) network where vehicles communicate with fixed nodes (RSUs) that intermediate the communication, and (c) a hybrid Vehicle-to-Road (V2R) network that combines V2V with V2I, where vehicles can communicate with each other as well as communicate with the infrastructure either by a single hop or a mulltihop, according to the vehicle position [7]. Figure 2.1 depicts the three possible VANET architectures.

Typically, an in-vehicle network is composed by two types of units: (a) an on-board unit (OBU) which is responsible for sensing and for communication using various technologies such as DSRC, LTE, Wi-Fi, among others, and (b) an application unit (AU) which is responsible for executing applications. Vehicular networks have characteristics which distinguish them from other networks. Unlike sensor networks, VANETs do not have concerns about the devices' power since the vehicle is able to produce continuous power to computing and communication devices. Furthermore, they have higher computational capabilities and, sometimes, a predictable mobility since they have to follow existing roads. On the other hand, they also face some challenging tasks [8]. The network is extremely dynamic since the vehicles can travel on several roads and at different speeds, which cause topology changes and frequent connections and disconnections between vehicles.

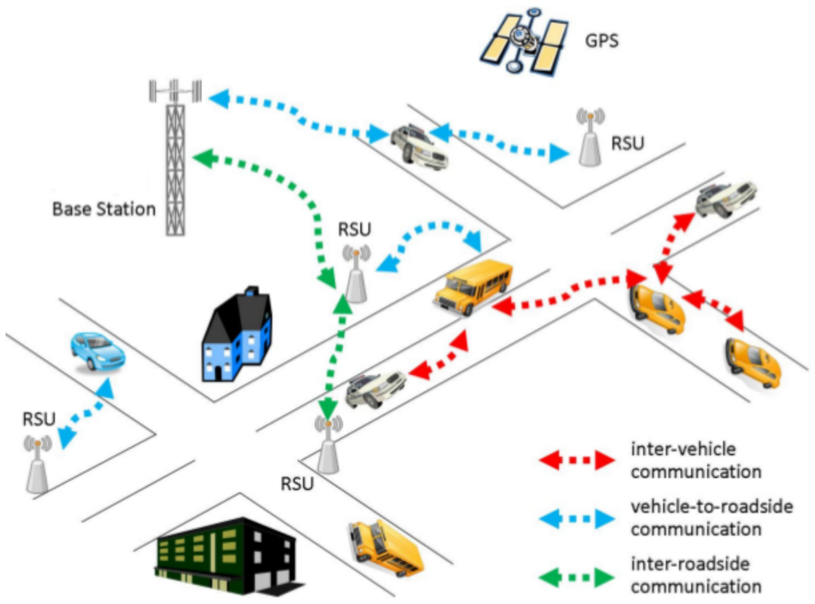

GPS
RSU
Base Station
RSU
RSU
RSU
inter-vehicle communication
vehicle-to-roadside communication
inter-roadside communication

Figure 2.1: Types of VANETs' infrastructures [9].

2.2 Anomaly detection Algorithms

Anomaly detection has been explored by several fields such as machine learning, data mining or statistical approaches. This section makes a theoretical approach to some algorithms that have already been used to detect anomalies in different fields.

2.2.1 Classification algorithms

Classification uses a labeled dataset in order to build a model named classifier, thus being considered a supervised method. It classifies each instance in one of many subclasses, assuming that the dataset is composed by a representative set of each one of those subclasses. Typically, classification is divided in two steps: training and testing. The training phase uses the labeled data to build the classifier model. The testing phase classifies each instance accordingly to the model built. However, it is hard to build a labeled dataset with anomalies since they are not too frequent and new anomaly patterns can always occur. Nonetheless, this section presents some of the most popular classification techniques used in anomaly detection.

2.2.1.1 Neural Networks

A neural network is a set of N neurons that are divided into three or more layers: an input layer, an output layer and one or more hidden layers. The neurons can be connected to each other and, to each connection a weight is attributed. During the training phase, the network *learns* by adjusting those weights in order to be able to correctly predict the output given the input data. Typically, this training phase involves long periods of time due to the large amounts of data that are necessary to build the model, which makes the use of a neural network unfeasible for some

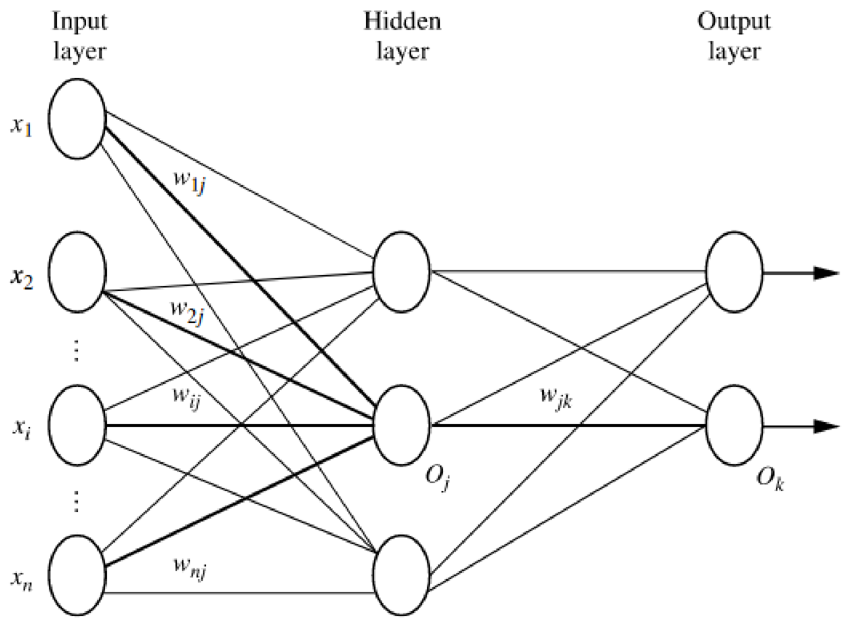


Figure 2.2: Multilayer feedforward network with 2 layers [10]

applications. Furthermore, user-defined parameters are needed in order to build the network, such as the number of neurons per layer or the number of layers. On the other hand, they are very tolerant to noisy data and can be used when there is little knowledge of the relationships between attributes and classes [10]. One of the simplest neural networks is the feedforward neural network that can be seen in figure 2.2. It is called a feedfoward network because none of the connections cycles back to a neuron of a previous layer. It is composed by two layers of output units (input layer does not count), therefore called a two-layer neural network. The input data is given to the input layer that forwards it to the hidden layer with a certain weight. Then, each neuron in the hidden layer calculates the weighted sum of the values received from the input layer and applies a non-linear function to it, such as the **sigmoid** that is given by:

$$O_j = \frac{1}{1+e^{-I_j}} \tag{2.1}$$

where O_j is the output of the jth neuron given the input I_j. Then, that value is forwarded to the output layer that performs the same process and produces the network's prediction. To update the network, the **Backpropagation** algorithm is applied. The predicted value is compared to the real one and the error is calculated. Then, the error is propagated backwards by updating the weights in order to minimize the mean-squared error between the network's prediction and the actual value [10].

2.2.1.2 Bayesian Networks

A Bayesian Network is used when dependencies can exist between variables. It is defined by two components: a Directed Acyclic Graph (DAG) and a Conditional Probability Table (CPT)

[10]. Each node of the DAG represents a random variable. These variables can be discrete or continuous and represent data attributes or *hidden* variables, *i.e* variables that are not specified in the data. Each arc of the DAG represents a probabilistic dependency. If an arc is drawn from a node A to a node B, then A is parent of B and B is descendent of A. Thus, representing a causal relationship between them. A variable C that does not descend from A nor B is considered to be conditionally independent from them. A Bayesian Network is also composed by a Conditional Probability Table (CPT) for each variable. For instance, the CPT for a variable X, specifies the conditional distribution $P(X|Parents(X))$, where $Parents(x)$ denotes the parents of X.

One or more nodes within the network can be selected as an output node that can either return a class or a probability distribution that gives the probability of each class. The training of a Bayesian Network, *i.e.* learning the value for the CPT tables, can be achieved by using the **gradient descent** algorithm [10]. This strategy is used to search for the values that best model the data. At each iteration the values of the tables are updated, eventually converging to a local optimal solution.

2.2.1.3 Support Vector Machines

Support Vector Machines can be used either for linear or non-linear data. It uses a non-linear mapping that transforms the input training data into a higher dimension. In this new space dimension, the algorithm searches for an ideal hyperplane that better separates one class from another. According to [10], with an ideal mapping to a sufficiently high dimensional space, data instances from two classes are always separable by a hyperplane. SVM's purpose is to find the hyperplane that best separates training data and that is capable of separating future instances. As such, it looks for the hyperplane with the largest margin, *i.e* the one that guarantees a bigger separation between the two classes, known as maximum marginal hyperplane (MMH). The SVM algorithm finds the hyperplane using support vectors. These are data instances from the training data that are equally close to the MMH. Figure 2.3 shows a a graphical representation of a 2 dimensional dataset that is linearly separable by a MMH.

When in presence of non-linear data, it is not possible to separate the data with a straight line as seen in Figure 2.4. In such cases, it is necessary to map the data into a higher dimensional space, using a non-linear mapping, until is possible to linearly separate the data with a hyperplane.

2.2.2 Distance based algorithms

Distance based algorithms use distance or similarity between data instances to find outliers. However, several methods to calculate similarities have been proposed. Euclidean distance is the simplest measurement and is given by the following formula:

$$d(p,q) = \sqrt{\sum_{i=1}^{d}(p_i - q_i)^2} \tag{2.2}$$

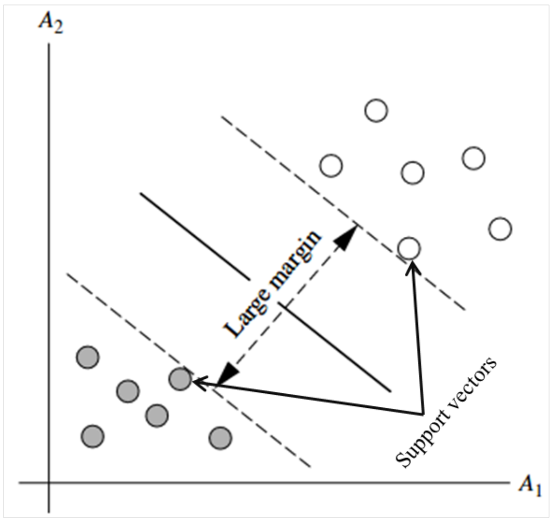


Figure 2.3: Data linearly separable by an MMH [10].

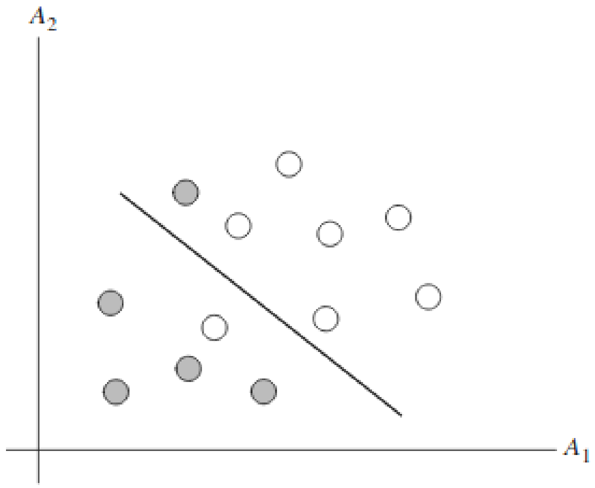


Figure 2.4: Data linearly inseparable by an MMH [10].

where p and q are the points between which the similarity is being calculated and d is the number of dimensions.

The Mahalanobis distance is used when the dataset has different mutual correlations and is given by :

$$\sqrt{(p-q)^T S^- 1(p-q)} \tag{2.3}$$

where S is the covariance matrix that measures the correlations between dimensions in the data set. The Minkowski distance of order *l* is given by:

$$(\sum_{i=1}^{d} |p_i - q_i|^l)^{\frac{1}{l}} \tag{2.4}$$

when l = 1 we get:

$$\sum_{i=1}^{d} |p_i - q_i| \tag{2.5}$$

which is known as the Manhattan distance. The euclidean metric is used when all the dimensions have the same units while the Mahalanobis can be used with different units among the dimensaions.

2.2.2.1 K-Nearest Neighbor

Nearest Neighbor algorithms are based in the assumption that the close-by data points are likely to have the same label [11]. The *k*-Nearest Neighbor measures the distance or similarity between a given data instance and its *k* nearest neighbor. In a classification approach, an instance can be classified with the same label as its k-Nearest Neighbor. It can also be classified with the average label of its k nearest neighbors or as the majority of the labels within its k nearest neighbors.

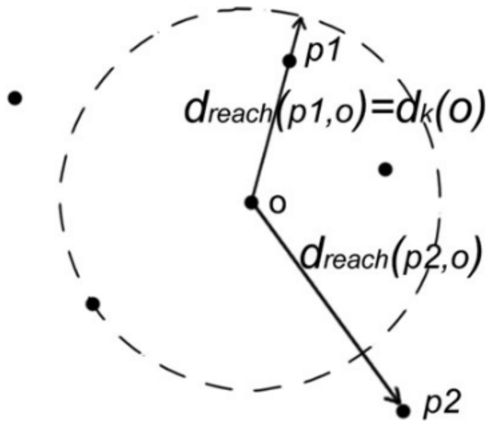


Figure 2.5: Reachability distance for k=3 [12].

The outlier detection approach consists in measuring the distance to its k-nearest neighbor and compare that value with a certain threshold. That value can be manually defined or obtained by the training dataset. The problem with kNN is that it is lazy-learning. At each instance, it must go by the entire dataset to find the k-nearest neighbors which can be computationally expensive.

2.2.2.2 Local Outlier Factor

The Local Outlier Factor (LOF) is a density based approach to find outliers in a dataset. A density based methodology compares the local density of the neighborhood of a point with the density of the vicinity of its neighbors [12]. In particular, LOF compares the density of the neighborhood a point p with its k nearest neighbors. Let $N_k(p)$ be the set of points whose distance to p is less than the distance between p and its k nearest neighbor $d_k(p)$. The LOF algorithm works by finding the local reachability density of a point p that is given by:

$$l_k(p) = \frac{||N_k(p)||}{\sum_{q \in N(p)} d_{reach}(p,q)} \tag{2.6}$$

where $d_{reach}(p,q)$ is the reachability distance of a point q from p and is given by $max\{d_k(q), d(p,q)\}$. Figure 5.8 illustrates how to calculate this value.

Finally, the LOF value is calculated by comparing the local reachability density of p with the local reachability of all points in $N_k(p)$:

$$LOF_k(p) = \frac{\sum_{q \in N_k(p)} \frac{l_k(q)}{l_k(p)}}{||N_k(p)||} \tag{2.7}$$

2.2.3 Cluster Algorithms

Clustering is the task of grouping similar instances inside the same group and dissimilar instances in different groups [11]. Figure 2.6 shows a a graphical representation of clustering with four clusters.

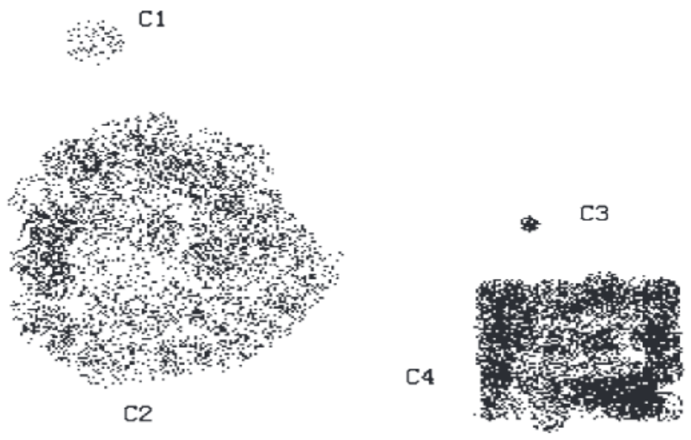


Figure 2.6: Result of a clustering technique with four clusters [13]

Cluster techniques may have a few variations. Some algorithms assign each instance to a certain cluster while others may allow for some data instances not to be clustered. A third approach may allow for clusters to overlap, having instances that can belong to more than one cluster. The clusters can be used for classification, where each represents a class, or for outlier detection, where the small or less dense clusters represent outliers.

2.2.3.1 K-Means

K-Means is one of the most used clustering algorithms. This technique assigns each instance to a certain cluster from k possible clusters [10]. Each cluster is represented by its centroid. The algorithm works as following: First, the k number of clusters and their initial centroids must be user defined. Then, each instance in the dataset computes its distance to all the centroids and the instance is assigned to the closer cluster. Then, the centroids of each cluster are updated by calculating the mean of the objects belonging to the cluster. The algorithm halts when there is no centroids change. Although this is one of the most used algorithms for clustering, it has several problems due to the fact that the number of clusters k, and the initial centroids are user defined and may have severe influence in the final result. The *k*-Medoids algorithm is a variation of the *k*-Means algorithm. Instead of calculating the centroids as the mean of all objects inside the cluster, the centroids are the data points in the cluster that lie closer to all the other instances in the same cluster. The accuracy of these algorithms for anomaly detection is too dependent on the value of k. Beyond that, they are too sensitive to outliers since they can all be grouped within the same cluster and be considered normal.

2.2.3.2 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

The DBSCAN was proposed by [14] to discover clusters of random shape from noisy data sets, requiring only one scan of the dataset. For each instance, the number of neighbors within a distance r are calculated. An instance is labeled as a core-point if has at least p points within a distance r.

Then, clusters are formed for each core-point. In each iteration, two clusters are merged if they contain core points within a distance r from each other. A point that is not a core-point, but is inside a cluster, *i.e* in the neighborhood of a core-point, is considered a border-point. A point that is not a core-point and it is not inside any cluster, is considered noise. A demonstration of this operation can be seen in figure 2.7. Data instance A has 6 points in its vicinity so it is considered as a core-point (MinPts=6). B is inside A's neighborhood but it has less than 6 points in its own vicinity, so it is considered a border-point. C is far from any other data instance so it is considered noise.

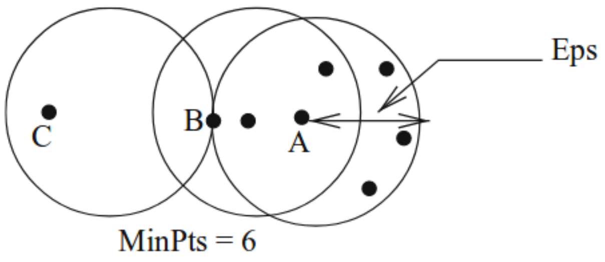


Figure 2.7: A is considered as a core-point because it has MinPts (6) points within a distance Eps. B has only 4 points in its neighborhood but it is inside A's vicinity so it is considered a border-point. C is considered noise. [12]

2.2.4 Statistical Algorithms

Statistical approaches can be divided into two different categories:

- Parametric - assumes that the data was generated by a parametric distribution with parameter ϕ and a probability density function f(x,ϕ), where x is an observation instance. The parameter ϕ is estimated from the data. Alternatively, a statistical hypothesis test can be used [15]. The null hypothesis *H0* to that test is that an instance was generated by the estimated distribution (with parameter ϕ). An instance is considered an outlier if *H0* is rejected by the statistical test [16, 17]. Parametric approaches assume that the distributions of the data, normal or anomalous, is known. Thus it is used when the distribution of the anomalies are known. Regressions models are parametric techniques widely used to find anomalies within time series data. First, a regression model is fitted on the data. Then, the resulting model can be used to forecast future instances, using the forecasting error as a metric to find anomalies. Regression models can also classify instances using the residuals which are the parts of the instances that are not explained by the model.

- Non-Parametric - define the model according to the dataset, instead of defining it a priori. In these techniques, there are no assumptions made about the data and the entire model is built using historical data. These approaches are widely used in anomaly detection since it is difficult to know all the anomalous distributions, thus this approaches are able to deal with anomalies never seen before.

2.2.4.1 Gaussian Mixture model

Gaussian mixture models are a parametric unsupervised technique that can be categorized as a statistical algorithm or as a clustering algorithm because it discovers classes in data that can be represented as a weighted sum of M Gaussian distributions. Let D=(x_1,...,x_n) be a dataset composed by n instances and Λ=(λ_1,...,λ_M) be the parameters for each of the M Gaussian distributions, where $\lambda_j = (\mu_j, \sigma_j)$ denotes the parameters of a distribution centered in μ_j with a standard deviation σ_j. The probability of an instance $x_i \in D$ being generated by the set composed by the M distributions is given by :

$$P(x_i|\Lambda) = \sum_{j=1}^{M} w_j P_j(x_i|\lambda_j) \tag{2.8}$$

where $P_j(x_i|\lambda_j)$ represents the probability of the instance x_i being generated by the distribution with parameters $\lambda_j = (\mu_j, \sigma_j)$ and w_j denotes the weight of that distribution such that $\sum_{i=1}^{M} w_i = 1$ to ensures that all objects are generated by the M distributions.

Consequently, the probability of the entire set D being generated by the M Gaussians is given by:

$$P(D|\Lambda) = \prod_{i=1}^{n} \sum_{j=1}^{M} w_j P_j(x_i|\lambda_j) \tag{2.9}$$

For an univariate Gaussian distribution the probability that an instance $x_i \in D$ belongs to a distribution of parameters $\lambda_j = (\mu_j, \sigma_j)$ is given by :

$$P(x_i|\lambda_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(x_i - \mu_j)^2}{2\sigma^2}} \tag{2.10}$$

Thus, Equation 2.8 becomes:

$$P(x_i|\Lambda) = \sum_{j=1}^{M} w_j \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(x_i - \mu_j)^2}{2\sigma^2}} \tag{2.11}$$

Finally, applying equation 2.9:

$$P(D|\Lambda) = \prod_{i=1}^{n} \sum_{j=1}^{M} w_j \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(x_i - \mu_j)^2}{2\sigma^2}} \tag{2.12}$$

The parameters Λ for which equation (2.12) is maximized can be achieved by using the **Expectation Maximization algorithm (EM)**. Initially, random values are assigned to λ. Then, for each instance two steps are made. The E step and the M step. In the E step, the probability of an instance $x_i \in D$ belong to each distribution is calculated by:

$$P(\lambda_j|x_i,\Lambda) = \frac{P(x_i|\lambda_j)}{\sum_{k=1}^{M} P(x_i|\lambda_k)} \tag{2.13}$$

In the M step, the parameters Λ are adjusted so that the equation 2.12 is maximized:

$$\mu_j = \frac{1}{M}\sum_{i=1}^{n} x_i \frac{P(\lambda_j|x_i,\Lambda)}{\sum_{k=1}^{n} P(\lambda_j|x_k,\Lambda)} \tag{2.14}$$

$$\sigma_j = \sqrt{\frac{\sum_{i=1}^{n} P(\lambda_j|x_i,\Lambda)(x_i-\mu_j)^2}{\sum_{i=1}^{n} P(\lambda_j|x_i,\Lambda)}} \tag{2.15}$$

2.2.4.2 Linear Regression

Linear regression is a parametric algorithm that fits the data into a straight line. A variable y, called the dependent variable, can be modeled as a linear function of another variable x, called independent variable, by the linear function $h(x) = wx + b$, where the coefficient w denotes the slope of the line and the coefficient b specifies the point in which the y axis is intercepted by the line. These parameters are tuned in order to minimize the Mean Square Error (MSE) [11] that is given by:

$$\frac{1}{n}\sum_{i=1}^{n}(h(x_i) - y_i)^2 \tag{2.16}$$

2.2.4.3 Auto Regression Integrated Moving Average (ARIMA)

The Auto Regression Integrated Moving Average is the most general class of models for forecasting a time series. It predicts a variable based on its own history giving a bigger weight to its recent values than older ones. It also allows for non-stationarity and works by a moving average to remove some noise [12]. This model consists in two components, an Auto Regression (AR) component and a Moving Average (MA) component. The AR(p) assumes that a value x_t can be predicted as a linear function of its p past values, $x_{t-1}, x_{t-2}, ..., x_{t-p}$ and is given by:

$$x_t = \sum_{i=1}^{p} \Phi_i x_{t-i} + \varepsilon_t \tag{2.17}$$

where Φ_i is the weight given to the x_{t-i} value and ε_t is the *error* that is assumed to be normally distributed with mean 0 and variance σ^2. Introducing the operator Backshift $B^i_{x_t} = x_{t-i}$ which is used to refer to a variable at the ith previous time point, the equation 2.17 can be rewritten as

$$x_t = \sum_{i=1}^{p} \Phi_i B^i_{x_t} + \varepsilon_t \tag{2.18}$$

or

$$\varepsilon_t = (1 - \sum_{i=1}^{p} \Phi_i B^i) x_t \tag{2.19}$$

The Moving Average (MA) component captures the effect of *shock variables* that are randomly distributed and affect the current level for an interval of time q. Thus, the MA(q) is given by:

$$x_t = \varepsilon_t - \sum_{i=1}^{q} \Theta_i \varepsilon_{t-i} \tag{2.20}$$

using the Backshift operator, the equation 2.20 can be rewritten as:

$$x_t = (1 - \sum_{i=1}^{q} \Theta_i B^i)\varepsilon_t \tag{2.21}$$

when AR(p) and MA(q) are combined, an Auto Regressive Moving average ARMA(p,q) is obtained by:

$$x_t = \Phi_1 x_{t-1} + ... + \Phi_p x_{t-p} + \varepsilon_t - \Theta_1 \varepsilon_{t-1} - .... - \Theta_q \varepsilon_{t-q} \tag{2.22}$$

or, using the Backshift operator:

$$(1 - \sum_{i=1}^{p} \Phi_i B^i)x_t = (1 - \sum_{i=1}^{q} \Theta_i B^i)\varepsilon_t \tag{2.23}$$

However, ARMA(p,q) assumes that the data is non-stationary which is not always true. Under this conditions, the Auto Regressive Integrated Moving Average (ARIMA) was developed. It introduced a new parameter *d* that allows to capture non-stationarity in the time sequences:

$$(1 - \sum_{i=1}^{p} \Phi_i B^i)(1 - B)^d x_t = (1 - \sum_{i=1}^{q} \Theta_i B^i)\varepsilon_t \tag{2.24}$$

2.2.4.4 Histograms

Histograms are the most simple non-parametric statistical approach. It represents a graphical method for summarizing the distribution of a given attribute, X. The range of values for X is partitioned into disjoint consecutive sub-ranges, referred as bins. The range of a bin is known as the width and is typically the same for all bins. For each sub-range, a bar is drawn with a height that represents the total count of items observed within the sub-range. This approach can be used for anomaly detection by counting the amount of times each value occurs. The ones with lowest values, or values below some threshold, can be classified as anomalies.

2.3 Evaluation Methods

The evaluation of an algorithms' performance is crucial when deploying an algorithm. It indicates how successful the predictions made by the trained model were. Several methodologies can be used but most of them use four different values:

- True Positive (TP) - refers to the number of instances that belong to the class and were correctly labeled.

- True Negative (TN) - refers to the number of instances that do not belong to the class and were correctly labeled.
- False Positive (FP) - refers to the number of instances that do not belong to the class and were incorrectly labeled.
- False Negative (FN) - refers to the number of instances that belong to the class and were incorrectly labeled.

Confusion Matrix

The confusion matrix is a table where an entry in row i and column j shows the number of instances from class i that were labeled as the class j. The figure 2.8 shows a confusion matrix for a multiclass classification.

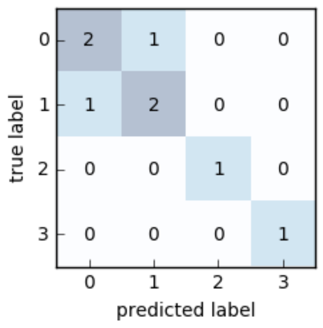


Figure 2.8: Confusion matrix example.

Accuracy

The accuracy is the percentage of test instances that were correctly classified.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.25}$$

Error Rate

The error rate is the percentage of test instances that were incorrectly classified.

$$ErrorRate = \frac{FP+FN}{TP+TN+FP+FN} \tag{2.26}$$

Recall

The recall measures the percentage of instances from a class that are correctly classified as such.

$$Recall = \frac{TP}{TP+FN} \tag{2.27}$$

Precision

Precision measures the percentage of instances labeled as positive that are actually positive.

$$Precision = \frac{TP}{TP+FP} \tag{2.28}$$

F1 Score

The F1 Score is a combination of the recall and the precision.

$$F1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{2.29}$$

k-fold Cross Validation

In this method, the dataset D is divided into k subsets of equal size $D_1, D_2, ..., D_k$. It is an iterative process of k iterations. At each iteration i, the subset D_i is used for testing while the other $k-1$ subsets are used to train the model. The accuracy is calculated by dividing the number of correctly labeled instances from the k iterations by the number of samples in the original dataset D [10].

Receiver operating characteristic

The Receiver operating characteristic (ROC) curves shows the trade-off between the true positive rate (TPR) and the false positive rate (FPR) of a given model. It is frequently used to compare two models. The accuracy of the model is given by the area under the ROC. Figure 2.9 shows a comparison between two models, M1 and M2, using the ROC technique. The model M1 has a higher area under the curve, thus being more accurate.

2.4 Distributed Computing

Distributed computing systems are networks with a large number of nodes that operate cooperatively in order to achieve a desired objective [18]. Dividing a problem among the available nodes allows for high computational capabilities. Driven by the explosion of the amount of available data, distributed computing systems have been evolving in order to be able to effectively process all that data. This process can be made by two methods:

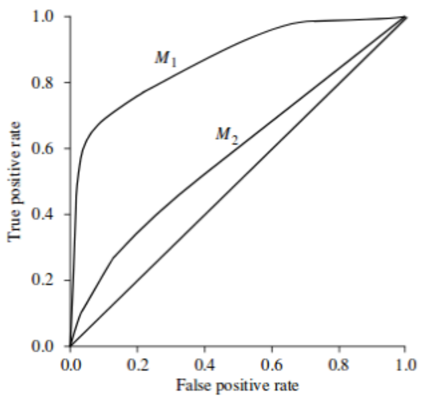


Figure 2.9: Comparison between two models, M1 and M2, using the ROC method. M2 has a higher accuracy because it has a larger area under ROC [10].

- Batch process - The data is inserted into the system at once. Algorithms running in batch mode have access to the entire dataset, thus can perform complex and longer operations.
- Stream process - The data is inserted into the system by data streams of small size. Algorithms running in stream mode must perform simple operations with each data stream.

This section presents some of the most popular data processing technologies both for batch mode and streaming mode.

2.4.1 Hadoop

Hadoop is an open-source software system that allows the distributed processing of large data sets across clusters [3]. It offers three types of tools:

- The Hadoop Distributed File System (HDFS), which is a fault-tolerant distributed file system that partitions and replicates the data to be stored on the nodes of the cluster.
- The Hadoop MapReduce, which allows for processing large datasets on parallel computer systems.
- The Hadoop YARN (Yet Another Resource Negotiator), which is responsible for job scheduling and cluster resource management.

2.4.2 Apache Spark

Spark is an open source engine for large-scale data processing. It can be ran along with Hadoop, EC2 or Mesos and can access several distributed storage system including HDFS, Cassandra, HBase, and S3 [4]. Spark's main advantage is that it does not require to store the data to disk after each iteration like Hadoop MapReduce. Instead, Spark performs in-memory processing of data, only using the disk in case of need. It has been proven that spark can be 100 time faster than Hadoop [19].

Spark has a built-in module called Spark-streaming that allows for scalable, high-throughput, fault-tolerant stream processing of live data streams that can come from different sources such as Kafka, Flume, ZeroMQ, Kinesis, among other [4].

A user program built on Spark is called an Application. An application comprises two types of operations: transformations and actions. Transformations are lazy which means that when some operation is called it is not executed immediately. The actions triggers the execution of all previous transformations by spawning a Job composed by a set of tasks. Each job gets divided into smaller sets of tasks called stages that depend on each other. The task is the unit of work that will be sent to one executor. Spark Architecture is divided into 5 agents:

- Cluster - The collection of nodes on which Spark is running
- Master - The cluster manager that is responsible for negotiating resource requests made by the driver to the resource manager.
- Worker - The workers receive instructions from the Master and launches Executors to execute tasks on behalf of the Driver.
- Executor - Is a JVM container launched by each Worker that executes Spark tasks.
- Driver - Responsible for distributing the tasks by each Worker's Executor. It also receives computed results from each Executor's tasks.

Spark supports three cluster deployments modes, each with its characteristics with respect to where Spark's components run within a Spark Cluster. All three of these frameworks have two components. A central master service that decides where and when each application gets to run and a slave service running on every node that actually starts the executor processes.

The **standalone mode** is the simplest approach and it is included with Spark. The machine where the Spark application process is running is the Driver and the Master is allocated arbitrarily. Each worker launches its own JVM to run the executor.

In **YARN mode** each application instance has an Application Master process that is responsible for requesting resources from the ResourceManager and telling NodeManagers to start containers on its behalf. Spark supports two modes for running on YARN, "yarn-cluster" mode and "yarn-client" mode. In yarn-cluster mode, the driver runs in the Application Master. In yarn-client mode, the Application Master is merely present to request executor containers from YARN.

In **Mesos mode** the Mesos master is the cluster manager. For each job, Mesos determines what machines handle what tasks. The advantages of deploying Spark with Mesos include dynamic partitioning between Spark and other frameworks. Given that, multiple frameworks can coexist on the same cluster without resorting to a static partitioning of resources. Mesos also supports two types of cluster modes: client-mode and cluster-mode. In client mode the driver runs on a client machine, not a part of Mesos cluster. In cluster mode the driver runs within Mesos's master.

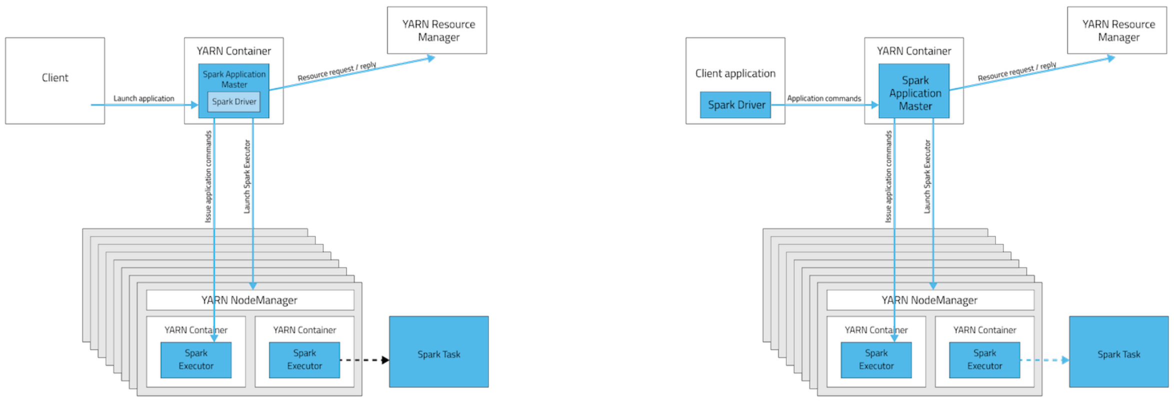


Figure 2.10: YARN cluster mode (left) and YARN client mode (right).

2.4.3 Apache Storm

Apache Storm is an open source engine that provides a scalable and fault-tolerant distributed real-time computation. It is composed by three abstractions: spouts, bolts and topologies. Spouts are the source of data streams and can receive data from different brokers such as Kestrel, RabbitMQ, or Kafka. The bolts are the processing units with all the logic of a computation. The topologies are networks composed by spouts and bolts. Storm has been proven to be extremely fast with up to a million tuples per second per node [6]. The main difference between Storm and Spark is that Storm uses task-parallel computations while Spark does data-parallel computations. The first consists in simultaneous executions of different functions while the seconds consists in diving the dataset among the nodes and execute the same function in all of them.

2.5 Summary

This chapter presented a theoretical introduction to the subjects addressed by this dissertation. First a simple approach to Vehicular Networks was made. Then, several algorithms widely applied for anomaly detection were described. These algorithms were separated into four categories: classification, distance-based, clustering and statistical. Furthermore, a review was made on methods of evaluation of the above mentioned algorithms. Finally, distributed computing systems were characterized.

Chapter 3

Anomaly Detection in networks

Anomalies are characterized as changes in the networks' behaviour that occur before or during an anomalous event [20]. Typically, they are problems that are not known or never happened before. This means that they are not identified by their characteristics, but by modeling what it is considered to be normal. Anomalies can be caused by a variety of reasons such as device malfunctions, network failure, network overload or malicious attacks. These events lead to deviations in the networks' normal pattern of operation, which are then classified as anomalies [21]. The problem arises when it is necessary to define what is considered normal. This definition depends on several network metrics such as traffic volume, data types or applications deployed to the network [20]. Thus, the first step in anomaly detection is to find a methodology to define what is the normal behaviour of the network. To do this, it is necessary to ally the human insight with mathematical models in order to identify if a certain situation is, or is not, an ideal representation of a normal behaviour. Then, a statistical method is applied to the data in order to detect the anomalies. Several methods based in machine learning or data mining are available in the literature. Most of them are based on the same methodology: create a model using a dataset that is considered normal, or anomaly free, and classify a data instance as an anomaly if it does not fit the trained model. Typically an anomaly detection approach is dived in four steps [16]:

- Data collection
- Selection of relevant features
- Analysis of preproccessed data
- Validation

This chapter presents a literature review about some methodologies used in order to build an anomaly detector. Since anomaly detection in vehicular networks has not yet been explored in depth, this chapter only presents methods to detect anomalies in environments such as wireless networks, sensor networks or computer networks that have some characteristics of interest.

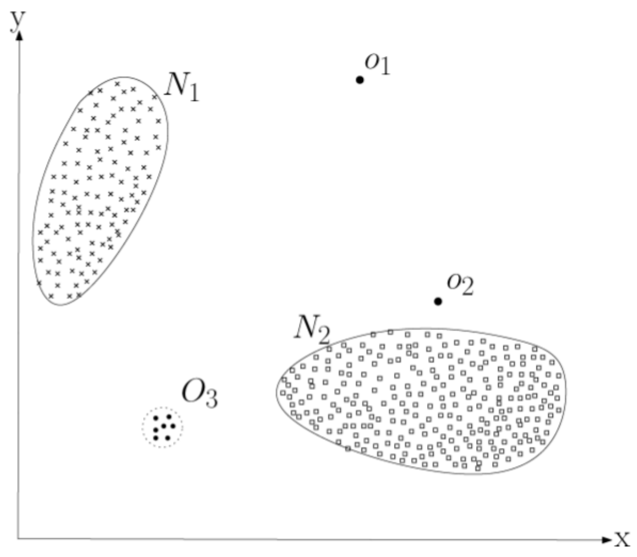


Figure 3.1: Two dimensional dataset with anomalies.N1 and N2 are considered normal regions. O1 is considered a point anomaly. O2 is considered a contextual anomaly regarding N2. O3 is considered a collective anomaly regarding N1 and N2 [17].

3.1 Type of Anomalies

Network Anomalies can be caused by a variety of different reasons. However, they are normally divided into two categories, network performance related and security-related [16]:

- Performance related anomalies can also be divided into three causes:
  - Bad software or hardware design
  - Bad design or configuration of systems
  - Lack of management or monitoring
- Security related anomalies are also dived into three groups [17]:
  - Point Anomaly: instances that are exceptional when compared to the remaining data.
  - Contextual Anomaly: instances that are exceptional when compared to a certain context.
  - Collective Anomaly: group of instances that is considered different from the normal behavior.

Figure 3.1 shows a simple example of anomaly detection in a 2 dimensional dataset. N1 and N2 are considered normal regions. O1 is considered a point anomaly since it is a single point far away from any normal region. O2 is considered a contextual anomaly regarding N2 because it lies outside N2's boundaries. O3 is a group of instances but with a total number of instances much smaller than N1 or N2 so it is considered a collective anomaly.

3.2 Data labels

3.2.1 Supervised learning

A supervised algorithm requires a training dataset where data is labeled as normal or as known anomalies. This labeling process is usually made by humans which is expensive due to the amount of time wasted doing the labeling [16]. Another problem that arises when building datasets to be used in supervised learning, is that anomalous patterns are significantly less frequent than the normal ones. Thus, they are difficult to represent in an accurate and representative manner [17]. To counter this problems, [22, 23, 24] developed techniques to generate artificial data to be mixed with the original data such as discard or introduce data instances or permute the data position within the dataset. New data instances are put to the test against the built model and are classified in one of the many classes.

3.2.2 Semi supervised learning

Semi supervised techniques in anomaly detection assume that the dataset is composed only by normal data. As they do not require to generate representative data for the anomalies, they are more often used than supervised methods. Typically, these methods build models for the normal behaviour and classify as anomalies the instances that do not fit the model, *i.e.* exceed certain thresholds or that do not belong to any cluster [16].

3.2.3 Unsupervised learning

Unsupervised learning techniques do not require the training dataset to be labeled, thus being widely used for anomaly detection. This methods assume that normal data instances are far more frequent than the anomalous ones and that anomaly data can be statistically differentiated from normal data [17]. Unsupervised techniques group data into categories or sub groups based on their properties. The groups are then assigned with a label (normal/anomaly) based on a certain criteria, depending on the algorithm used [16].

3.3 Classification-based anomaly detection

Anomaly detection classification can be divided into two categories:

- One class anomaly detection – The model assumes that the dataset is composed by only one class that is normal. The model must build a boundary around the normal class and classify every instance that lies outside the boundaries as an anomaly.
- Multiclass anomaly detection – assumes that the dataset is composed by several normal classes. Thus, the model must be able to distinguish between the normal classes. If an instance is not assign to any of those classes it is considered an anomaly.

3.3.1 Neural Networks-Based

Neural Networks are typically composed by N neurons which are divided into three subsets: the input neurons (belonging to the input layer) , the hidden neurons (belonging to the hidden layers) and the output neurons (belonging to the output layer) . The input neurons receive the input data and send signals to the hidden layer. This layers only interacts either with other hidden layers or with the output layers. The output layers receives signals from the hidden layers and produce the output data. The simplest approach of a neural network is using a three-layers network topology in order to simplify the model and the learning process. A simple approach using an artificial neural network (ANN) was made by [25] where the author built an anomaly detector for a simple IoT network. Each node of the wireless sensor network was responsible for reading the value of the temperature at a given instance. The ANN was composed by three inputs (neurons): the device ID, the temperature value and the delay between transmissions. They also chose a five-layer network with three hidden layers without further explanation. They were able to attain a 99% prediction accuracy and only 1% of false negatives. Although it suggest that NN can be used in anomaly detection, this work was made to a really simple network with a small dataset. [26] proposed a method based on a Random Neural Networks (RNN) to detect anomalies in a cellular network by analyzing the amount of DNS requests issued within a time bin. The RNN is a complex neural network approach that combines artificial neural networks with queuing networks methods. This approach has special interest due to the fact that the data shows a 24-hour seasonality and a weekly pseudo-cycle with marked differences between week days and weekends/holidays. They used a full labeled dataset of a full month of DNS requests containing normal measures and three types of known anomalies. The goal was to detect the full duration of an anomaly instead of just detecting an anomalous event. The RNN model was able to detect 90% of anomalous instances with a False Positive Rate lower than 1%. This approach was also compared to other known algorithms such as SVM, C4.5 Decision tree and Naive Bays and was able to outperform them.

Uber developed a method for detecting anomalies in a vehicular network [27]. They explored a dataset with the total number of daily complete trips, that showed a time pattern between week days and another between weekends. They used a Bayesian Neural Network (BNN) that introduced uncertainty to deep learning models from a Bayesian perspective. They estimated uncertainty from three sources : (a) model uncertainty, (b) inherent noise and (c) model missspecification. The model uncertainty was estimated using a Monte Carlo Droupout. This approach randomly drops out hidden unit with a probability p while computing the neural network. They estimated the model misspecification uncertainty using an encoder-decoder framework. The encoder extracted the representative features from a time series so that the decoder could reconstruct that time series. The quality of encoding each sample will dictate how close the testing set is to the training set. The inherent noise was estimated via the residual sum of squares using another valid independent dataset. They also implemented a prediction network with a Multi Layer Perceptron (MLP) that works together with the encoder-decoder and the Monte Carlo Droupout in order to forecast the next time series value. This method was able to forecast the future value 95% of the times. [28] also proposed a wireless network anomaly detector based in Multi-Layer Perceptrons (MLP) to detect device's malfunctions or abnormal variations in its measurements. The authors implemented a time based MLP (TBMLP), which is a backpropagation-trained MLP whose inputs are time-delayed values, with some modifications. The structure of the neural network was composed by several TBMLPs, each one associated with a single function, connected to a single MLP that is responsible for predicting the next value for each function. This approach was made because it increased the propagation speed over the network due to the less number of connections between nodes. Also, it makes the system more robust to functions changes because the functions are isolated in separate TBMLPs. If the value predicted by the NN and the real value had a difference bigger than a manually defined threshold, the instance was considered an anomaly. Although this approach was able to attain high detection rate, its performance relies in the defined threshold because there is no generic rule to define that value. A lower one increases the detection rate but also increases the false positive rate. On the other hand, a higher threshold decreases the detection rate.

3.3.2 Bayesian Networks-Based

Bayesian Networks can be used for anomaly detection because they can encode dependencies between variables, which handles situations where data is missing. They can also represent casual relationships which might be helpfull to make predictions. Lastly, it efficiently models problems where it is necessary to group previous knowledge with the data [16]. A simple Bayesian network was used by [29] to detect anomalies in vessels. This work unveiled some interesting methodologies because the dataset used was very similar with the one to be used in this dissertation. Each vessel would periodically record a data entry with a time stamp, vessel identification, vessel type, and its current coordinates. The Bayesian Network was trained with an anomalous-free dataset. For each instance, the join probability was calculated. The anomaly detection was made using the average of the joint probabilities of the last k instances. The averaged value would be considered an anomaly if it exceeded a defined threshold. However, this methodology discarded the timestamp which ignored the seasonality, conflicting with one of this dissertation main objectives. This work was enhanced by [30] by introducing timeseries analysis as well as other features such as the weather. Their approach was made by dividing the dataset into tracks, each representing a journey of a certain vessel. Thus, the anomaly detection was made for each vessel and not for the vessel's network. [31] proposed a real time anomaly detection for a wireless sensor network based in a Bayesian network that utilizes temporal and spatial correlations to differentiate faulty devices from the valid ones. It uses a dataset composed by air temperature readings and it tries to detect the faulty observations. The Bayesian network was able to capture spatial relationships between neighbor sensors to adapt to the environmental changes. The temporal correlation was incorporated by extending the Bayesian network to a Dynamic Bayesian network. The results showed that the proposed model was able to detect faulty sensors, faulty network components or even predict missing values.

3.3.3 Support Vector Machines-Based

Support Vector Machines are typically used in anomaly detection as a one-class classifier. This method consists in creating a model of one class (normal data) limiting it by the boundaries defined by the support vectors. Thus, the dataset used in this classification method is expected to have a low amount of anomalies. If an instance lies inside the learned boundaries, it is considered a normal value. On the other hand, if it lands outside the boundaries it is considered an anomaly. [32] proposed a one-class quarter-sphere support vector machine for wireless sensor networks (WSN) outlier detection. It used a dataset collected from a WSN that measured temperature, humidity, light and voltage at a 30 second interval. The proposed method takes advantage of spatial and temporal correlations that exist between sensor data to identify outliers in realtime. The authors also proposed three methodologies for updating the normal model including: updating (a) at each time interval, (b) at a fixed-size time window, and (c) depending on the previous decision results (adaptative). Furthermore, they also compared three different kernel functions to measure the similarity between two vectors in the feature space including: (a) Linear kernel function, (b) Radial basis function and (c) Polynomial kernel function. The results showed that the three kernel functions had very similar results but the model update methodology which yield the higher area under ROC for anomaly detection was the Adaptative Outlier detection. A method using one-class support vector machines to detect outages in a cellular network was proposed by [33]. The dataset used was composed by measurements made by the user equipment and contained information about its localization, signal power and neighbors. The SVM was also compared with a Local Outlier Factor detector but the first yield better results. The experiments showed that the area under the ROC reached a maximum of 98%.

3.4 Clustering-based anomaly detection

Clustering methods group similar data into clusters by distance measurements [34]. Clustering is primarily an unsupervised technique but has been applied to semi supervised learning in the past few years [35]. The most common procedure consists in defining a set of points to be the clusters centers. The test instances are grouped to the nearest cluster center. To cluster an instance as an anomaly it is necessary to keep in mind the three following assumptions [17]:

- Assumption 1 : If only normal data clusters are made, an anomaly cannot belong to any cluster, i.e do not fit well with existing clusters.
- Assumption 2: If a cluster contains both normal and anomalous data, normal instances lie closer to the cluster centroid.
- Assumption 3: If there are clusters with various sizes, the normal clusters are the ones denser or with the higher number of instances, while anomalies lie in small or sparse clusters.

The first assumption suggests that not all points must belong to a cluster. For instance, [14] proposed DBSCAN, a density-based clustering algorithm that formed noiseless clusters in the presence a noisy dataset. [36] developed a method to remove the clusters from the original data and then identify the residual instances as outliers. The second assumption assigns an anomaly score to an instance by computing its distance to the cluster centroid. [37] proposed a method where the clusters are discovered using the k-means algorithm and then outliers are removed from the dataset. They consider a point as an outlier if its distance to its clusters' centroid c is bigger than p times the mean distance of c to all the data points, being $p > 1$. Next, the clusters were recalculated without the outliers. Very similar to this method, [38] proposed an Outlier Removal Clustering (ORC), that uses k-means to cluster the data. Then, it removes outliers from the dataset by computing the instance's distance to the clusters' centroid and dividing it by the maximum distance from the centroid to a data instance. If the value exceeds a certain threshold, it is considered as an anomaly and removed from the dataset. [39] evaluated the performance of k-Means, improved k-Means, k-Medoids, EM clustering and distance-based outlier detection algorithms when trying to detect unknown network intrusions or attacks. They found distance-based outlier detection to be the method which yielded the best results with an accuracy of 80.15%. The last assumption considers a clusters as an anomaly cluster if its total number of instances is lesser than a certain threshold. [13] proposed a method to identify outliers called FindCBLOF. After computing the clusters, they classify them as large or small clusters given the number of instances each cluster holds. Then, a Cluster-Based LOF (CBLOF) value is attributed to each data object. The CBLOF is a measure for identifying the degree of each object being an outlier. If the object belongs to a small cluster, the CBLOF is calculated by measuring the distance to its closest cluster. On the other hand, if the object belongs to a large cluster, the CBLOF is calculated by the distance from the object to its cluster.

Various clustering algorithms use fixed-width clusters to find anomalies due to being a linear time ($O(N)$) approximation algorithm [17]. For instance, [40] defined a constant width for all clusters and when an instance is in test, it computes the distance to all existent centroids. If the smallest distance to a centroid is smaller than the defined width, the instance is assigned to that cluster. In the other hand, if the distance if bigger than the defined width, a new cluster is created with the instance being the centroid. The clusters are classified as anomalies if they are not part of the N percent of clusters containing the largest number of instances.

A clustering approach was made by [41] in order to find anomalies in a wireless cellular network by analyzing users call records. The authors compared a k-means approach with a hierarchical clustering one to find anomalies. They find that both yield an accuracy of 90% so k-means was chosen due to its lesser complexity.

3.5 Nearest Neighbour-based anomaly detection

The nearest neighbor-based algorithms detect anomalies by computing the distance or similarity between two data instances. The metric or parameter used to compute distances or similarities may vary greatly depending on the dataset. Typically, kth Nearest Neighbor (kNN) approaches attribute an anomaly score to a data instance given their distance or similarity to the kth nearest neighbor. [42] proposed a different approach where the anomaly score of a data instance is based on the sum of distances to the k nearest neighbors. The algorithms based in kNN are considered lazy-learning because they learn from the entire dataset instead of building a model. This method is typically costly due to the fact that for each prediction, it is necessary to learn from the entire dataset which requires a large amount of resources. The authors in [43, 44] demonstrated a light weight method to detect anomalies in computer networks using KNN and a genetic algorithm for feature selection. They created a metric measure called strangeness and assign it to each instance. The measure was calculated by dividing the sum of the distances between an instance and the k nearest instances of the same class by the sum of the distances between that instance to the k nearest instances of other classes. If the value exceeded a certain threshold it would be considered an anomaly. They were able to reach a True Positive score higher than 99%, only needing 0.14 seconds to diagnose an instance.

KNN has also been applied to anomaly detection in wireless sensor networks.Author in [45] proposed a kNN anomaly detector that solved the lazy-learning problem in order to be able to operate online. It used a dataset from a WSN that measured the temperature and humidity in 30 seconds intervals. Their solution was based in a hyper-grid structure, consisting in hyper-cubes of fixed size, where the data was mapped to. An instance y was classified as an anomaly if it had less than k instances in its detection region that was a hyper-cube centered in y. Thus, the distances to the y's k nearest neighbors was computed and if it exceeded the fixed size of the hyper-cube, it was not considered to be inside y's detection region. They were able to get 96% of accuracy but with 8% of false positives.

A time series analysis of a network's traffic was made by [46] in which they used kNN with a clustering method called Micro Clustering Outlier Detection (MCOD), to reduce the number of distance-based calculations that were necessary to be done in order to find anomalies. The author increased MCOD's performance by establishing multiple time windows within the algorithm. For any instance p, it verifies if the instance is in a range r of a micro cluster. If it is not, a new micro cluster centered in p is created, if there are at least k instances in a range r. If there are less than k instances in range r of p, the instance is considered as an anomaly. The author also used cluster density analysis to increase its results. Density is also commonly used to find anomalies. These algorithms operate by estimating the density distribution of the input space. For each instance, it estimates the density of its neighborhood and it classifies as anomalies the instances that lie in low density regions, *i.e* that have less than k instances within a radius r.

3.6 Statistical-based anomaly detection

Statistical-based anomaly detection algorithms fit a statistical model of the data and evaluate the probability of a certain instance belonging to the model or not [17]. Instances with a low probability of belonging the the model are considered as anomalies. To fit statistical models, two techniques have been deployed: parametric and non-parametric.

3.6.1 Parametric techniques

Regression models can be categorized as parametric techniques. They fit a model to the data and use the residual of each test instance to calculate the anomaly score. The majority of these methods is being applied to time-series anomaly detection. ARMA and ARIMA models [47] are the basis of most of the time series anomaly detectors. For instance, [48] used an ARIMA model to build a streaming data anomaly detection for mobile networks. It analyzed a dataset composed by the network load and the number of dropped calls. The system was built following three principles: (a) Real-time streaming (b) data agnostic and self-learning (c) lightweight and self-contained. The dataset used was composed by time series data that showed a daily and weekly pattern. The anomaly score of an instance was calculated by the difference between the value forecasted by the ARIMA model and the actual real value. Then, the obtained result was used to calculate the anomaly probability based in a statistical test that used the values of the n most recent anomaly scores. However this model only attain a precision of 53% and a recall of 73% due to the large amount of false positives.

Gaussian models are another type of parametric techniques. These models assume that the data is generated by a Gaussian distribution where the parameters are estimated using Maximum Likelihood Estimates (MLE), *i.e.* the parameters that are most likely to generate the observed data are chosen [49]. The anomaly score of an instance is computed by the distance from the data point to the estimated mean. If this value exceeds a certain threshold, the instance is classified as an anomaly. A modification of the Gaussian Mixture Model (GMM) is used by [50] in order to detect outliers in seasonal univariate network traffic data. They use GMM to represent the data as a probability density function. Then, outliers are identified and removed from the dataset by computing their distance to other points on the density scale given the PDF. The test instances are compared against the retrained GMM's without the anomalous data points. However, this method suffers when not all the outliers are removed from the dataset.

3.6.2 Non-parametric

The simplest non-parametric model is the histogram method used in [51]. This method requires normal data to build the histogram and the test instances are classified as anomalies if they do not lie on a normal bin. [52, 53] used an improved histogram method that was able to handle multi-variate data for an intrusion detection system. [54] applied a histogram based anomaly detection to a wireless network and achieved an accuracy higher than 85%. However it assumes that the network is hierarchical and static. For instance, [55] shows an anomaly detection technique based on a chi-square statistic for distance measurements given by the formula:

$$\chi^2 = \sum_{i=1}^{n} \frac{(X_i - E_i)^2}{E_i} \tag{3.1}$$

Where X_i is the observed value of the ith variable, E_i is the expected value of the ith variable and n is the number of variables. When the number of variables is large enough the χ^2 has approximately a normal distribution. Then, it uses the 3σ rule that is a parametric technique, to find the thresholds of anomalous values. The 3σ method considers a value as an anomaly if it has a distance to the estimated mean (μ) higher than 3σ, being σ the standard deviation of the distribution. The interval $\mu \pm 3\sigma$ is estimated to contain 99.7% of the data instances [56]. More recently, [57] also used the 3σ rule as well as a moving average to detect anomalies in a time series log data that showed a daily pattern.

3.7 Summary

This chapter made a review on anomaly detection methodologies applied to networked systems such as sensor networks, computer networks or ad-hoc networks that disclose relevant features. First the anomaly concept was defined. Then, the types of datasets that might be used were characterized. Lastly, the methodologies available in the literature were described and divided into four categories: classification, clustering, distance-based and statistical-based.

Chapter 4

Data Characterization

Veniam operates a vehicular network deployed in more than 600 vehicles over three different continents (Europe, Asia and North America) with several different time zones and cultures. Throughout the cities, the vehicles collect information from sensors such as the position, speed, distance and even provide Internet access to users. The data is sent from the vehicles to the Cloud either by real-time technologies or by delay tolerant ones, *i.e.* they only send the data when they are near an access point that allows them to send it by DSRC or Wi-Fi. The vast majority of vehicles belongs to STCP, a public transport company that runs the buses in Porto. Given that STCP's fleet follows scheduled routes, the data patterns are expected to be constant with some variations caused by the remaining fleets that have a more diverse behaviour. However, STCP's data also suffers some seasonal changes caused by scheduling changes (school vacations or events in the city), holidays or timezone changes.

To this study, three different data sources were analyzed, two from real-time technologies, heartbeats and sessions, and one from a delay tolerant technology, the location. The study of heartbeats was the most thorough and the knowledge was applied to the remaining, with small adjustments.

4.1 Heart Beat

The heartbeat is the network's control message and it is sent over real-time technologies while a vehicle is active. The message is composed of a timestamp and the vehicle's GPS position at the moment the message was sent. Each OBU sends a heartbeat at each minute, while the RSUs send it at every 30 seconds. When an unit looses connection to the network it tries to send a heartbeat at every 2 seconds to verify if the network is back online.

Since the data arrives by the minute, the first assessment made to the data was to verify its behavior when aggregated by the number of different nodes messaging at each minute. The data showed a high variability caused by network's delay that resulted in inverse spikes between consecutive values that, at first glance, could look like anomalies. However, the average of the two values would lead to a normal value. The first approach to remove this variability was to assess

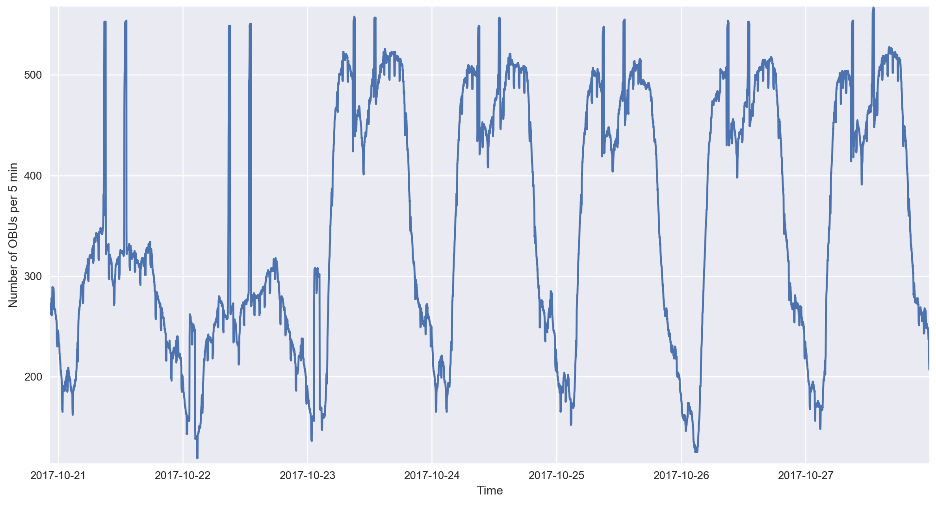


Figure 4.1: Heart beats data from all the Veniam's networks aggregated by the number of distinct devices within the last 5 minutes during a week of October.

not only the data object but the average of the values within a window of data instances. This way, the data would become robust to noise caused by spikes. However, this solution was still too noisy which led to another approach that consisted in aggregating the data by the number of different OBUs messaging within the last 5 minutes interval. In other words, the data is assessed by the minute with the sum of distinct OBUs that have sent a Heartbeat within the last 5 minutes. This way, the data can be evaluated as soon as it arrives but without the noise caused by the network's delays. Figure 4.1 depicts the described data behavior during a week in October. The data shows a weekly pattern, with a daily seasonality. The weekdays are very similar to each other with some minor differences. At weekends, the Sundays show lower values than Saturdays. The pattern shown at Holidays is different from weekdays or weekends, although it resembles the latter. Beyond that, the network is deployed in several countries that have specific holidays, turning this problem into a challenging task. It is important to notice the two spikes that happen every day. This is the result of an application called "Snooze" which is responsible to awake every OBU from 08:50 to 09:10 and from 13:50 to 14:10, even the ones that are not moving. The goal of this application is to verify if there are any problems with some devices that were supposed to awake and were not able to do it. It is possible to observe the success of the smoothing technique by the lack of unusual spikes in the data.

4.2 Sessions

The sessions data flow represents a user accessing the hotspot Wi-Fi provided by each vehicle. This dataset is composed by the session ID, the start time (time at which the user connected to the hotspot) and the end time (time that the user last sent data). The session ID is given by the OBU

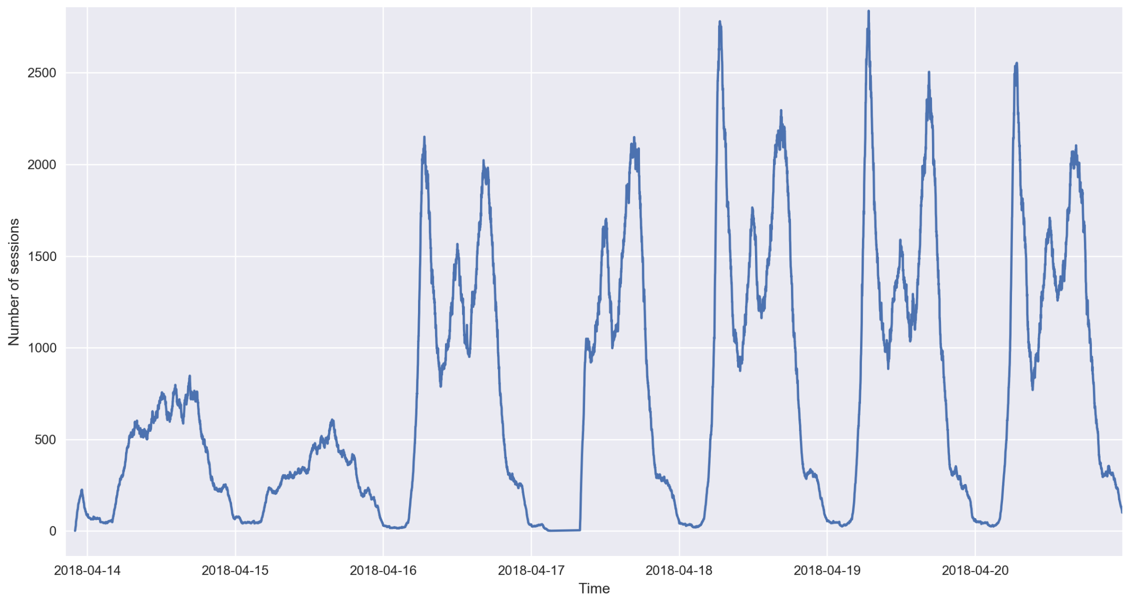


Figure 4.2: Sessions data aggregated by the number of distinct devices within the last 60 minutes during a week of April.

id and by the user. This means that a user will always have the same session id while connected to the same OBU, even when the connection drops and reconnects (until a timeout). The data is sent in real-time as soon as the session ends. The data flow also displays a weekly pattern but way noisier than the heartbeats, which is expected because sessions are created by users and they do not have a schedule as strict as the buses from STCP. To evaluate the sessions data, the aggregation of sessions per minute had to be increased from 5 minutes to 60 minutes to decrease the noise. This means that at every minute, the last hour was analyzed. Figure 4.2 depicts the number of sessions aggregated by 60 minutes interval with a 1-minute slide window. In addition to the problems that were identified for the heartbeats, this data source also suffers from having very few data for some periods of the day (late nights) and much bigger differences between weekdays.

4.3 Location

The Location data flow is collected by the second, per OBU, and it stores the vehicle position, speed and traveled distance since the last event. Considering that the routes followed by the STCP fleet are, usually, the same, the amount of distance traveled within a time interval follows a constant pattern. Given that the Location data is sent to the Cloud via a delay tolerant technology, the data analysis cannot be made using a near real-time processing such as the heartbeats. Thus, it is necessary to wait a period of time to make sure that the data already arrived at the Cloud. With that in mind, the data was analyzed by the minute but with aggregations of one hour. This means that in an online system, the anomalies could only be detected 1 hour after it happened. Figure 4.3 depicts the behavior of the aggregate data over two weeks in April. It is possible to observe that

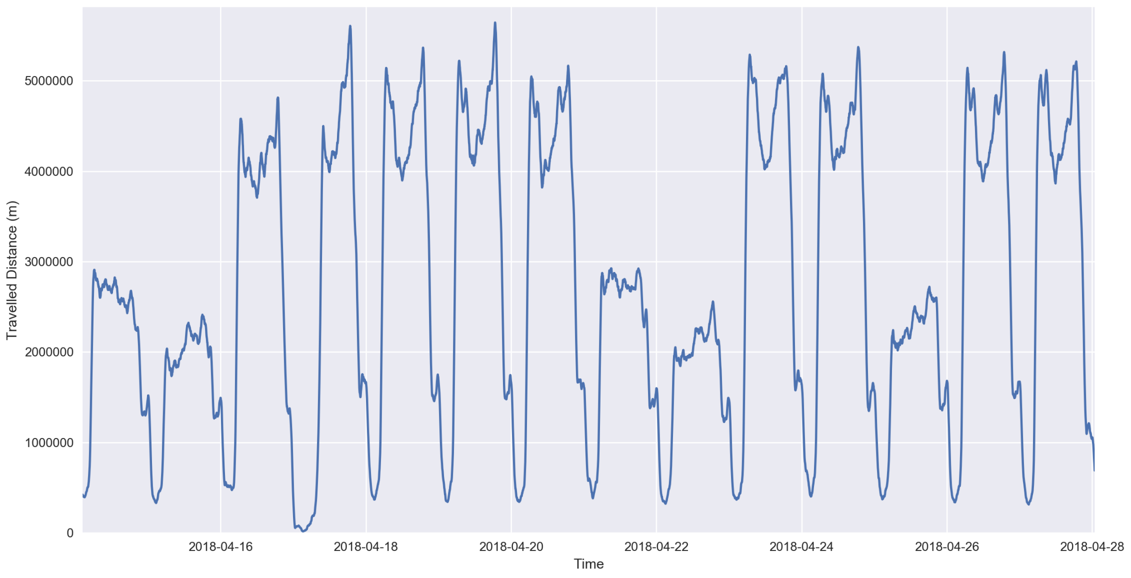


Figure 4.3: Location data aggregated by the number of distinct devices within the last 60 minutes during two weeks of April.

the pattern is similar between weekdays but, unlike the heartbeats, there are visible variations that may or may not be caused by anomalies. It is also observable that there is a clear anomaly at the end of day 16 and a Holiday at day 25.

4.4 Dataset

The data analysis was made using a dataset that comprised two months of data: September, and October. These months were selected because they held some special events that the system should account for such as Holidays (5 October), STCP schedule changes (school vacations) and a time zone change caused by the end of Daylight Saving Time (29 October). This dataset was divided into training set and testing set, with the last week of October being the testing set.

The data stored at Veniam is not labeled, thus the accuracy of an anomaly detection system cannot be made automatically but it has to resort to the opinion of a specialist. Given that such task requires too much effort, it was decided to evaluate the system using artificial anomalies. To do that it was necessary to understand what kind of problems can happen in the network. The anomalies that are easily identified are the ones that cause a complete failure of the network, where every OBU loses connectivity and is unable to deliver data. This type of anomaly is clearly identified in data sources that are sent via real-time technologies but might go unnoticed when dealing with delay tolerant ones. However, other types of anomalies can occur without being easily detected like bugs in software updates, failures within APIs, cloud problems or RSUs/cellular towers malfunctions. Taking this information into account, 2 types of artificial anomalies were introduced:

- Type 1: During a period of time, 10% of the messages that arrive are discarded or duplicated.

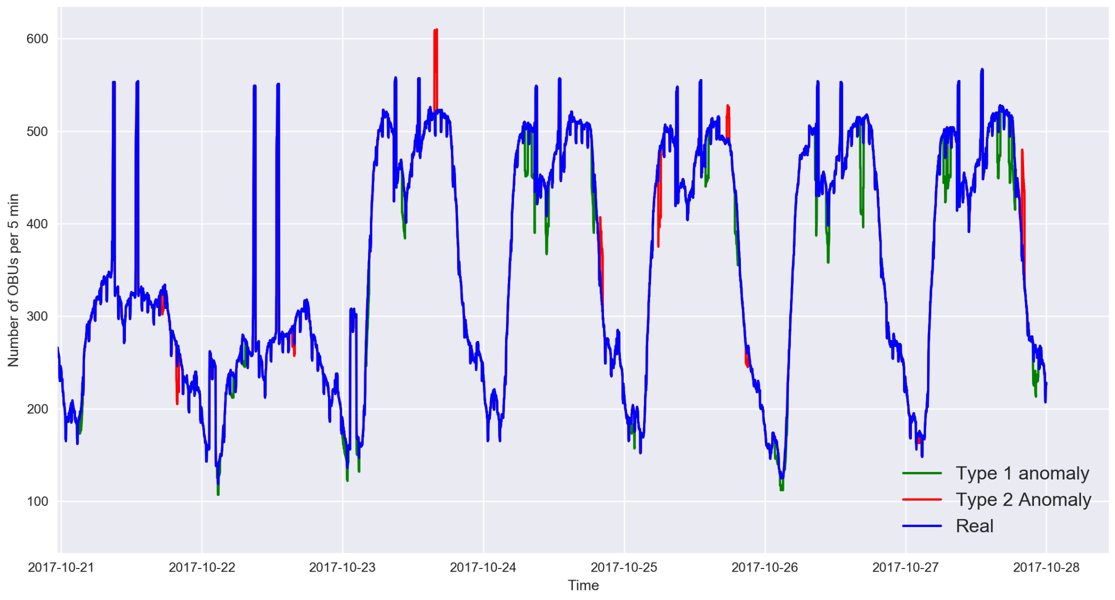


Figure 4.4: Week in October chosen as the testing set with artificial anomalies.

- Type 2: During a period of time, messages that are sent inside a geographic area are discarded or duplicated.

Using a labeled dataset with artificial anomalies that simulate malfunctions in the network, will allow evaluating the performance of the algorithm instead of just comparing it with the anomalies detected by the Veniam's operations team. Taking this into account, several artificial anomalies were randomly introduced to the selected datasets. The selected dataset for the heartbeats can be observed in Figure 4.4. The type 2 anomalies were created using four geographic areas using geohash with 5 characters. This reflects in the four areas presented in table 4.1, each with 25 km^2. To avoid anomalies that are easy to detect, only one of the areas was used at each anomaly.

Geo Hash	min Lat	max Lat	min Lon	max Lon
ez3f7	41.1767578125	41.220703125	-8.6572265625	-8.61328125
ez3f5	41.1328125	41.1767578125	-8.6572265625	-8.61328125
ez3f4	41.1328125	41.1767578125	-8.701171875	-8.6572265625
ez3fh	41.1328125	41.1767578125	-8.61328125	-8.5693359375

Table 4.1: Geohash and respective location used to make type 2 anomalies.

Chapter 5

Exploration of Anomaly Detection Techniques

This chapter presents an exploratory work towards finding the best-suited solution to the available data. Several algorithms were tested by processing a dataset of Heartbeats that was classified as normal by Veniam's Operations Team. The goal was to verify the strengths and weaknesses of each one of them so that the best ones could be selected to build the anomaly detection pipeline.

Chapter 4 presented the data characteristics as a time series with a weekly pattern, daily seasonality, and no labeled anomalies. Given that, and the literature review made in chapter 3, it was decided that the most suitable anomaly detection techniques were the ones based on time series forecasting, statistical approaches, and density-based outlier detection.

The density-based approaches assume that an instance is likely to be anomalous if it deviates from the values of its neighbors. It makes sense to use this kind of approach because STCP's fleet follows a scheduled route, what causes the number of active vehicles to be similar at a specific time of a certain the day. Thus, it is expected that a normal object lies within a dense neighborhood.

The time series forecasting uses historical data to predict future instances. This approach can be supervised since it is possible to predict data whose real values are already known. For instance, the data from October can be used to predict the values of a week in November. The forecasting accuracy can be assessed by comparing the predicted values with the real ones. However, the anomaly detection process must be, once again, unsupervised since there is no knowledge of when the prediction error reflects into an anomaly.

The statistical approaches build models based on measures of the statistical dispersion of a distribution. These methods assume that the data at specific time of the day can be modeled by a known distribution. A data instance can be classified as an outlier if it lies outside the boundaries given by measures of variability.

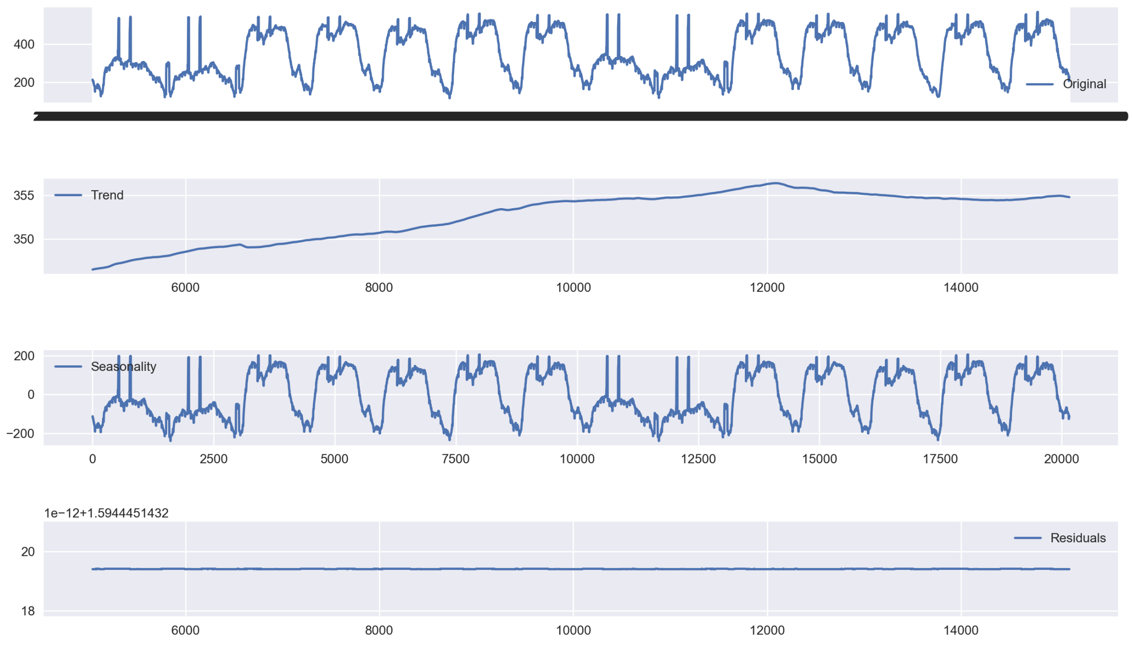


Figure 5.1: Decomposition of the testing time series into Trend, Seasonality and Residuals.

5.1 Forecasting Time Series

A time series is a sequence of data instances taken at successive equally spaced points in time. When dealing with time series forecasting, two main analysis must be made. First, it is necessary to understand the dataset in order to extract its components. Then, those components are used to tune the parameters in order to make better predictions. Normally, a time series can be decomposed in three components:

- Trend - Increasing or decreasing behavior of the series over time
- Seasonality - Repeating patterns or cycles of behavior over time
- Noise/Residuals - Variability in the observations.

Figure 5.1 depicts the seasonal decomposition of a time series from the Heartbeats data into seasonality, trend and residuals assuming an additive model. It can be seen that the trend has low variability and the residuals are near zero which strongly suggests a stationary time series. With the information retrieved, time series forecasting algorithms can be used to try to fit models on historical data and use them to predict future observations of that series. The following time series forecasting methods were used as supervised methods using the data from September and October to predict the last week of October. The goal of this study was to verify whether the algorithms were able to forecast future values with a low forecasting error so that those errors could be used to detect anomalies.

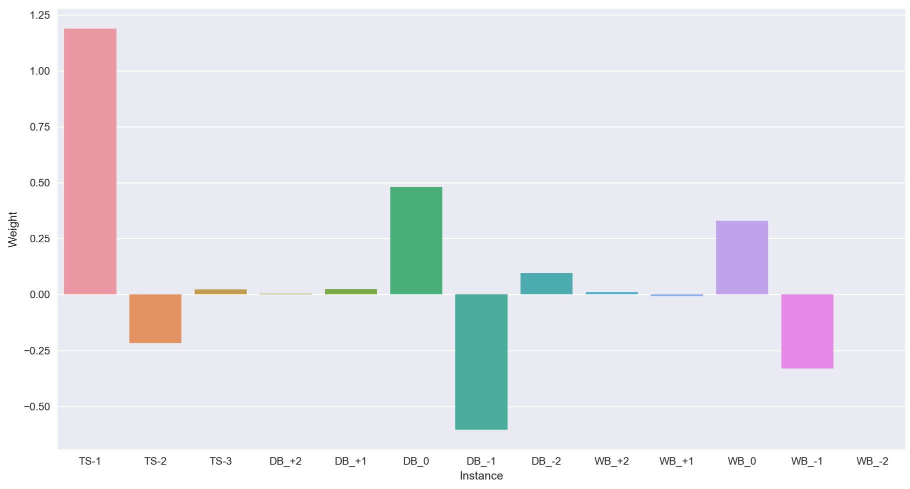


Figure 5.2: Weights assigned by the linear regression model to each historical data. TS-N represents the Nth previous instance. DB_N represents the instances of the day before and WB_N represent the instances of the week before.

5.1.1 Linear Regression

The linear regression was used to predict the future values taking into account the historical data. The algorithm was implement using Weka[1] that is software that provides a collection of machine learning algorithms for data mining tasks. It was chosen due to its easy to use User Interface. The algorithm uses a training dataset to build a model that gives weights to historical data in order to try to predict the future value. Given the data characteristics defined in chapter 4, the model was trained using the data from the previous day and the previous week, at the same time stamp. For each day, a window of 5 instances centered in the testing time stamp, *i.e.* timestamp $\pm$ 2 minutes, was considered by the algorithm. The weights assigned to each instance are depicted in figure 5.2. TS-N represents the instances that occurred N minutes before the instance that is going to be predicted. DB-N represents the instances that occurred the day before and WB-N represent the instances that took place the week before. The value of N represents the minute relatively to the one that is being predicted. For instance, if the system is trying to predict the value at 15:00, DB_0 represents the value at 15:00 of the day before and DB_-1 represents the value at 14:59.

It is possible to observe that the instances that have the most impact are the ones that occurred at the same time stamp and immediately before. It is also worth to notice that the value that happened at the same timestamp has a positive weight while the one before has a negative weight. This means that the linear regression tries to model the differences between values to predict the future one. The previous instance is the one with higher weight because it is responsible to make the prediction consistent with the values of that day. The remaining instances have a low weight and do not have a high influence on the predictions. The model was able to fit the data with a Mean Square error of **1.6** OBUs per minute. The linear regression can predict the future value

[1]https://www.cs.waikato.ac.nz/ml/weka/

very quickly because it only needs to calculate the abovementioned equation to get an output. However, the training step might be computationally heavy depending on the amount of data in the training set.

5.1.2 Support Vector Machines

The SVM for regression was also implemented for time series forecasting using the algorithm SMOReg provided by Weka. As well as the Linear Regression, this algorithm uses historical data from the day before and the same weekday of the previous week to predict the future value. The RegSMOImproved [58] algorithm was used to learn the parameters given a training dataset with historical data. The kernel function was selected using the Weka's Experimenter which allowed to execute the SMOReg using all the possible kernel function with different configuration parameters selecting the one with the best results. The Polynomial Kernel was selected with a Mean Square Error of **1.5**. Although the SVM can predict future values with a low error, it requires a higher computational power per instance than the linear regression.

5.1.3 M5 Model Tree

Due to the good results presented by the linear regression, the M5 model tree was also put to the test. This algorithm is a decision tree that fits linear regression models at every leaf of the regression tree and that every parent in the node is also associated with a regression model. It can be configured to generate a regression tree or a model tree. It is also possible to configure the algorithm to have pruning, which is a machine learning technique used to reduce the size and complexity of a decision tree, making it less likely to overfit. The model that displayed the best results was the one that generated a model tree with pruning and it yields a Mean Square error of **1.66**.

5.2 Statistical Approach

A statistical approach for outlier detection uses a probabilistic model to determine the likelyhood of an instance belonging to a distribution. Given that the data shows a well defined weekly pattern, the anomaly detection can be achieved using measures of the statistical dispersion of the historical data distribution at a certain time interval. Figures 5.3 and 5.4 depicts the data distribution in two different intervals. It is possible to observe that at different time intervals the data shows different distributions, at 00:30 the data shows a bimodal distribution while at 08:30 the data shows uni modal distribution. Thus, it is possible to conclude that the data does not displays a constant pattern throughout the day.

5.2.1 Interquartile Range

The Interquartile Range (IQR) divides the dataset into quartiles to measure the dataset variability. It uses the difference between the third quartile (Q3) and the first quartile (Q1) instead of using the

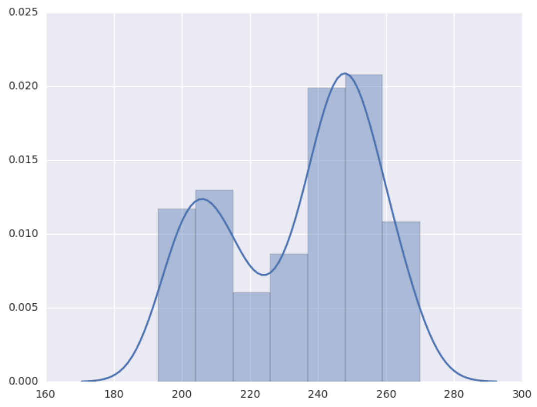


Figure 5.3: Distribution of values received during a 20 minutes interval centered in 00:30 for 22 weekdays.

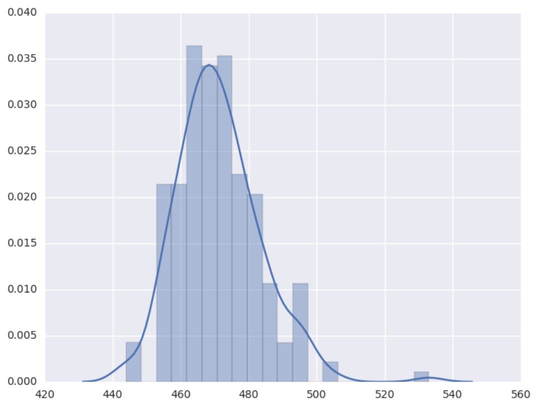


Figure 5.4: Distribution of values received during a 20 minutes interval centered in 08:30 for 22 weekdays.

total range in order to be robust to outliers within the data. A data instance was evaluated by the limits given by the IQR of the distribution of the historical data at the exact same time, multiplied by an offset N. However, this solution ended up being too sensitive to noise and it was decided to use time intervals centered in the testing time stamp with a width of 20 minutes. The size of the window was chosen empirically. This approach assumes that, within a 20 minute interval, the data remains similar. Figure 5.5 plots the behaviour of this approach using the boundaries given by Q2 $\pm$ 1.5*IQR which is a standard value used when dealing with interquartile ranges. The shaded area depicts the bounds of normality and the red dots represent the instances that lie outside the boundaries. Assuming a normal dataset, without anomalies, this model yields a 5.1% FPR. It is possible to observe that this approach is able to model the weekdays successfully but it struggles to fit the weekends.

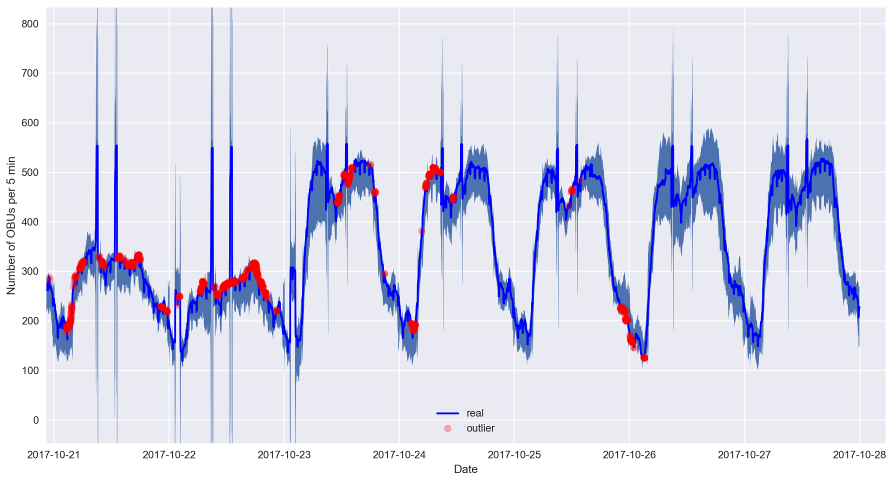


Figure 5.5: Behavior of the IQR when using 1.5*IQR as boundaries. The shaded area depicts the bounds of normality and the red dots represents instances that lie outside the bounds.

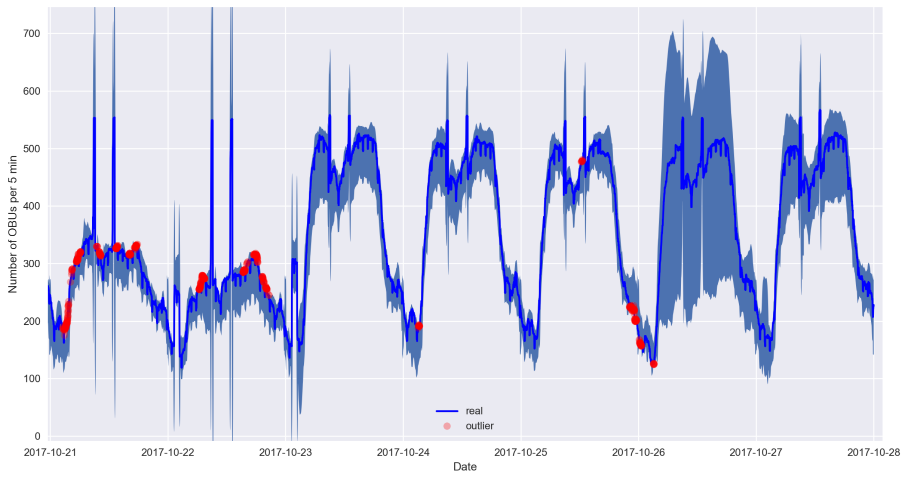


Figure 5.6: Behavior of the 3Sigma rule modeling a week of October. The shaded area depicts the bounds of normality and the red dots represents instances that lie outside the bounds.

5.2.2 3 Sigma

The three sigma rule is a statistical calculation that refers to data within three standard deviations from a mean. This rule states that 99.73% of data lies within three standard deviations of the mean if the data is normally distributed. As previously seen the data is not always normally distributed throughout the day, however the Chebyshev's inequality states that about 88 % of the instances lie within 3 standard deviation of the mean for non-normally distributed data. This approach was used similarly to the IQR, where each instance is evaluated with the distribution of the data that lied within a 20 minute interval centered in the testing instance. The figure 5.6 depicts the boundaries given by this algorithm that yield a 3% FPR. It is possible to observe that the model is unable to fit weekends and that the boundaries for Thursday are too wide. This is caused by a holiday that occurred at a Thursday (5 of October) that increased the standard deviation. Thus, it is possible to conclude that this approach is too sensitive to holidays or anomalies within the historical data.

5.3 Density-based Outlier Detection

Density-based outlier detection algorithms are built under the assumption that normal data points occur around a dense neighborhood. Similarly to the statistical approach, this technique can be used to classify an instance by measuring the distances to historical data that occurred at the same time stamp of the same day of the week.

5.3.1 K Nearest Neighbour

The K Nearest Neighbours (KNN) algorithm classifies each object by computing the similarity to its K nearest neighbors as the anomaly metric. It assumes that instances whose average distance to its nearest neighbors is lower have less probability of being an anomaly. Given the data characteristics, each instance is evaluated with the KNN among the historical data that lies within the

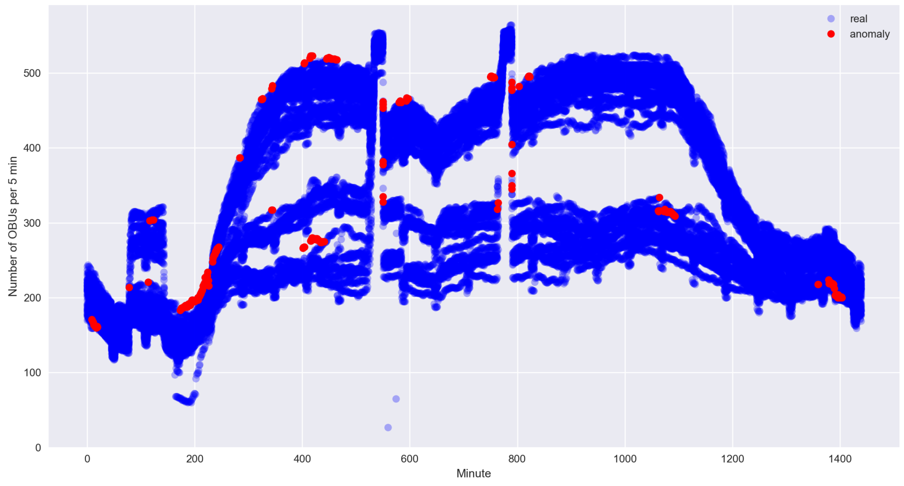


Figure 5.7: Distribution of 2 months of data per minute of the day. The blue dots represent historical and normal data while red dots represent instances from the testing week whose average distance to its 15 Nearest neighbors is larger than the defined threshold.

time interval centered in the testing object and with a width of 20 minutes (such as the IQR and the 3Sigma). To choose the value of K, it was necessary to consider that a lower K would make the method too sensitive to noise and a higher K makes it computationally expensive. Having that in mind, the 15 nearest neighbors were chosen by being the square root of the number of instances within the historical data. This approach is commonly used as a rule of thumb when dealing with KNN [59]. Several metrics can be used when dealing with KNN but for this study, the average distance to its 15 Nearest neighbors was chosen. The anomaly threshold was defined as the 99th percentile of the distribution of the average distances to the 15 NN among the training data. Figure 5.7 depicts the data behavior per minute of the day. The blue dots represent historical data while the red ones represent anomalies within the testing week. It is possible to observe that the data instances that are considered anomalies are the ones that lie in underpopulated areas. The algorithm yield a FPR of 0.96% but it is necessary to keep in mind that the KNN is a lazy learner and it uses the entire dataset for every instance. Even though the amount of data can be reduced by using a smaller window, it can grow when using too much historical data.

5.3.2 Local Outlier Factor

The Local Outlier Factor (LOF) is a density-based algorithm that classifies instances based on a distance metric called local reachability distance, explained in section 2.2.2.2. For each object, it assigns a LOF value that is given by the average local reachability density of the k neighbors divided by the object's own local reachability density. Anomalies are identified as samples that have a substantially lower density than their neighbors. Such as the previous algorithm, this algorithm uses the distance to the K Nearest Neighbors among the historical data that lies within the time interval centered in the testing object and with a width of 20 minutes. Such as the previous algorithm, the choice of K was made using the square root of the number of training instances.

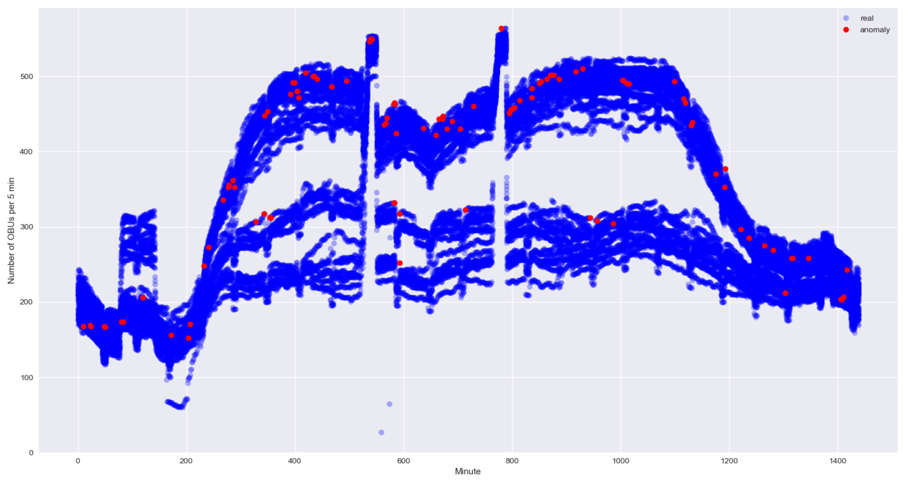


Figure 5.8: Distribution of 2 months of data per minute of the day. The blue dots represent historical and normal data while red dots represent instances from the testing week that were classified as outliers by the LOF.

This algorithm was implemented using the scikit-learn library [2] that, for a given dataset, classifies each object as normal or outlier. It uses a parameter called "contamination" that defines the proportion of outliers in the dataset. This value is used to define the threshold on the decision function. This means that the threshold is going to be biased by the expected proportion of anomalies in a dataset, what may lead to false positive anomalies when facing an anomaly-free dataset. Figure 5.8 depicts the behavior of the algorithm with a contamination of 0.01 when submitted to a week of normal data. The blue dots represent historical data while the red ones represent anomalies within the testing week. This algorithm yields an FPR of 1.08% but, unlike the KNN, this algorithm classifies as anomalies objects that lie within dense neighborhoods. This suggests that this approach may lead to a high number of False Positives.

[2]http://scikit-learn.org/

Technique	Algorithm	Low complexity per instance	Low training time	MSE	FPR
Time series forecasting	LR	✓	✓	1.6	-
	SVM	X	X	1.5	-
	M5	X	X	1.66	-
Statistical	IQR	✓	✓	-	5.1%
	3Sigma	✓	✓	-	3%
Density-based	KNN	X	-	-	0.96%
	LOF	X	-	-	1.08%

Table 5.1: Characteristics of the explored algorithms.

5.4 Solution

The trade-offs of each of the explored algorithms are presented in table 5.1. The linear regression is the only forecasting algorithm that has a low complexity per instance, however the SVM was the one with the lowest forecasting error. The statistical approaches are very similar between each other. Although the 3Sigma rule had a lower FPR, it was very sensitive to noise within historical data. The density-based approaches were also very similar but, unlike the KNN, the LOF classified as anomalies instances that lied in dense neighborhoods.

Given the dataset characteristics described in chapter 4, two possible problems could be approached:

- Detect pattern deviations
- Detect anomalous intervals

The first problem consists in detecting breaks in the normal pattern at a specific time of the day which could indicate the beginning (or end) of an anomalous period. This method would only alert the network's administrator of a possible start of an anomaly. It is possible to argue that, in an ideal situation, after the first anomaly is detected, the next one will mark the end of the period. However, it can never be guaranteed that a new anomaly occurs within the period of the first one. The second scenario consists in detecting the entire duration of the anomaly, *i.e.* alert the network's administrator of every anomalous point within the anomalous interval. This approach focuses on detecting instances that significantly deviate from the expected value. The proposed solution was focused on solving the first problem, where the administrator would only be alerted by the beginning of the period.

It was decided to divide the process into two steps: screening and classification. The screening step processes every data instance so it must be simple, fast and with low computational power. The classification step only receives objects that are flagged by screening step so it can be of higher complexity and, as a consequence, slower. Its goal is to discard the false positives without missing true anomalies. This way, the system can be able to make an online processing with some complexity. The proposed solution is presented in Figure 5.9.

5.4.1 Screening

The screening phase is the first step of the anomaly detector. It is the one by which all instances need to pass to be evaluated. Its goal is to detect all anomalous points, *i.e.* achieve a 100% True Positive Rate, but at the same time filter the maximum amount of false positives. Since this stage is going to process a great amount of data, it must evaluate each point in a lightweight manner, so that it can be fast and with low computational requirements. To detect changes in the pattern, two methodologies were selected: the linear regression for time series forecasting and a statistical approached based on the interquartile range. The statistical approach used the interquartile range to calculate the maximum and minimum boundaries of normality. Since the goal is to detect unusual breaks within the data pattern, it was decided to use the first order difference instead of

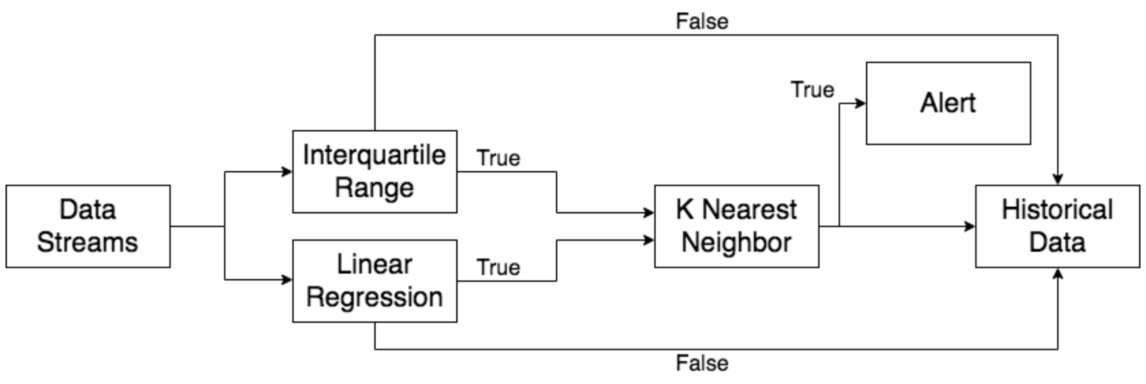


Figure 5.9: Proposed anomaly detection pipeline.

using the absolute value. The first order difference consists in subtracting the previous value to verify if the data trend is increasing or decreasing. This way, a data instance is evaluated relatively to its predecessor and if that difference is normal at that specific time of the day. This approach can be used in this first step because it only needs to check whether an instance is between the boundaries. These boundaries can be calculated offline using the historical data, making this approach fast and lightweight. In parallel to the statistical approach, it was decided to implement a linear regression model. This way, they can complement each other increasing the screening's true positive rate. The linear regression was implemented as time series forecaster that uses historical data to predict future values. Since the data shows a weekly pattern, the one-step prediction can be made using the combination of values that occurred at the same time in previous days or previous weeks. This algorithm uses the historical data to build a model and only calculates the predicted value using the equation outputted by the model. This way, it does not requires a high computational power to produce outputs promptly.

5.4.2 Classification

The Classification step processes the data that was flagged by the linear regression or by the statistical approach as an anomaly. Thus, it can be of higher complexity, since it does not have to finish the processing of an object before the arrival of the next one. Having that into account, the KNN was selected. This algorithm is a density-based approach that assumes that an instance is anomalous if there is low density within its neighborhood. It can only be used in a second step because it is a lazy-learner, *i.e.* for each value, it uses the entire data to search for the nearest neighbors, instead of building a model. Given that, it may become slow when the historical data increases.

Chapter 6

Results

The results presented in the following chapter were obtained using a dataset from the network's heartbeats composed by 2 months of training data and one week of test data. To evaluate the solution, several anomalous periods of two different types of anomalies were introduced into the testing set as described in Chapter 4. The results presented assumes that the real data **is not anomalous** and only the artificial anomalies are true positives. Although this might not be totally true, there is no way to detect if the real data is anomalous besides an expert evaluation.

6.1 Screening

6.1.1 Statistical Approach

The first approach used to detect pattern breaks was the method based on the Interquartile Range that analyzes the first order difference. This means that every instance is evaluated by the difference between its value and its predecessor. This approach suffers from the behavior of an application called "Snooze" that twice a day, from 09:50 to 10:10 and from 13:50 to 14:10, awakes every device to send heartbeats to verify the existence of failures or problems within devices. These four instances have a significant impact on the study of the differences because they display spikes caused by a sudden increase in the number of active devices that negatively affect the data. Thus, even if the value of any of those four instances lied outside of the boundaries, it would not be counted to the calculation of the FPR, avoiding having to increase the boundaries only to include those values.

September and October were used as a training set to build the IQR model. To use this statistical approach it is necessary that the data shows a constant trend, which means that historical data will be replicated in the future. The seasonal decomposition of the time series was already presented in Chapter 5 Figure 5.1, which proves that the trend has low variability. To every instance in the training set, the difference between its value and the value of its predecessor was calculated and stored in a database. To each testing object, all the values that occurred within a

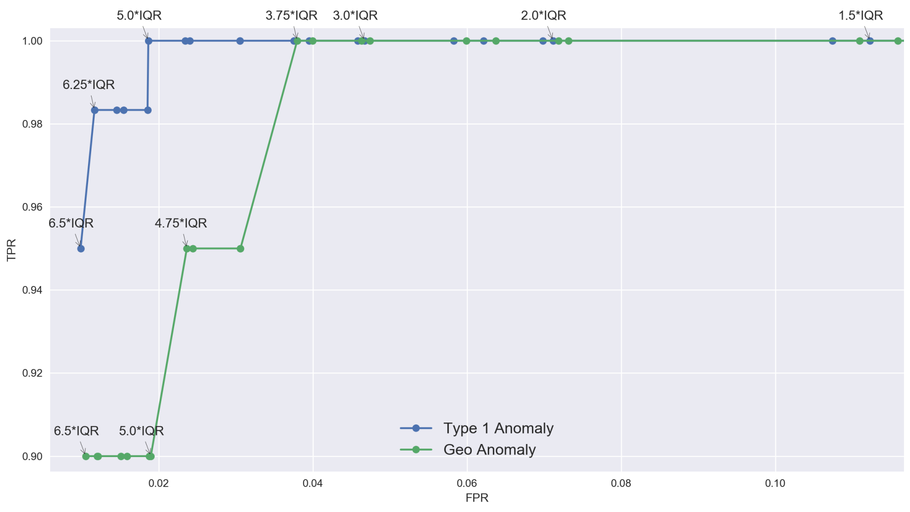


Figure 6.1: FPR vs TPR using different thresholds for the two datasets

time interval of 20 minutes centered in the testing value, *i.e.* ± 10 minutes, were used to calculate the boundaries given by:

$$max = Q2 + N \times IQR \tag{6.1}$$

$$min = Q2 - N \times IQR \tag{6.2}$$

where Q2 represents the second quartile which is also the median of the data. The size of the window was defined empirically.

This approach needs to find the value of N that best fits the data and guarantees a high TPR with a low FPR. Since the data is believed to be anomaly-free, the artificial datasets with two different anomaly types were submitted to the algorithm. To choose the best offset to which the IQR should be multiplied, an iterative process that uses different thresholds was applied. The resulting ROC curve is presented in Figure 6.1 where it is possible to see the resulting TPR and FPR using thresholds from 1.5 to 6.5 with a step of 0.25. It is possible to observe that the type 1 anomaly only needs an offset of 5 to guarantee a 100% TPR, which, ideally, a phase 1 must have, and 4% FPR. On the other hand, the type 2 anomaly (Geo) needs tighter bounds to guarantee the same TPR but with twice the FPR. Having these results into account, and the assumption that the first step is supposed to let all true positives pass, even if it means increasing the false positives, the 3.75 value was selected as the offset to be used. Table 6.1 shows the exact TPR and FPR value that this algorithm achieves with the defined parameters.

Dataset	**TPR**	**FPR**
Type 1	100%	3.73%
Type 2	100%	3.68%

Table 6.1: Statistical approach results.

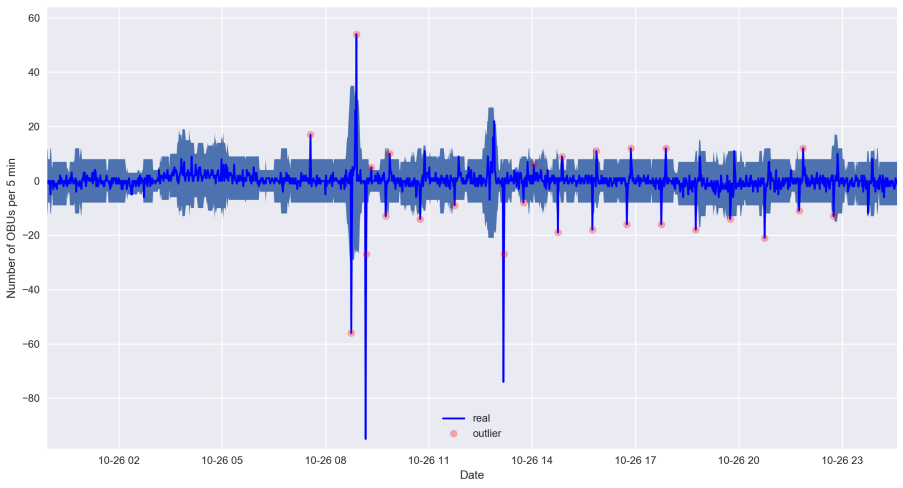


Figure 6.2: Behavior of the developed model using a threshold of 3.75 during a day of the testing week.

Figure 6.2 depicts the behavior of the first order difference during a day of the testing week without anomalies. The blue line represents the real data while the shaded area is given by equations 6.1 and 6.2, with a threshold of 3.75. The red dots display the data instances whose value lie outside the boundaries and, for that reason, are classified as anomalies. It is possible to observe that the number of anomalies is 22 out of 1440 possible instances, producing a 1.5% FPR for that day.

Figure 6.3 shows the detection of one of the artificial anomalies that belongs to the type 2 anomalies dataset, using the defined parameters. Both of the ends of the anomalous intervals are outside of the shaded area which represents the bounds of normality. However the remaining points of the anomalous interval are within the limits because this methodology is only expected to detect pattern deviations.

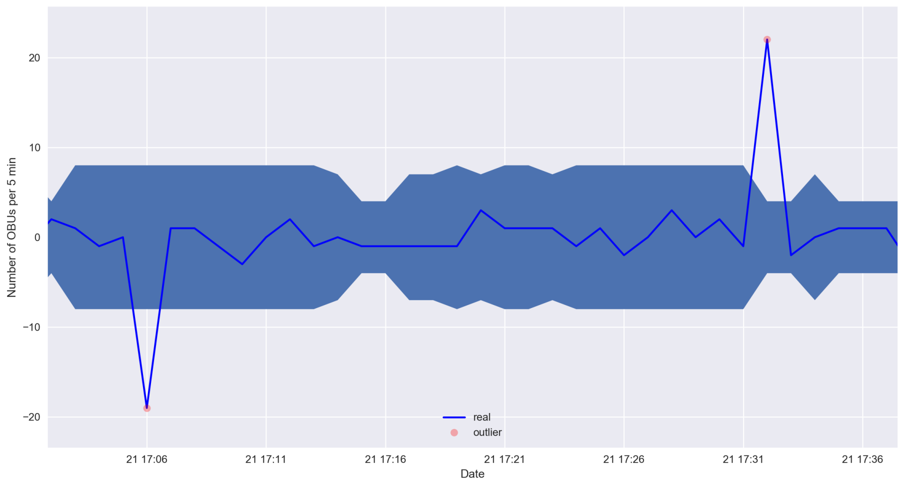


Figure 6.3: Detection of an anomaly by the developed model.

Previous days used	Best Fit
1,2,3	25.3%
1,7	10.2%
1,2,7	5.1%
2,7	21.5%
7	23.9%
1,2,3,4,5,6,7	8.5%
1,2,3,7	5.5%

Table 6.2: Percentage of times that a certain forecaster predicted the value closer to the real one.

Previous days used	Mean Abs Error
1,2,3	1.696
1,7	1.586
1,2,7	1.61
2,7	1.627
7	1.667
1,2,3,4,5,6,7	1.574
1,2,3,7	1.546

Table 6.3: Mean Absolute error of each forecaster.

6.1.2 Forecaster

The linear regression for time series forecasting was also used in the Screening step. First, it was necessary to find which combination of historical data best modeled the data in order to make accurate predictions. Several combinations were tested and the percentage of times that a specific combination was chosen as the best model is represented in table 6.2. The average forecasting error for each one of the forecasters is also presented in table 6.3. It is possible to observe that the average error is not related with the regressor that best models the data. The model that combines the three previous days was the most used but it was the one with the highest mean absolute error.

Although these forecasters yield decent results in fitting the real values, they may suffer from having anomalies within the historical data. Since the threshold for anomaly detection is relatively small, a data instance will easily be mistaken as an anomaly if the historical data deviates from the normal pattern for that time interval. Therefore, it was decided to use an optimistic ensemble that combines several forecasters selecting the one with the smallest error relative to the actual value. Let P_i be the predicted value of a forecaster i and R the real value for the same time stamp. The error of each forecaster is given by:

$$e_i = |P_i - R|, i \in 1, ..., 7 \tag{6.3}$$

Thus, the prediction given by the optimistic ensemble P_o is given by:

$$P_o = \arg\min_e (P_1, ..., P_7) \tag{6.4}$$

This approach assumes that an instance is anomalous if its value cannot be explain by any of the combinations of the previous days. This way, the model will be able to adapt to unexpected (but normal) behaviors such as holidays whose behavior is similar to weekends. If one of the days used to predict the future value suffers an anomaly, the system will use another forecaster that does not use that specific day. This way, the forecaster will be more robust to errors which leads to the decrease of the FPR. This approach was able to attain a mean absolute error of 0.9 which is notably lower than the one achieved by the forecasters individually.

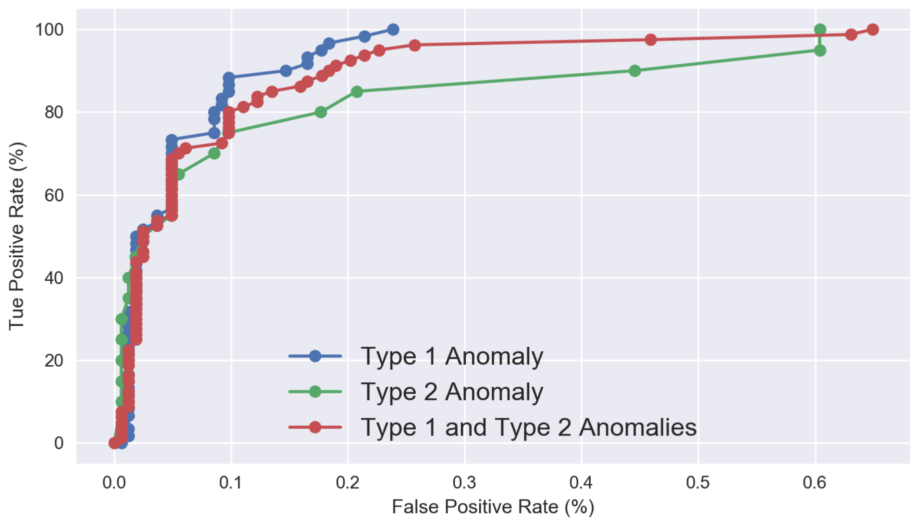


Figure 6.4: ROC curve for each of the datasets when applied to the algorithm with different thresholds.

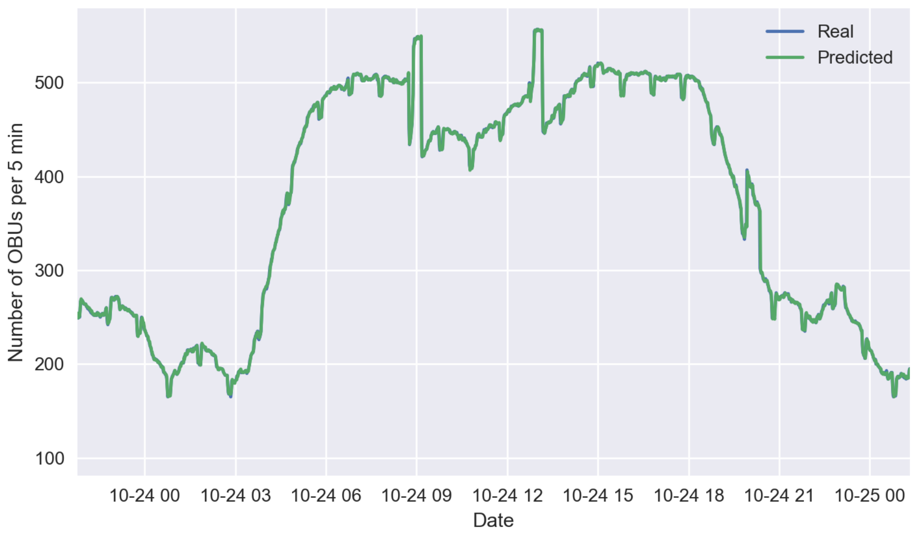


Figure 6.5: Predictions made by the forecaster approach overlapping with the real data.

To calculate the minimum forecasting error, *i.e.* the difference between the real value and the forecasted one, from which the point is considered to be an anomaly, the dataset was submitted to the optimistic ensemble using several different thresholds. The resulting ROC curve for the combined model was plotted in Figure 6.4. It is possible to observe that to achieve a 100% TPR on the dataset with both of the anomaly types, it requires a FPR of 0.65%, while letting one anomaly go unnoticed only reduces the FPR to 0.63%. The dataset with only type 2 anomalies can attain a 100% TPR with a 0.6% FPR while the type 1 dataset only requires a 0.24% FPR for the same TPR. Given that, the threshold that yield a TPR of 100% was chosen to predict the future values of the testing week. Figure 6.5 shows the predicted values using the chosen model, versus the real ones during one day of the testing week. It can be seen that there is only the green line (forecasted) which indicates that both the forecasted values and the real ones were overlapping, meaning that

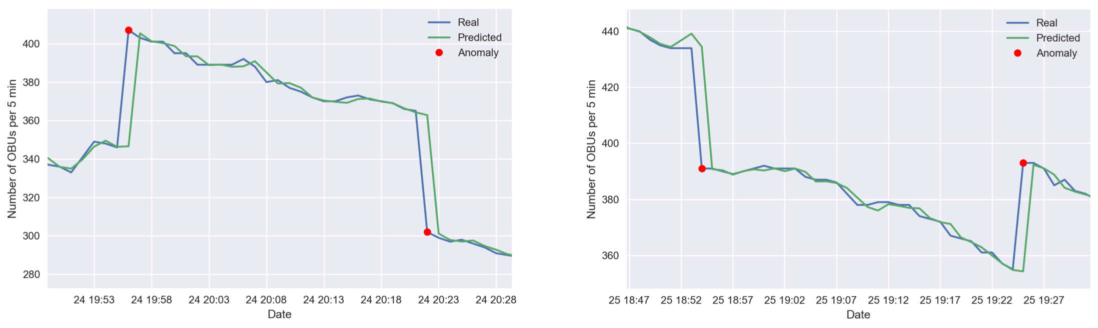


Figure 6.6: Behavior of the forecaster approach when facing two different artificial anomalies.

the model successfully fits the data. Figure 6.6 shows two types of artificial anomalies, one simulating a situation where data was duplicated and another with a scenario where data was lost. In both situations, the model forecasts a value with a considerable error compared to the actual value, which results in an anomaly. However, as in the statistical method, after the first anomalous point, the model adapts to the new "pattern" and only detects the end of that anomalous interval. In addition to the great results showed before, this approach is a great option to use in a first stage since the linear regression is a lightweight process that can quickly classify instances with a low computational power requirement. On the other hand, this model needs to be retrained, which can be computationally intensive. However, since the data shows a daily pattern, it only makes sense to retrain the model once per day.

6.2 Classification

The Classification step is used when any of the Screening algorithms claims that an instance is anomalous. The first task in designing a density-based algorithm is the characterization of the available data. In addition to the analysis made in Chapter 4, the data distribution by the minute during the 2 months of training was analyzed and presented in figure 6.7. It is possible to observe that there are three different patterns in the data, most likely caused by weekdays, Saturdays and Sundays. To prove that, a heat map with the average number of vehicles at a specific time of a certain day of the week is presented in figure 6.8. By the analysis of the image, it is possible to conclude that Saturdays and Sundays display a different number of active vehicles, but also that there are some unexpected differences. For instance, at 02:45 of Sundays and Mondays there is a higher number of vehicles than in the remainder of the weekdays. Thus, it was decided that the assessment of a certain data point was made using historical data that occurred within the same time interval on the same weekday. This solution has the downside of being unable to adapt to holidays because the pattern of a Holiday resembles a Saturday. The solution to this problem was to use the first order difference. This way, the assessment of the data is made by the difference between a value and its predecessor, being independent of the absolute value. If there is a holiday or a new fleet is introduced, the absolute values increase but the differences between values might

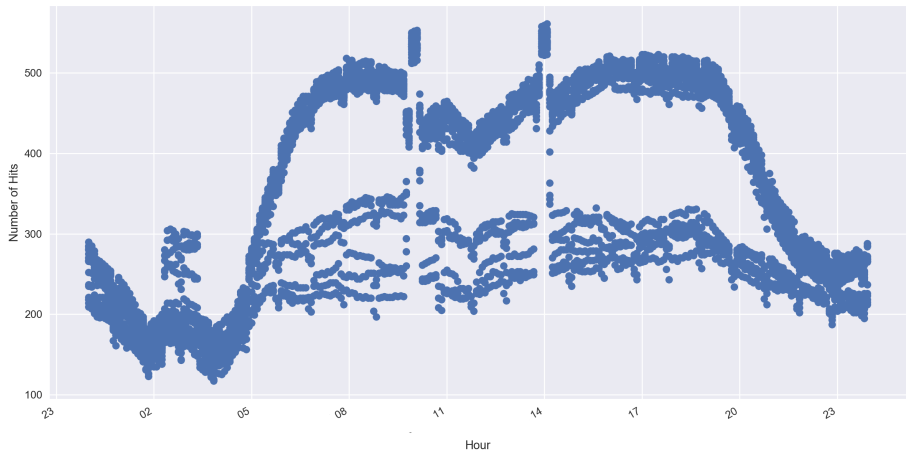


Figure 6.7: Data distribution by the minute during the 2 months of the training set.

stay constant. Another solution could be to use the library holidays[1], where the main holidays are already set, and some others can be appended so that they could be modeled as a weekend. However, although the pattern of a holiday is similar to a Saturday, it turns out to have some differences that can lead to some False Positives. This solution could be able to model the data now but it would not be able to scale because the network is deployed over several countries with several different holidays, where each one of them might have their own pattern. A possible workaround of this problem would be to build a different model for holidays in each country.

As described in section 5.3.1, this step makes classifications by analyzing the distances to the 15 nearest neighbors of each instance. Given the abovementioned characteristics and to reduce the impact of noise, the nearest neighbors are searched among all values that have occurred on the same weekday and within the time interval centered in the testing instance with a width of 20 minutes. To each testing instance, a score is calculated based on the values of the distances to its 15 nearest neighbors. If that score exceeds a certain threshold, the data object is considered anomalous. Having that into account, three different metrics were hypothesized:

- Distance to the 15th Nearest Neighbor
- Average distance to the 15 Nearest Neighbors
- Relative average distance to the 15 Nearest Neighbors

The first was rejected because it was too sensitive to a noisy neighbor, *i.e.* if it has 14 close samples but only 1 far away, it will be considered an anomaly although it is unlikely for it to be one. The second was also rejected due to the variability of the values throughout the day. Since the threshold is expected to be unique for every instance, it is more likely to exceed a defined threshold when dealing with large values because the average distance to its neighbors will also be higher. Thus, the anomalies would be more frequent at rush hours than at late nights where there are a low amount of active devices. The third metric was selected as the best solution because it

[1]https://pypi.python.org/pypi/holidays

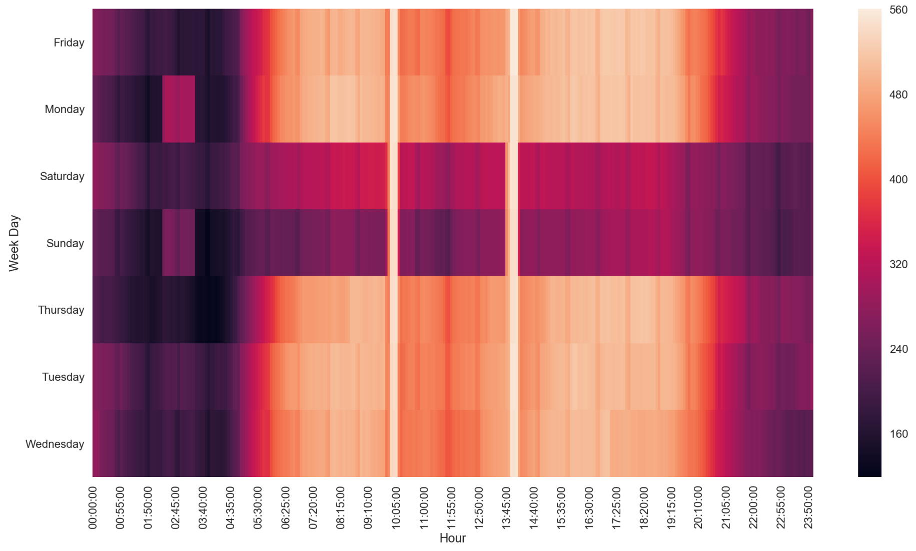


Figure 6.8: Heatmap showing the average number of vehicles at a specific minute of a certain weekday.

assesses an instance relative to the distribution of the data within that time interval. This way, the data will be on the same scale independently of the amount of active devices. Thus, the threshold can be general for all instances. This generalization is possible if the values within the interval are normalized for that distribution. Two normalization processes were considered: the min-max that changes the ends of the set and organizes the others within the new domain interval; and the z-score normalization that organizes the values using the median and standard deviation of the original interval. However, since the min-max is too sensitive to outliers within the training set, the z-score was chosen as the normalization method and it is given by:

$$v' = \frac{v - mean(V)}{stdev(V)} \tag{6.5}$$

where v' is the normalization of v that belongs to the set V, and mean(V) and stdev(V) are, respectively, the median and standard deviation of the set V.

Algorithm 1 presents the process of normalization and search of the 15 nearest neighbors.

```
for every test instance do
    Find the time interval centered in the testing value;
    Retrieve all values within that interval from the database;
    Calculate mean and standard deviation of the historical data;
    for every value within the interval do
        Normalize the value;
    end
    Normalize testing value;
    Find the 15 nearest neighbors within the normalized set;
    Calculate the average distance to the 15 nearest neighbors;
end
```

Algorithm 1: Finding the 15 Nearest Neighbors

The distribution of the scores of the training set is presented in figure 6.9. More than 70% of the instances had a score of 0.0 but it was not depicted on the image due to visualization purposes. It is possible to observe that higher scores have less frequency and that the 95th percentile is about 0.9.

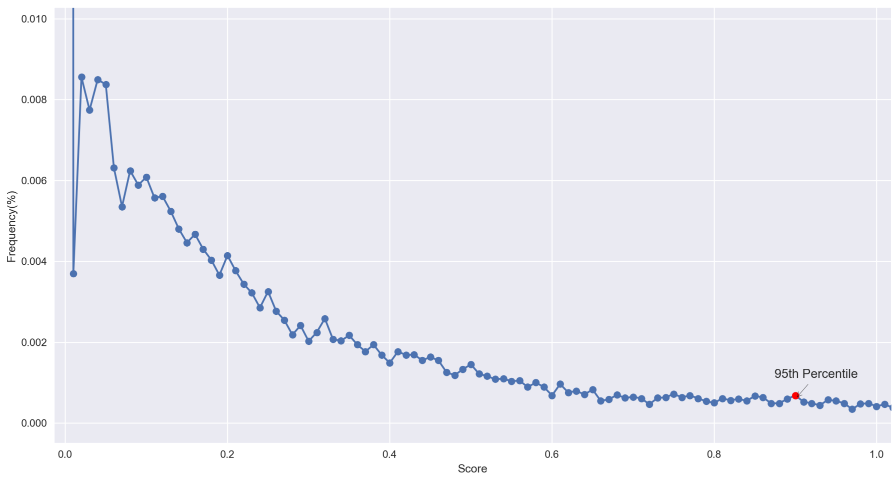


Figure 6.9: Distribution of the scores of the training set. The 0 has a frequency of about 70% but it is hidden to visualization purposes.

Each instance was evaluated using a score that was given by the average distance to the 15 nearest neighbors. To find the score that best models the data, the two artificial datasets where submitted to the algorithm in an iterative process that at each iteration increases the threshold by 0.01. The resulting ROC curve is presented in figure 6.10. For a 100 % TPR with the type 2 anomaly dataset it was necessary to have a threshold of 1.2 which resulted in a 1.2% FPR. The type 1 anomaly needed a tighter thresholds to achieve the same TPR which caused a higher FPR

(5.9%). However, if the threshold to attain a 95% TPR was chosen, the FPR would decrease to 0.6% in type 1 anomaly dataset and 0.13% in the type 2, which is a remarkable decrease.

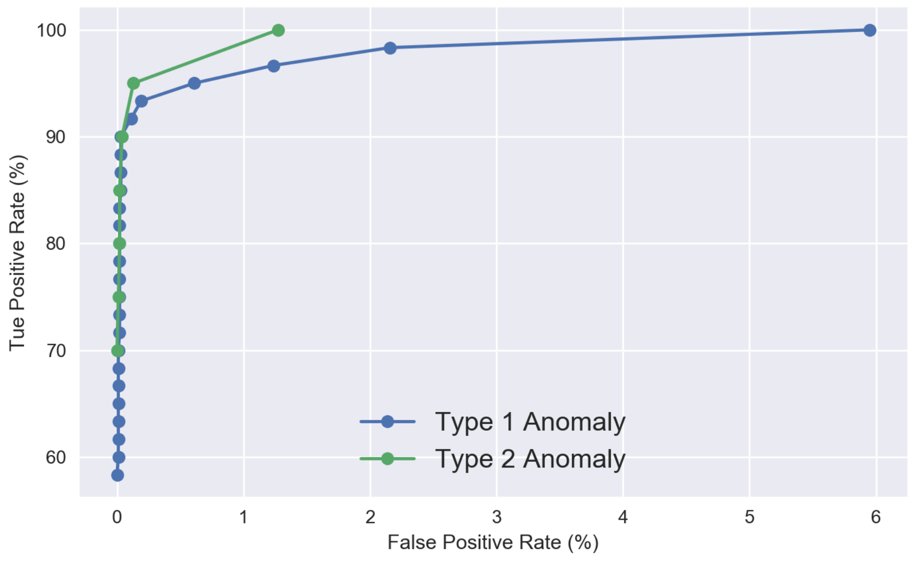


Figure 6.10: ROC curves for the KNN using datasets with artificial anomalies separately.

Since this approach requires a higher FPR to achieve the 100% TPR than the forecaster, other normalization process was tested. The following equation was used to normalize the data:

$$score = \frac{\frac{\sqrt{\sum_{i=1}^{K}(n_i - p)^2}}{K}}{\tilde{x} + 1} \tag{6.6}$$

The numerator represents the average distance to its neighbors, where p represent the testing instance, n_i denotes the ith Neighbor, and K the number of neighbors. The denominator is the median of the values inside the window from the historical data, $\tilde{x}$, plus one to avoid dividing by zero. The denominator is used to normalize the value for that specific interval. This approach is believed to be robuster than the previous because the normalization process could suffer from outliers within historical data that would increase the standard deviation. Both the anomalous datasets were submitted to the algorithm using this normalization process to choose the thresholds that attain the best TPR-FPR ratio. The correspondent ROC curve is plotted in figure 6.11. It is possible to observe that this approach is able to attain a 100% TPR, decreasing the maximum FPR from 5.9% to 2.2% relatively to the previous approach. However, at a 97% TPR, this approach is surpassed by the previous one. This means that the previous is unable to effectively detect 2 out of 60 anomalies but is a good approach to find the remaining. On the other hand, this new approach can detect the entire group of anomalies with a better trade-off but loses performance when letting some anomalies go unnoticed. Both could be used in this step but, since the goal was always to achieve a 100% TPR the second approach was chosen.

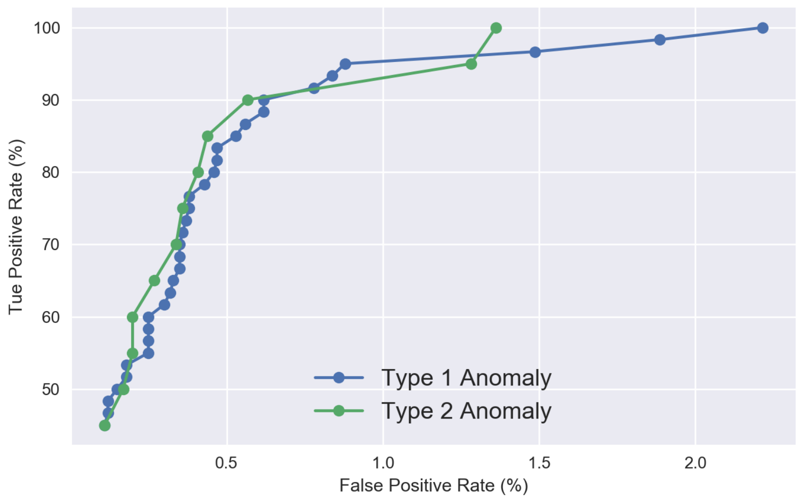


Figure 6.11: ROC curves for the KNN using datasets with artificial anomalies separately and a normalization given by the score of equation 6.6

6.3 Full Pipeline Analysis

The system consists in integrating the screening step with the classification step consecutively. This means that all the data instances will be processed by the screening step but only the ones that are positive anomalies will follow to the classification step. The previous sections assessed the algorithms separately but this aims to evaluate their combination. Table 6.4 shows the TPR and FPR of each of the components when an artificial dataset composed by 80 anomalies of both types was submitted to the system. All thresholds were tuned to achieve a 100% TPR with the lowest FPR possible, thus the KNN used the normalization defined in equation 6.6.

The final results yield a 100% TPR and a 1.3% FPR. Since the testing week is composed by 10080 data instances, this means that there is, on average, 131 anomalies per week and 18.7 per day. Since the artificial dataset has 40 anomalous intervals, there were 80 anomalies per week (the beginning and the end of the interval) which represents 11.4 anomalies per day. This means that there is, on average, a total of 30.1 anomalies per day (11.4 TP and 18.7 FP) but only 11.4 (37.9%) are true positives.

These results can be improved by discarding the Statistical approach that introduced too many false positives and only use the forecaster followed by the KNN. The results presented in table 6.5 show that the system also achieved a 100% TPR, but it decreased the FPR to 0.38%, which repre-

Step	Component	TPR	FPR
Screening	Statistical	100%	3.04%
	Forecaster	100%	0.637 %
Classification	KNN	100%	1.3%

Table 6.4: Results of the System composed by Statistical, Forecaster and KNN.

Step	Component	TPR	FPR
Screening	Forecaster	100%	0.637 %
Classification	KNN	100%	0.38%

Table 6.5: Results of the System composed by Forecaster and KNN.

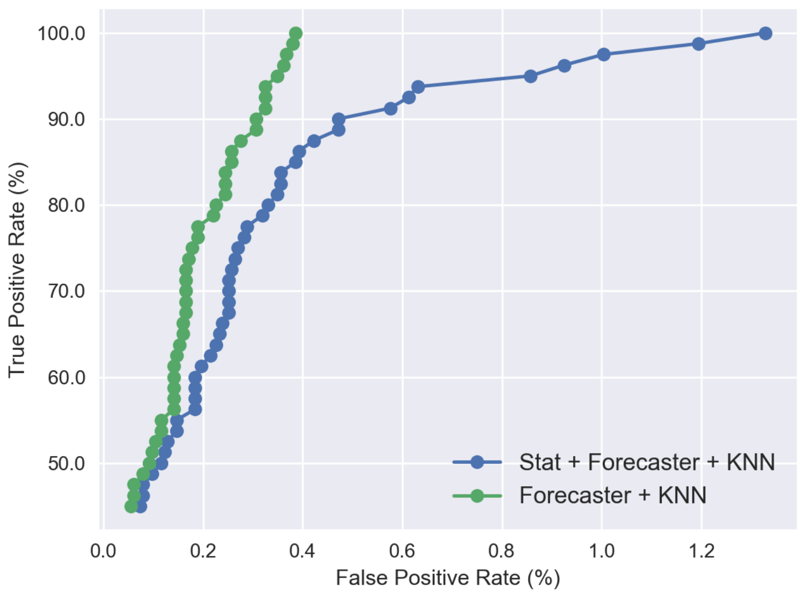


Figure 6.12: ROC curves for the system with and without the statistical application.

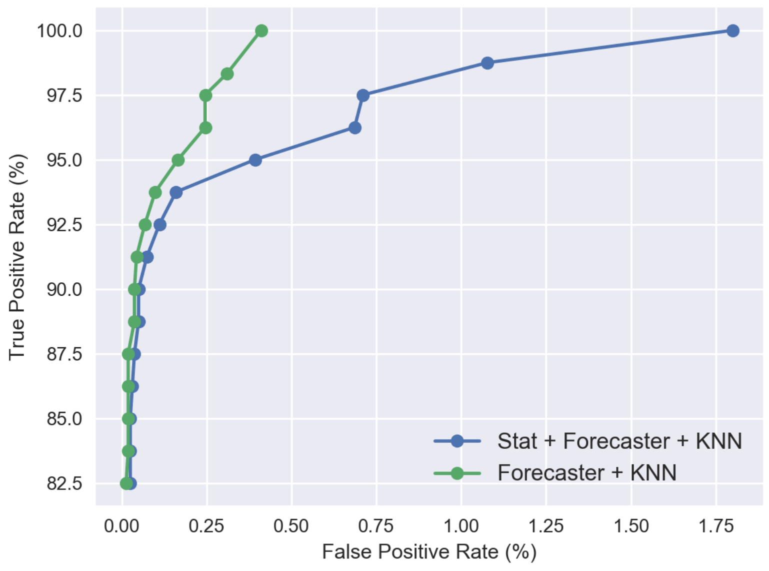


Figure 6.13: ROC curves for the system with and without the statistical application using the z-score as a normalization function for the KNN algorithm.

sents 16.87 anomalies per day, 11.4 of which (67.6%) being true anomalies. However, discarding the Statistical might cause loss of robustness in step one, leading to some true positive misses.

An alternative solution could be to allow to miss some true positives in order to reduce the false positives. The correct decision can only be made after assessing the cost of a false positive and the cost of missing a true positive. If a positive means that a worker is going to be alerted and some expensive action must be made, than the goal is to reduce the FPR. On the other hand, if missing a true positive has a large negative impact in the network, the TPR must be maximized. This trade-off must be analyzed by the users in order to define the values that have the best cost-benefit ratio.

Figure 6.12 depicts the ROC curve for the KNN algorithm when applied to the artificial dataset using different anomaly thresholds for the system with and without the statistical approach. It is possible to observe that allowing to miss 1 anomaly per week reduces the FPR from 1.3% to 1.19%. This means that the full system would have, on average, 25.94 anomalies per day, 11.4 of which being positive (43.9%). If only the Forecaster was used, the FPR would decrease from 0.38% to 0.37% which represents an increase in the true anomalies detection rate from 66.7% to 68%. Letting more true anomalies go unnoticed would decrease even more the false positives.

Although the normalization with z-score did not have the best result for a TPR of 100%, it was also tested to verify its behavior when applied to the complete pipeline. The ROC curve for the system with and without the statistical application is presented in figure 6.13. It is possible to observe that, for a 100% TPR, it achieves a 1.8% FPR when using the statistical method but only 0.4% without the statistical approach. If one anomaly could be disregarded, those values would decrease to 1.01% and 0.31%, which are lower than the previous system. Thus, it is necessary for the network administrator to choose the combination of thresholds or the normalization function that best fit the needs and the costs.

6.4 Validation

The validation process is made using a different dataset than the one used to tune the parameters. The first week of November with 80 artificial anomalies, 40 of each type, was chosen to validate the proposed solution. The dataset was submitted to the system without further parameter tuning, with the thresholds set to achieve a 100% TPR. However, the z-score normalization was chosen for the KNN. As previously demonstrated, the best solution was the one that comprised the Forecaster followed by the KNN, discarding the statistical approach. The forecaster yield a FPR of 0.71% and a TPR of 97.5%, missing two out of the 80 anomalous objects. The KNN was applied to the Positive instances and reduced the FPR to 0.44% maintaining the TPR. Figure 6.14 presents the result over the validation week. The green dots represent the true anomalies (Artificial) while the red dots represent False positives.

It is possible to observe a clear real anomaly within the data, when the number of active devices goes to 0. The algorithm is able to detect that failure and for that reason those instances can be considered as true anomalies, decreasing the FPR to 0.407% (5.86 anomalies per day). Another problem that can be observed in the image is that the 1st of November is a holiday, so it makes a Wednesday to behave differently from normal. Although the algorithm is able to deal with the majority of instances of the holiday, it cannot deal with the night between day 1 and day 2 that also behaves like a holiday/Weekend. The algorithm does not expect the sudden increase in active devices because it is not normal of a Thursday, so it leads to a large amount of false positives depicted in Figure 6.14. It is also possible to observe that the snooze application is also very sensitive to this algorithm. Although it is expected to have a big difference at that time of the day, the values can vary too much, some times leading to false positives. Without the anomalies that

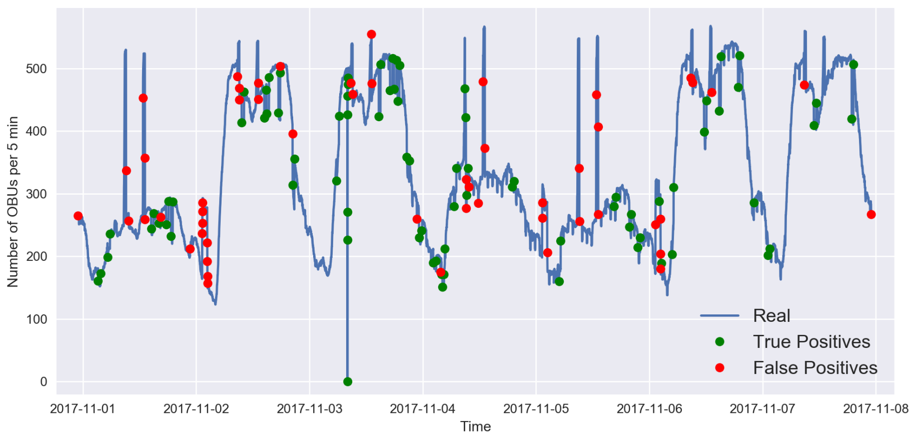


Figure 6.14: Behaviour of system when submitted to the validation week. Red dots represent false positives and green dots represent true positives.

occurred at snooze time, that is a characteristic of Veniam's network only, the FPR would decrease to 0.27% which would represent an average of 3.8 anomalies per day.

6.5 Application to Other Data Sources

Beyond the traditional validation process, the algorithm was also validated using different data sources to verify if this approach could be generalized to all available data. The system was applied to Sessions and Location data. As referred in Chapter 5, these datasets were aggregated in 1 hour intervals and evaluated minute by minute. Given that, when an anomaly is introduced it will have a low impact on the aggregated value because only one of the 60 minutes is anomalous. As the test window includes more anomalous minutes, the anomalies gain impact, forming a "U" shaped pattern (Figure 6.15), unlike the heartbeats that showed a visible break. As the proposed algorithm is focused on detecting pattern breaks, it is harder to detect anomalies if the anomaly occurs more smoothly.

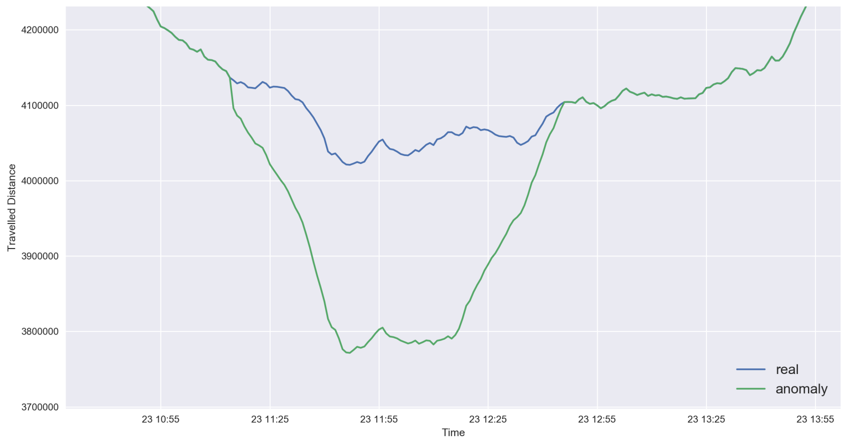


Figure 6.15: Type 2 anomaly in location data.

The location data was aggregated with the total distance travelled within an hour. Since this is completely different from the Heartbeats data, it was necessary to make a similar study to the one made for the Heartbeats to find the best suited thresholds. The three algorithms were submitted to a week of April with artificial anomalies and 2 month of historical data were used. Figure 6.16 depicts the ROC curve for each one of the three algorithms of the system and for the system as a whole. A 100% TPR entails a high FPR for all the approaches, and for visualization purposes it was removed from the plot. If one anomaly is allowed, the FPR notably decreases for all algorithms. To evaluate the entire system, the thresholds for a 95% TPR were chosen due to the great FPR reduction. With the chosen parameters, the statistical approach had a FPR of 4.8% and the Forecaster yield a FPR of 3.5%. This lead the Screening step to filter approximately 94% of the instances. The Classification step had a TPR of 95% with a FPR of 2.67%. If the statistical approach was discarded, and only the forecaster was used in the screening step, the FPR would decrease to 1.2%. If one more anomaly could be supported (90% TPR), the resulting FPR would decrease to 0.404%. Once again, this must be a system variable that the Network Administrator can tune to best fit his interests. To validate this solution, a new dataset with artificial anomalies was submitted to the system without further parameter tuning to achieve a 95% TPR. Figure 6.17 depicts the outputs of the system that was able to achieve a 93.5% TPR with a 0.3% FPR. It is possible to observe that a real anomaly occurred during the 6th of May but the system was able to detect it.

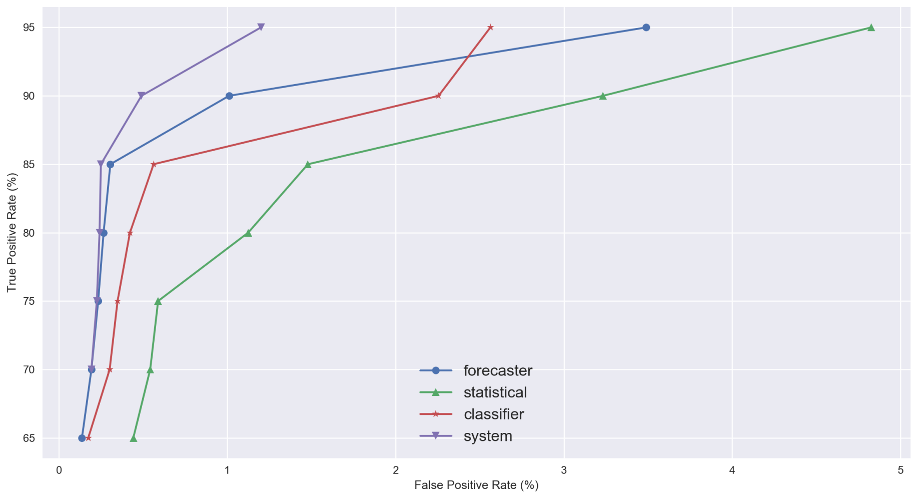


Figure 6.16: ROC curves for each of the algorithms independently and for the system as a whole when processing the location data.

As described in Chapter 4, the sessions are too variable due to the unpredictable amount of users accessing per minute. On average, there are 15000 sessions per day which means that the average number of sessions per minute is, approximately, 10. It is also worth to note that, on average, each vehicle only gathers 40 sessions per day, which reflects in 5 sessions per hour (8 hours of work). This means that, at rush hour there is a significant number of sessions while at nights the number of sessions is much lower. As said before, the developed solution only detects

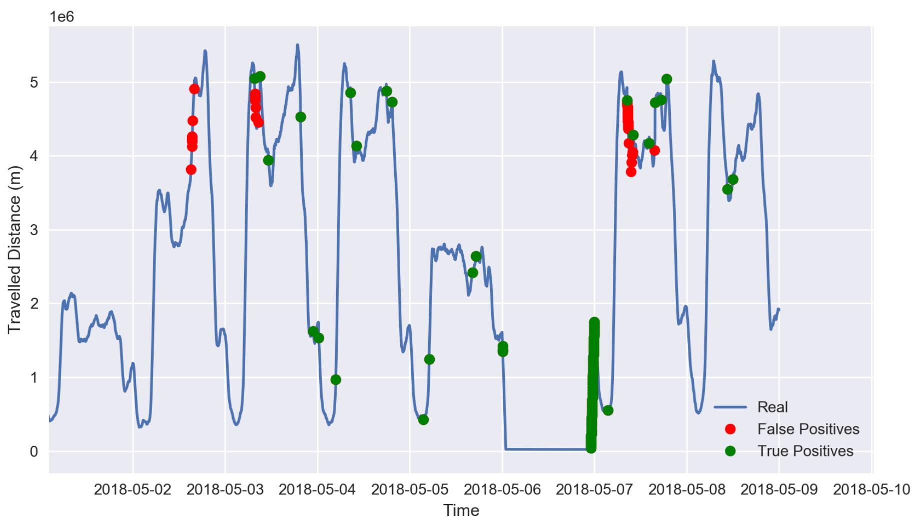


Figure 6.17: Validation location

breaks in the pattern, thus to detect an anomaly a minute of data must have some impact which is not the case in this scenario. Such a low amount of data leads to an imprecise data model that is unable to detect the anomalies. The ROC curves presented in figure 6.18 demonstrate that the developed system is unable to model the sessions data effectively. A solution to this problem could be to increase considerably the aggregation period of the sessions data which would remove the online characteristic of the proposed solution, but would make the data more steady.

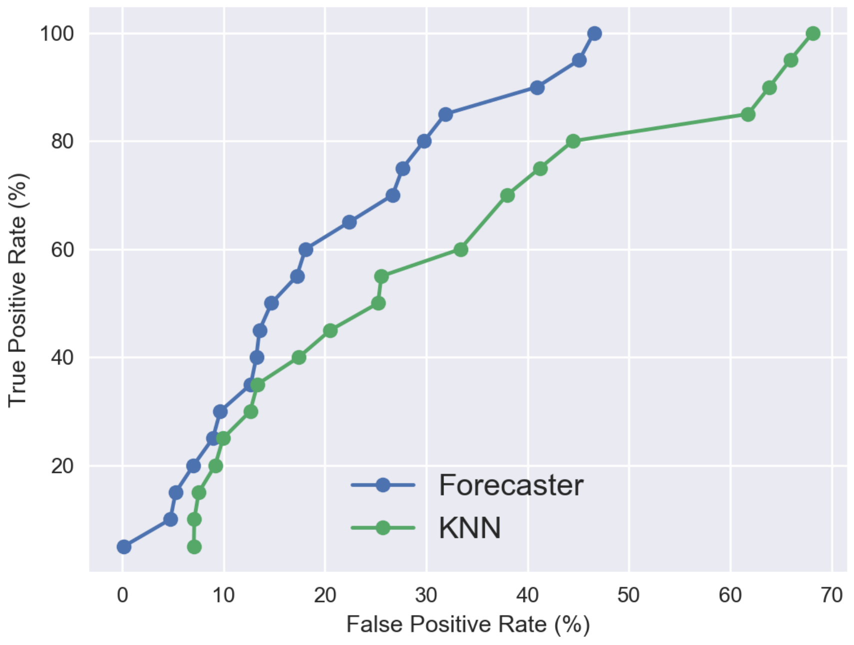


Figure 6.18: ROC curves for the Forecaster and for the K Nearest Neighbor when trying to detect anomalies within Sessions data.

6.6 Detecting Entire Anomaly

The developed algorithm was able to detect pattern breaks within data, however that approach was unable to identify the entire duration of the anomaly. In order to achieve that, the statistical approach and the KNN algorithm were modified to use the absolute value instead of the differences. The forecaster cannot be used to detect the entire duration of the interval because it adapts to the new pattern.

The ROC curves of the two independent algorithms when trying to detect the entire anomalous interval is presented in Figure 6.19 (left). It is possible to observe that, as expected, the TPR/FPR ratio is worse than the one achieved by the pipeline that only detects pattern breaks. For instance, to achieve a 80% TPR it is necessary to have a 44.7% FPR on the Statistical approach and a 36.8% on the KNN, which is an unbearable solution. In order to improve this results, the anomaly detection pipeline suffered a slight modification, depicted by the flow chart in figure 6.20. The screening step was composed only by the Forecaster whose job was still to detect pattern breaks. When the Forecaster detects an anomaly, the Statistical approach is used to evaluate the following instances until 5 objects in a row are considered normal. This value was chosen empirically. Each instance that is flagged by any of the steps is forwarded to the KNN. The forecaster was configured to have a 100% TPR as described in section 6.1.2. Then, to find the best thresholds for the Statistical approach and for the KNN, an algorithm was developed to find the resulting TPR and FPR of each combination of thresholds between the two algorithms. The ROC presented in Figure 6.19 (right) represents the best results, *i.e.* lowest FPR for each TPR, achieved by the system.

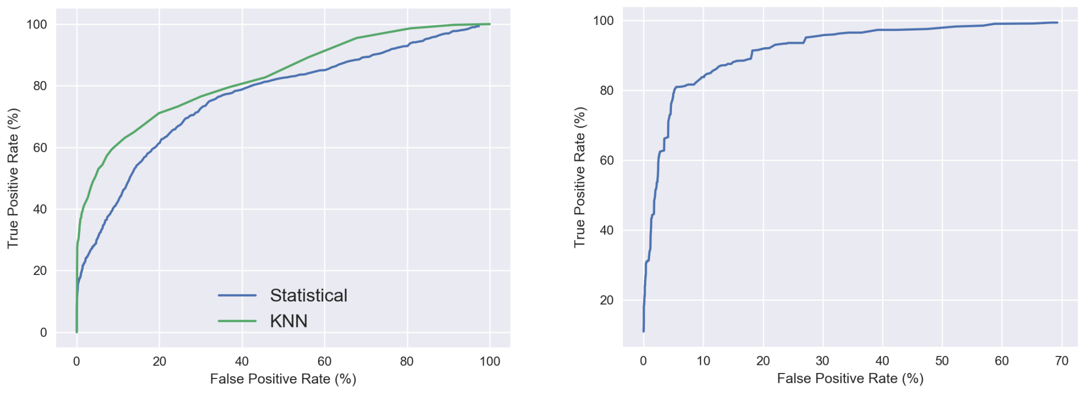


Figure 6.19: ROC curve of the statistical approach and the KNN individually (left) and the entire system (right).

Although the usage of the anomaly detection pipeline reflected great improvements, the FPR was still to high compared to the one achieved by the previous approach. This algorithm uses absolute values so it will also be unable to adapt to holidays or the addition of new fleets. Beyond that, it introduces too much logic in a solution that is supposed to be implemented as a distributed application. In order for these applications to be efficient, the algorithm must be able to process data as a batch, which cannot be made in this scenario. Thus this approach resulted in worse

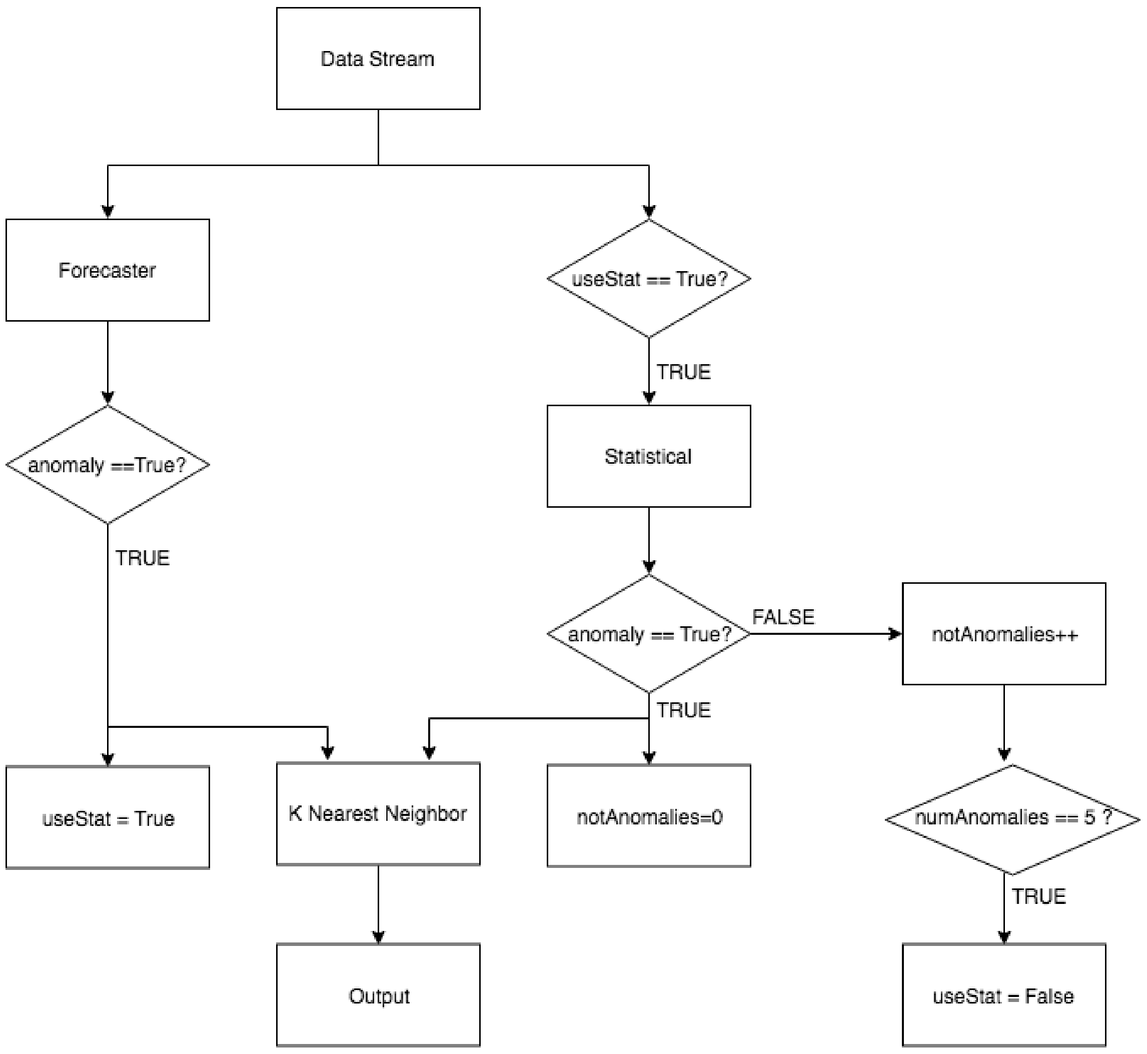


Figure 6.20: Flow chart describing the algorithm variation.

results in every aspect and it only makes sense if detecting the entire duration of the anomaly is crucial.

Chapter 7

Implementation

The proposed solution was implemented as streaming anomaly detection on Veniam's Distributed cluster located at Microsoft Azure Cloud Computing Platform [1]. The cluster architecture is presented in Figure 7.1 and it is composed by Kafka as a message broker, Apache Spark for distributed data processing, Parquet over HDFS for distributed data storage and Zeppelin Notebooks for data visualization and analysis.

The anomaly detection application was supposed to be a single application executing 24 hours per day. However, the application would end up crashing due to reaching the maximum number of processes inside a JVM. Since this problem was suspected to be caused by structured streaming architecture and not by bad coding, it could not be solved. The solution was to deploy the application as a lambda architecture, depicted in Figure 7.2. This kind of deployment takes advantages of both stream and batch processing by separating the system into different applications. Normally, this architecture is composed of three layers: a speed layer, a batch layer, and the serving layer.

The speed layer application was called **Ingestion Application** and it consisted in consuming and aggregating the data in real time using Spark Structured Streaming, storing the aggregated

[1] https://azure.microsoft.com/en-us/

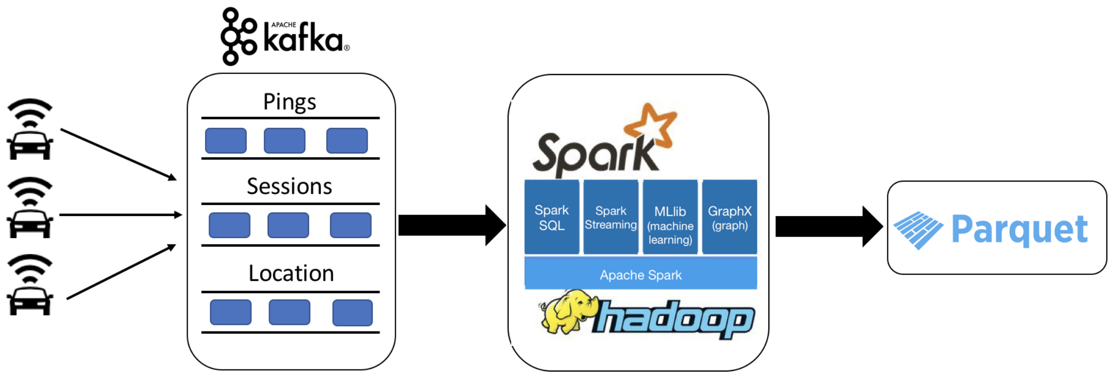


Figure 7.1: Data Processing pipeline.

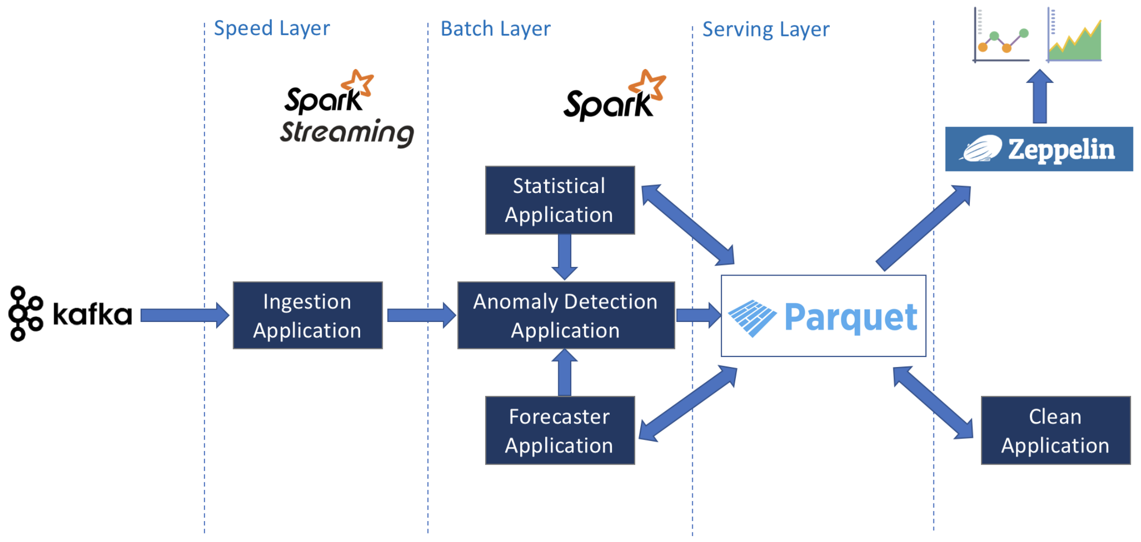


Figure 7.2: Lambda architecture of the proposed solution.

data into the serving layer which, in this case, was a file in Parquet format inside HDFS.

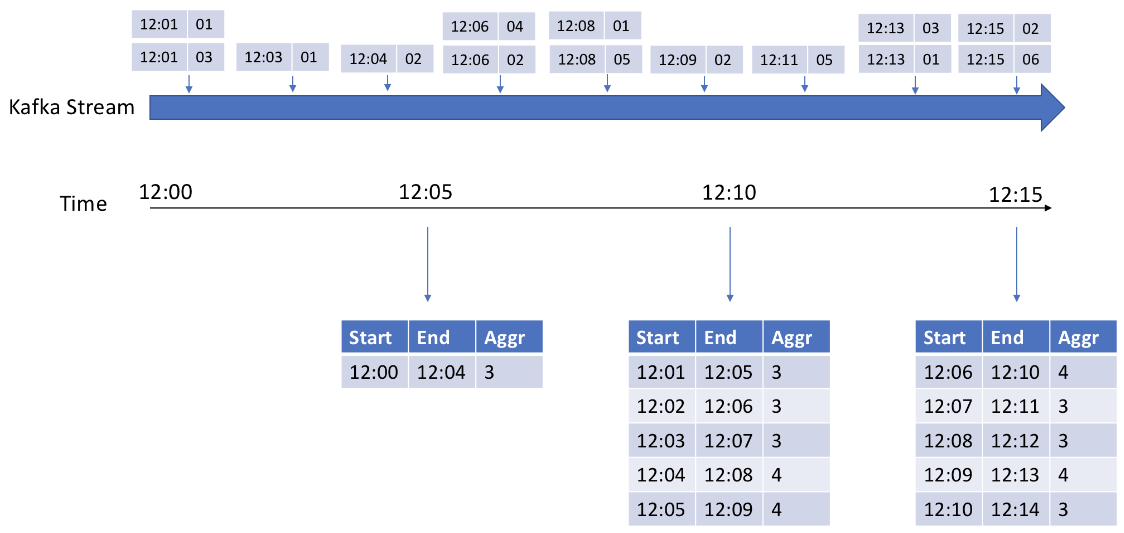
The batch layer was composed of three applications: The **Forecaster Application**, which reads data from the serving layer and trains several linear regression algorithms and stores them on the serving layer.

The **Statistical Application**, that reads data from the serving layer and for each time stamp, it calculates the interquartile range for all the values that occur within the interval centered in that timestamp with a specific width. It stores the results as a table where each row stores the maximum and minimum thresholds for every minute, from 00:00 to 23:59, of a specific day of the week.

Lastly, the **Anomaly Detection Application** reads the most recent data and verifies if it is within the limits defined by the statistical application and the forecaster application. If one of them considers the point as an anomaly, it is submitted to the KNN classifier. The output (anomaly or not anomaly) is stored in the serving layer using a Parquet table.

7.1 Ingestion Application

The Ingestion application consumes data directly from Kafka using Spark Structured Streaming which is a fault-tolerant stream processing engine built on the Spark SQL engine. It treats data streams as unbounded tables where data is being continuously appended as soon as it arrives. When a query is made to the table, a batch-like process is applied to the unbounded table as if it was a static table. This way, the streaming computation will be very similar to a batch process allowing to use Spark SQL queries to the data streams. A trigger of five minutes was defined, which means that at every five minutes the aggregation function will be made to the newest data. The data was aggregated by counting, at each minute, the number of different OBUs that sent a Heartbeat within the past 5 minute interval. Structured Streaming also allows to deal with late data



Start	End	Aggr
12:00	12:04	3

Start	End	Aggr
12:01	12:05	3
12:02	12:06	3
12:03	12:07	3
12:04	12:08	4
12:05	12:09	4

Start	End	Aggr
12:06	12:10	4
12:07	12:11	3
12:08	12:12	3
12:09	12:13	4
12:10	12:14	3

Figure 7.3: Structured Streaming ingestion and aggregation process.

by using a watermark, W. This feature lets the engine keep track of the state until W minutes. If data arrives within W minutes after the end of the interval it will be aggregated but if it arrives later than the threshold it will be dropped. Since Heartbeats are sent at each minute by the OBUs, it was defined a watermark of 1 minute. Figure 7.3 illustrates the execution of the streaming process when aggregating Heartbeats from the network. Each message from the Kafka stream is composed by a timestamp and an ID. The first output only gathered data from one interval and it outputted the aggregated value of 3 because the device with ID 01 sent two messages within the interval. The following outputs already presented 5 aggregation values, one per minute.

7.2 Forecaster Application

The Forecaster Application is responsible for training the linear regression models that will be used in the optimistic ensemble. As described in section 6.1.2, the Forecaster uses 7 parallel linear regressions, each of them using different historical data to predict the next value. The Forecaster Application uses Spark SQL to retrieve the last two months of aggregated data from Parquet and to build a dataset where each row stores the real value as the row label, and that point's historical data as the features. This dataset is given to each one of the forecasters that choose the respective features and train a linear regression model that is stored in HDFS using Parquet. Since there is a daily and weekly pattern in the dataset, the forecaster only uses historical data that occurred in a timestamp closer to the one being forecasted. Thus, it only makes sense to retrain the models once per day.

Value	Lag_1	Lag_1440	Lag_2880	Lag_10080
400	399	402	400	395
399	399	401	399	394
399	398	398	397	400
398	398	395	392	394

Value	Lag_1	Lag_1440	Lag_2880	Lag_4320
400	399	402	400	403
399	399	401	399	408
399	398	398	397	391
398	398	395	392	396

Figure 7.4: Datasets to be used to make predictions in the forecaster that uses the two previous days and the previous week (left) and the forecaster that uses the three previous days (right).

7.3 Statistical Application

The Statistical application implements the algorithm described in 6.1.1, where the data behaviour at a specific time stamp is analyzed by verifying if the result of the difference between the current value and the previous one is within the thresholds. Chapter 6 showed that the Statistical application only increased the FPR without improving the results. Nevertheless, it was implemented since it may be useful in the future and can easily be plugged in or not. The application uses Spark SQL to retrieve two months of aggregated data from parquet. For each time stamp it calculates the interquartile range of all values, of that specific week day, that lie inside the time interval centered in that specific time stamp with a width of 20 minutes. For instance, if the testing instance occurred on a Friday at 00:30, the algorithm calculates the interquartile range (IQR) of all values that occurred between 00:20 and 00:40 on Fridays. To calculate the thresholds it multiplies the IQR by 3.75 as defined in section 6.1.1. The output consists in a table where each row stores information about the maximum and minimum thresholds for a specific minute of a specific day of the week.

7.4 Anomaly Detection Application

The Anomaly Detection application runs every five minutes and it analyzes the latest instances that were aggregated and stored by the Ingestion application. For each of the new data points, it verifies if it is inside the boundaries defined by the statistical and the forecaster application. For instance, to evaluate the aggregated value that resulted from the number of distinct devices that sent an heartbeat within the interval [00:20;00:25[, the anomaly detection application loads the table created by the statistical application and filters the row that matches the center of the testing interval. For this case, it would filter the row that match 00:22 with the boundaries from 00:12 to 00:32. Then, it verifies if the aggregated value is within the boundaries defined for that specific time stamp. Next, it loads the forecasters trained by the Forecaster application and submits to each one of them a dataset where each row has the historical values that are needed for each forecaster to make predictions. Figure 7.4 presents two examples of datasets that could be submitted to two forecasters, the left one could be used in the forecaster that uses two previous days and the previous week, and the one on the right could be used in the forecaster that uses the three previous days.

After calculating the predictions, it calculates the difference between the forecasted value and the real one, for all the forecasters. If the lowest error among the seven forecasters is greater than 8, which is the threshold defined in section 6.1.2, it considers the point as an anomaly. If any of the two applications claim that an instance is anomalous, it is forwarded to the Classification step. The k Nearest Neighbour algorithm is not part of the ML/MLlib, which is a library that implements distributed machine learning algorithms to run on Apache Spark. Given that implementing a distributed KNN from scratch would be a heavy task, the Weka JAVA API was used instead. Since Weka is not distributed, the Classification step was executed only by the driver what implied to pull the necessary data from the workers to the driver. Although this is not an advisable coding habit when dealing with Big Data, it was guaranteed that only two months of data would be pulled to the driver, that was the necessary data to execute the k Nearest Neighbour defined in section 6.2. If the score given by the KNN lies outside the threshold defined in section 6.2, the point is considered to be an anomaly and a message is sent to a Slack channel in order to warn Veniam's operations team.

7.5 Clean Application

The Parquet file format provides columnar storage that decreases search time when dealing with a great amount of data by only loading the columns needed. It also stores metadata files for each Parquet file that contains the locations of all the column metadata start locations [60]. This way, it only opens the files that have the required data. Thus, it is advisable to store large files when using Parquet format in order to decrease reading time. However, the Ingestion application creates new Parquet files at each trigger, what origins multiple small sized files. Over time, the Anomaly Detection application would increase its runtime execution due to high overhead caused by reading several Parquet files. The Clean application was developed to solve this problem by running once a day, grouping small-sized files into a single Parquet file, partitioning the data by year, month and day. This tuning made sure that the Anomaly Detection application would always end its execution before the trigger of the next iteration.

7.6 Results and Performance

The proposed solution executed during a week of June. The score given by each subsystem of the anomaly detector was stored in Parquet so that it would be possible for the network administrator to change the thresholds to verify if a certain data instance would still be considered an anomaly. Figure 7.5 depicts the behavior of the system using the Forecaster followed by the KNN, with the thresholds tuned to achieved the 100% TPR described in chapter 6. The red dots represent the False positives detected by the system. The anomaly detector achieved an FPR of 0.28% but it is possible to observe that the system struggles with data instances during the Snooze interval. It is also possible to see that a real anomaly occurred between 10 AM and 17 PM on Friday and the system was able to successfully identify it. If those instances were considered True Positives, the number of False Positives during the entire week would decrease from 28 to 16. Only 4 of those 16 do not lie in Snooze interval. There were no anomalies identified by the Veniam's operations team during this week besides the one that occurred during Friday.

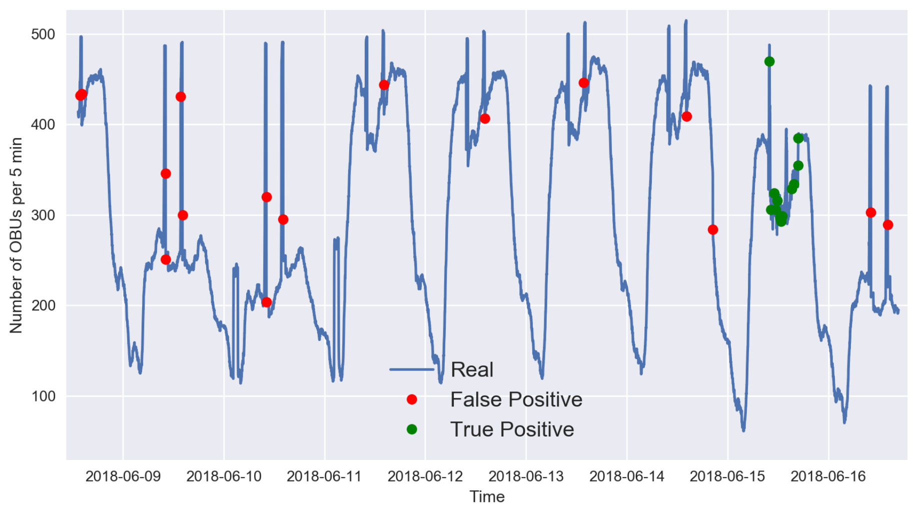


Figure 7.5: Outputs of the system during a week running on the distributed cluster. The red dots represent false positives, while the green ones represent true positives.

The implementation of the algorithm on Veniam's distributed cluster was also intended to make the solution able to scale. To prove that, two different applications were executed: one aggregating the Heartbeats data and other aggregating the Location data. As described in section 4, the Heartbeats data is sent by the minute while the Location data is sent by the second, making the later harder to process. Only the Ingestion Application needs to be studied to prove the scalability because it is the one who is going to receive larger amounts of messages when the fleet increases. All the other applications will always receive the same amount of data, independently of the amount of active devices, since they only process aggregated data.

Figure 7.6 depicts the Cumulative Distribution Function (CDF) of the number of input rows processed by each of the applications while figure 7.7 shows the CDF of the duration of the ingestion application per job for each application. It is possible to observe that the Heartbeats have fewer input rows per application than the Location data. This increase of about 6 times the number of input messages per application only increased the execution time of each application in about 5 seconds. This can be explained by the Spark's distributed processing and the Location aggregation function being the sum of the distances which is lighter than counting distinct devices as in Heartbeats. Nonetheless, the large increase in the number of messages processed is still enough to prove that the developed system is able to scale for future fleets. It is also worth to mention that the CDFs of the heartbeats suffered from the network anomaly that occurred on Friday, that increased the number of received messages and processing time. This happened because when a device looses connection it sends a Heartbeat to verify if the connection is up again. That process can occur several times if the network is constantly failing and coming back online.

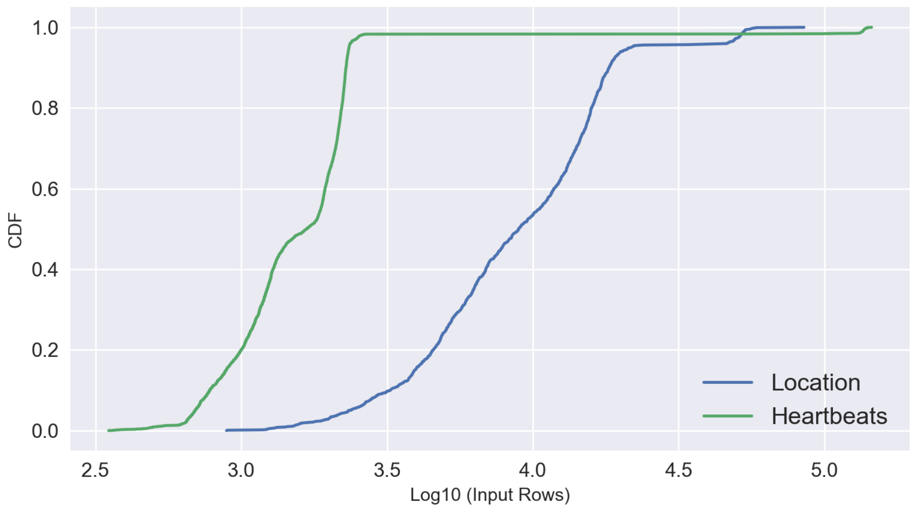


Figure 7.6: CDF of the number of input rows per job in each application using a logarithmic scale.

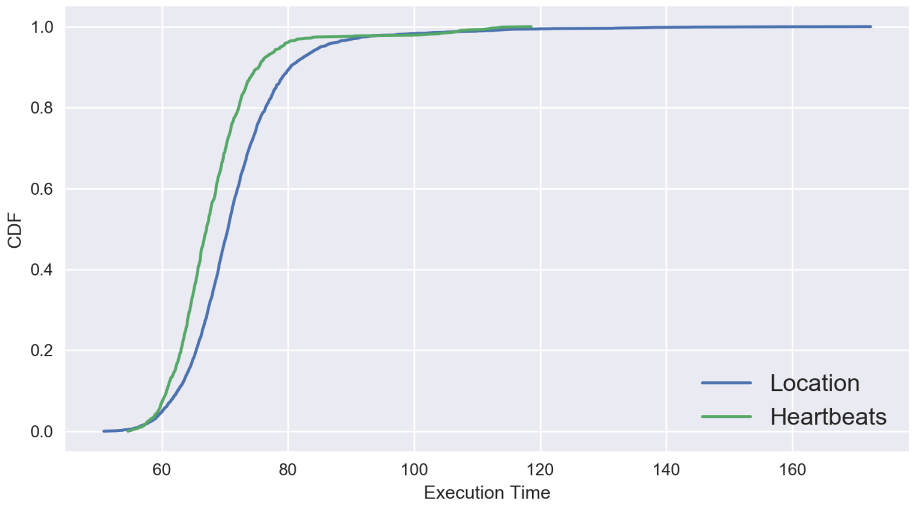


Figure 7.7: CDF of execution time per job in each application.

Chapter 8

Conclusion

This dissertation proposed a new method for an online detection of anomalies in a vehicular network by analyzing the first order difference between data streams. The goal was to develop a scalable system able to detect anomalies in a dynamic network where the topology is constantly changing and nodes are only active when the vehicle is moving.

First, the available data was analyzed and three different data sources, sent from different networks interfaces, were selected: Heartbeats, Location and Sessions. Each one of them was characterized in terms of patterns and behavior. It was seen that every data source displayed a weekly pattern with a clear difference between weekdays and weekends. Due to these factors, several anomaly detection algorithms from techniques such as time series forecasting, density-based outlier detection, and statistical approaches were tested to verify their trade-offs when dealing with the available data. The developed solution combined different data analysis approaches to build a pipeline that comprised two stages: a screening step followed by a classification step.

The screening step was composed by two models: a combination of several linear regressions that used different historical days to forecast the future values, and a statistical approach that calculated bounds of normality using the interquartile range of the data. The classification step comprised a K Nearest Neighbor algorithm that normalized the values within a specific time window to calculate the 15 Nearest Neighbors of a testing instance. The anomaly detection was made using a score given by the average distance to its 15 Nearest Neighbors.

A dataset from the Heartbeats with artificial anomalies was submitted to each one of the models to assess their True Positive Rate and False Positive Rate individually and as a System. The developed solution was able to model the data effectively with a low amount of false positives. The algorithm was able to deal with unexpected behavior such as holidays and schedule changes. The system was validated using a dataset from the Location source, that is completely different than the Heartbeats data. It was seen that the system was able to detect the anomalies without a significant increase in the False Positive Rate, proving that the system was able to model different kinds of data sources. On the other hand, the Sessions data was used to show the system's weaknesses.

The proposed solution was implemented on a distributed cluster running Apache Spark that ingested data directly from a Kafka message broker. The goal was to make an online classification of the data with the ability to scale. It was proven that the implemented system was able to deal with an increasing number of messages per minute without increasing the execution time.

8.1 Contributions

The proposed solution complied with all the requirements. It is able to detect anomalies in different data sources from a planetary-scale vehicular network that operates in different countries with different time zones and cultures that increase the network's dynamic.

The developed system is focused on the differences between values in order to be independent of the absolute values so that the impact of holidays or the introduction of new fleets can be handled.

It is able to detect anomalies in different data sources completely independent from each other, that are sent to the cloud via real-time or delay tolerant technologies. The heartbeats can be used to monitor the part of the network that uses cellular interfaces while the location can monitor the other part of the network that is sent by delay tolerant technologies, allowing to monitor the entire network.

The results suggests that this system can be reliably used as the first tool of the network's monitoring due to its high detection rate with low amount of false positives. This is an improvement over using only fixed thresholds provided by monitoring applications such as Nagios.

The results were shown using a sensitive analysis of all system parameters in order to provide guidance to a network administrator. This way he can choose the thresholds that best fit the desired TPR-FPR ratio to fulfill its needs.

The solution was implemented as a distributed application in a production environment in order to produce outputs in near real time, alerting the Network's Administrator of a possible anomaly within the network. Beyond that, the system is able to deal with a large amount of vehicles messaging at the same time, without increasing the execution time and using a low amount of resources.

8.2 Future Work

During the development of this dissertation several problems were identified that might be solved with the addition of the following improvements:

- Separate model for holidays - The holidays are specific of certain countries which might increase the network's behavior as soon as more countries are added to the network. They have special patterns that resemble weekends. However, they show some differences that also affect the pattern of the next day, leading to some False Positives.

- Dealing with Snooze - The Snooze application awakes every device during a period of 20 minutes. The time where the application starts and ends are favorable to the appearance of anomalies because there is a big difference between values in a short period of time. Even though this behavior is expected, some instances are still classified as anomalies. An improvement would be to train a classifier so that an instance at that time is less likely to be anomalous.

- Combine data sources - The developed system was tested using separate data sources. However they could be combined in order to verify if it would yield better results.
- Reduce the needs for thresholds tuning - One of the big disadvantages of the developed system is the need to recalculate the thresholds for every data source. Thus, it could be improved if the thresholds were normalized for every data source.
- Increase aggregation period - The sessions data was not able to be modelled by this solution. However, if the aggregation period was larger, the data could become more steady and improve the algorithms performance. On the other hand, it would remove the near real time characteristic of the system.
- Build a labeled dataset - Since this algorithm presented good results, it could be used to build a labeled dataset, enabling the usage of new approaches such as supervised techniques.

References

[1] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. Technical report, Cisco Internet Business Solutions Group (IBSG), 2011.

[2] John Gantz and David Reinsel. The digital universe in 2020: big data, bigger digital shadows, and biggest growth in the far east. Technical report, IDC, 2012.

[3] Apache Hadoop. URL: `http://hadoop.apache.org/`.

[4] Apache Spark. URL: `https://spark.apache.org/`.

[5] Spark Streaming. URL: `https://spark.apache.org/streaming/`.

[6] Apache Storm. URL: `http://storm.apache.org/`.

[7] Hassnaa Moustafa and Yan Zhang. *Vehicular Networks: Techniques, Standards, and Applications*. Auerbach Publications, 1st edition, 2009.

[8] J. J. Blum, A. Eskandarian, and L. J. Hoffman. Challenges of intervehicle ad hoc networks. *IEEE Transactions on Intelligent Transportation Systems*, 5(4):347–351, 2004.

[9] P. Bedi and V. Jindal. Use of big data technology in vehicular ad-hoc networks. In *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1677–1683, 2014.

[10] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Elsevier, 2012.

[11] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[12] K.G. Mehrotra, C.K. Mohan, and H.M. Huang. *Anomaly Detection Principles and Algorithms*. Springer International Publishing, 2017.

[13] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9):1641 – 1650, 2003.

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[15] V. Barnett and T. Lewis. *Outliers in statistical data*. Wiley, 1978.

[16] Dhruba K Bhattacharyya and Jugal Kumar Kalita. *Network Anomaly Detection: A Machine Learning Perspective*. CRC Press, 2013.

[17] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):15:1–15:58, 2009.

[18] Sourav Mazumder, Robin Bhadoria, and Ganesh Chandra Deka. *Distributed Computing in Big Data Analytics: Concepts, Technologies and Applications*. 2017.

[19] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[20] M. Thottan and Chuanyi Ji. Anomaly detection in ip networks. *IEEE Transactions on Signal Processing*, 51(8):2191–2204, 2003.

[21] T. Dunning and E. Friedman. *Practical Machine Learning: A New Look at Anomaly Detection*. O'Reilly Media, 2014.

[22] James Theiler and D. Michael Cai. Resampling approach for anomaly detection in multispectral images. In *IN PROC. SPIE*, pages 230–240, 2003.

[23] Naoki Abe, Bianca Zadrozny, and John Langford. Outlier detection by active learning. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 504–509. ACM, 2006.

[24] Ingo Steinwart, Don Hush, and Clint Scovel. A classification framework for anomaly detection. *J. Mach. Learn. Res.*, 6:211–232, 2005.

[25] J. Cañedo and A. Skjellum. Using machine learning to secure iot systems. In *14th Annual Conference on Privacy, Security and Trust (PST)*, pages 219–222, 2016.

[26] P. Casas, A. D'Alconzo, P. Fiadino, and C. Callegari. Detecting and diagnosing anomalies in cellular networks using random neural networks. In *2016 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 351–356, 2016.

[27] Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at uber. pages 103–110, 2017.

[28] L. Reznik, G. Von Pless, and Tayeb Al Karim. Signal change detection in sensor networks with artificial neural network structure. In *Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety*, pages 44–51, 2005.

[29] F. Johansson and G. Falkman. Detection of vessel anomalies - a bayesian network approach. In *3rd International Conference on Intelligent Sensors, Sensor Networks and Information*, pages 395–400, 2007.

[30] Steven Mascaro, Ann E. Nicholso, and Kevin B. Korb. Anomaly detection in vessel tracks using bayesian networks. *International Journal of Approximate Reasoning*, 55(1, Part 1):84 – 98, 2014.

[31] Ethan W. Dereszynski and Thomas G. Dietterich. Spatiotemporal models for data-anomaly detection in dynamic environmental monitoring campaigns. *ACM Transactions on Sensor Networks*, 8(1):3:1–3:36, 2011.

[32] Yang Zhang, Nirvana Meratnia, and Paul J. M. Havinga. Ensuring high sensor data quality through use of online outlier detection techniques. *International Journal of Sensor Networks*, 7(3):141–151, 2010.

[33] A. Zoha, A. Saeed, A. Imran, M. A. Imran, and A. Abu-Dayya. Data-driven analytics for automated cell outage detection in self-organizing networks. In *11th International Conference on the Design of Reliable Communication Networks (DRCN)*, pages 203–210, 2015.

[34] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Pearson Education, Limited, 2014.

[35] Sugato Basu, Mikhail Bilenko, and Raymond J. Mooney. A probabilistic framework for semi-supervised clustering. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 59–68. ACM, 2004.

[36] Dantong Yu, Gholamhosein Sheikholeslami, and Aidong Zhang. Findout: Finding outliers in very large datasets. *Knowledge and Information Systems*, 4(4):387–412, 2002.

[37] M. Ahmed and A. N. Mahmood. A novel approach for outlier detection and clustering improvement. In *IEEE 8th Conference on Industrial Electronics and Applications (ICIEA)*, pages 577–582, 2013.

[38] Ville Hautamäki, Svetlana Cherednichenko, Ismo Kärkkäinen, Tomi Kinnunen, and Pasi Fränti. Improving k-means by outlier removal. In *Proceedings of the 14th Scandinavian Conference on Image Analysis*, pages 978–987. Springer-Verlag, 2005.

[39] Iwan Syarif, Adam Prugel-Bennett, and Gary Wills. Unsupervised clustering approach for network anomaly detection. In *Networked Digital Technologies: 4th International Conference, Part I*, pages 135–145. Springer Berlin Heidelberg, 2012.

[40] Leonid Portnoy, Eleazar Eskin, and Salvatore Stolfo. Intrusion detection with unlabeled data using clustering. 2001.

[41] M. S. Parwez, D. B. Rawat, and M. Garuba. Big data analytics for user-activity analysis and user-anomaly detection in mobile wireless network. *IEEE Transactions on Industrial Informatics*, 13(4):2058–2065, 2017.

[42] Fabrizio Angiulli and Clara Pizzuti. Fast outlier detection in high dimensional spaces. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 15–26. Springer-Verlag, 2002.

[43] Y. Li and B. X. Fang. A lightweight online network anomaly detection scheme based on data mining methods. In *2007 IEEE International Conference on Network Protocols*, pages 340–341, 2007.

[44] Y. Li, T. B. Lu, L. Guo, Z. H. Tian, and L. Qi. Optimizing network anomaly detection scheme using instance selection mechanism. In *GLOBECOM 2009 - 2009 IEEE Global Telecommunications Conference*, pages 1–7, 2009.

[45] M. Xie, J. Hu, S. Han, and H. H. Chen. Scalable hypergrid k-nn-based online anomaly detection in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 24(8):1661–1670, 2013.

[46] K. Flanagan, E. Fallon, P. Connolly, and A. Awad. Network anomaly detection in time series using distance based outlier detection with cluster density analysis. In *2017 Internet Technologies and Applications (ITA)*, pages 116–121, 2017.

[47] G.E.P. Box, G.M. Jenkins, and G.C. Reinsel. *Time Series Analysis: Forecasting and Control*. Prentice Hall, 1994.

[48] M. Wang and S. Handurukande. A streaming data anomaly detection analytic engine for mobile network management. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 722–729, 2016.

[49] G. Casella and R.L. Berger. *Statistical Inference*. Brooks/Cole Publishing Company, 1990.

[50] A. Reddy, M. Ordway-West, M. Lee, M. Dugan, J. Whitney, R. Kahana, B. Ford, J. Muedsam, A. Henslee, and M. Rao. Using gaussian mixture models to detect outliers in seasonal univariate network traffic. In *2017 IEEE Security and Privacy Workshops (SPW)*, pages 229–234, 2017.

[51] Eleazar Eskin. Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 255–262. Morgan Kaufmann Publishers Inc., 2000.

[52] Kenji Yamanishi and Jun-ichi Takeuchi. Discovering outlier filtering rules from unlabeled data-combining a supervised learner with an unsupervised learner. pages 389–394, 2001.

[53] Kenji Yamanishi, Jun-ichi Takeuchi, Graham Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Mining and Knowledge Discovery*, 8(3):275–300, 2004.

[54] M. Xie, J. Hu, and B. Tian. Histogram-based online anomaly detection in hierarchical wireless sensor networks. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 751–759, 2012.

[55] Nong Ye and Qiang Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International*, 17(2):105–112, 2001.

[56] L.J. Kazmier. *Schaum's Outline of Theory and Problems of Business Statistics*. McGraw-Hill, 1996.

[57] Siwoon Son, Myeong-Seon Gil, and Y. S. Moon. Anomaly detection for big log data using a hadoop ecosystem. In *2017 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 377–380, 2017.

[58] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, and K.R.K. Murthy. Improvements to the smo algorithm for svm regression. *IEEE Transactions on Neural Networks*, 1999.

[59] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley-India, 2006.

[60] Apache Parquet. URL: `http://parquet.apache.org/documentation/latest/`.