Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2019

Diogo Daniel Soares Ferreira

Mecanismos para controlo e gestão de redes 5G: Redes de Operador

Control and management mechanisms in 5G networks: Operator networks

Universidade de Aveiro
Departamento de Eletrónica, Telecomunicações e Informática
2019

Diogo Daniel Soares Ferreira

Mecanismos para controlo e gestão de redes 5G: Redes de Operador

Control and management mechanisms in 5G networks: Operator networks

"*We can only see a short distance ahead, but we can see plenty there that needs to be done.*"

— Alan Turing

Universidade de Aveiro
2019

Departamento de Eletrónica,
Telecomunicações e Informática

Diogo Daniel Soares Ferreira

Mecanismos para controlo e gestão de redes 5G: Redes de Operador

Control and management mechanisms in 5G networks: Operator networks

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica da Professora Doutora Susana Sargento, Professora Catedrática do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Carlos Senna, Investigador do Instituto de Telecomunicações.

o júri / **the jury**

presidente / president

Professor Doutor Amaro Fernandes de Sousa
professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (por delegação do Vice-Reitor da Universidade de Aveiro)

vogais / examiners committee

Professor Doutor Rui Lopes Campos
professor auxiliar convidado da Faculdade de Engenharia da Universidade do Porto (arguente)

Professora Doutora Susana Sargento
professora catedrática do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientadora)

agradecimentos /
acknowledgements

Em primeiro lugar, agradeço à minha família, em especial aos meus pais e aos meus irmãos, não só pelo apoio total que me deram ao longo de toda a minha vida, mas também pela paciência que tiveram durante estes 5 anos de trabalho e estudo intenso.
Agradeço à Professora Doutora Susana Sargento e a todo o grupo de investigação Network Architectures and Protocols (NAP) o excelente acolhimento, apoio incondicional e todos os ensinamentos obtidos durante estes dois anos e meio de trabalho conjunto, onde sempre me foi dada a liberdade para desenvolver projetos de acordo com a minha vontade pessoal. Um agradecimento especial ao Doutor André Reis e ao Professor Doutor Paulo Salvador pelas discussões sobre Machine Learning, e também ao Doutor Carlos Senna pela ajuda na escrita desta dissertação.
Um grande agradecimento a todos os meus amigos que conheci durante este percurso, pelos momentos de descontração, pelos almoços, mas também pelas discussões interessantes sobre os mais variados temas, incluindo o trabalho apresentado nesta dissertação. Obrigado também a todos os professores que contribuíram para o meu crescimento académico.
Muito obrigado a todos os que direta e indiretamente contibuíram para a conclusão do meu mestrado.
Finalmente, um grande obrigado a Ele, sem o qual nada disto seria possível.
Agradeço à Fundação Portuguesa para a Ciência e Tecnologia pelo suporte financeiro com fundos nacionais e europeus através Fundo Europeu de Desenvolvimento Regional (FEDER), no âmbito do Programa Operacional Regional de Lisboa (POR LISBOA 2020) e do Programa Operacional de Competitividade e Internacionalização (COMPETE 2020) do Portugal 2020 (POCI-01-0247-FEDER-024539).

Palavras Chave

Previsão de séries temporais, 5G, Monitorização de rede, Aprendizagem Automática, Arquitetura de Previsão, Políticas de rede, Slicing de rede.

Resumo

Em redes 5G, séries temporais serão omnipresentes para a monitorização de métricas de rede. Com o aumento do número de dispositivos da Internet das Coisas (IoT) nos próximos anos, é esperado que o número de fluxos de séries temporais em tempo real cresça a um ritmo elevado. Para monitorizar esses fluxos, testar e correlacionar diferentes algoritmos e métricas simultaneamente e de maneira integrada, a previsão de séries temporais está a tornar-se essencial para a gestão preventiva bem sucedida da rede.

O objetivo desta dissertação é desenhar, implementar e testar um sistema de previsão numa rede de comunicações, que permite integrar várias redes diferentes, como por exemplo uma rede veicular e uma rede 4G de operador, para melhorar a fiabilidade e a qualidade de serviço (QoS). Para isso, a dissertação tem três objetivos principais: (1) a análise de diferentes datasets de rede e subsequente implementação de diferentes abordagens para previsão de métricas de rede, para testar diferentes técnicas; (2) o desenho e implementação de uma arquitetura distribuída de previsão de séries temporais em tempo real, para permitir ao operador de rede efetuar previsões sobre as métricas de rede; e finalmente, (3) o uso de modelos de previsão criados anteriormente e sua aplicação para melhorar o desempenho da rede utilizando políticas de gestão de recursos.

Os testes efetuados com dois datasets diferentes, endereçando os casos de uso de gestão de congestionamento e divisão de recursos numa rede com recursos limitados, mostram que o desempenho da rede pode ser melhorado com gestão preventiva da rede efetuada por um sistema em tempo real capaz de prever métricas de rede e atuar em conformidade na rede.

Também é efetuado um estudo sobre que métricas de rede podem causar reduzida acessibilidade em redes 4G, para o operador de rede atuar mais eficazmente e proativamente para evitar tais acontecimentos.

Keywords

Time-series prediction, Forecasting, 5G, Network Monitoring, Machine Learning, Prediction Architecture, Network Policies, Network Slicing.

Abstract

In 5G networks, time-series data will be omnipresent for the monitoring of network metrics. With the increase in the number of Internet of Things (IoT) devices in the next years, it is expected that the number of real-time time-series data streams increases at a fast pace. To be able to monitor those streams, test and correlate different algorithms and metrics simultaneously and in a seamless way, time-series forecasting is becoming essential for the pro-active successful management of the network.

The objective of this dissertation is to design, implement and test a prediction system in a communication network, that allows integrating various networks, such as a vehicular network and a 4G operator network, to improve the network reliability and Quality-of-Service (QoS). To do that, the dissertation has three main goals: (1) the analysis of different network datasets and implementation of different approaches to forecast network metrics, to test different techniques; (2) the design and implementation of a real-time distributed time-series forecasting architecture, to enable the network operator to make predictions about the network metrics; and lastly, (3) to use the forecasting models made previously and apply them to improve the network performance using resource management policies.

The tests done with two different datasets, addressing the use cases of congestion management and resource splitting in a network with a limited number of resources, show that the network performance can be improved with pro-active management made by a real-time system able to predict the network metrics and act on the network accordingly.

It is also done a study about what network metrics can cause reduced accessibility in 4G networks, for the network operator to act more efficiently and pro-actively to avoid such events.

Contents

List of Figures

List of Tables

Acronyms

5G PPP	5G Infrastructure Public Public Private Partnership
5G	Fifth Generation of Cellular Mobile Communications
5G NORMA	5G Novel Radio Multiservice adaptative network Architecture
AAI	Active and Available Inventory
Adam	Adaptative Moment Estimation
ANN	Artificial Neural Network
API	Application Programming Interface
AR	Autoregressive
ARMA	Autoregressive moving average
ARIMA	Autoregressive integrated moving average
BoF	Bag of Flows
BML	Business Management Layer
CapEx	Capital Expenditure
CCTA	Central Computer and Telecommunications Agency
CLAMP	Closed Loop Automation Management Platform
CMIP	Common Management Information Protocol
CMIS	Common Management Information Service
COMPA	Control, Orchestration, Management, Policy and Analytics
COPS	Common Open Policy Service
CSE	Cognitive Smart Engine
CSFB	Circuit Switched Fallback
DCAE	Data Collection, Analytics and Events
DDoS	Distributed Denial of Service
DPI	Deep Packet Inspection
DRMSET	Differentiated Root Mean Squared Error above Threshold
ECN	Echo State Network
ECOMP	Enhanced Control, Orchestration, Management & Policy
EM	Expectation Maximization
EML	Element Management Layer
eNB	E-UTRAN Node B
E-RAB	E-UTRAN Radio Access Bearer
eTOM	Business Process Framework
ETSI	European Telecommunications Standards Institute
FAB	Fulfillment, Assurance and Billing
FCAPS	Fault, Configuration, Accounting, Performance and Security
GRU	Gated Recurrent Unit
HTTP	Hypertext Transfer Protocol
IaaS	Internet as a Service
IN	Intelligent network
IPFIX	IP Information Flow Export
IETF	Internet Engineering Task Force
IoT	Internet of Things
IRNN	Identity Recurrent Unit
ISO	International Organization for Standardization
ISP	Internet Service Provider
ITIL	Information Technology Infrastructure Library
ITU-T	International Telecomunication Union Telecommunication Standardization Sector
KPI	Key Performance Indicator
LAN	Local Area Network
LSTM	Long Short-Term Memory Neural Network
NML	Network Management Layer
MA	Moving Average
MAPE	Mean Absolute Percentage Error
MIB	Management Information Base
ML	Machine Learning
MME	Mobility Management Entity
MMS	Multimedia Messaging Service
MOEFC	Multi-Objective Evolutionary Fuzzy Classifier
MSO	Master Service Orchestrator
NaaS	Network as a Service
NEL	Network Element Layer
NF	Network Function

NFV	Network Functions Virtualization
NMS	Network Management System
OAM&P	Operation, Admnistration, Maintenance and Provisioning
OOF	ONAP Optimization Framework
ONAP	Open Network Automation Platform
OSI	Open Systems Interconnection
OPEN-O	Open Orchestrator Project
OpEx	Operational Expenditure
P2P	Peer-to-Peer
PaaS	Platform as a Service
PCA	Principal Component Analysis
PDCP	Packet Data Convergence Protocol
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PNF	Physical Network Function
PPDIOO	Prepare, Plan, Design, Implement, Operate and Optimize
PSAMP	Packet Sampling
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RBF	Radial basis function
ReLU	Rectified Linear Unit
REST	Representational State Transfer
RRC	Radio Resource Control
RTC	Robust statistical Traffic Classification
RNN	Recurrent Neural Network
RMON	Remote Monitoring
RMSE	Root Mean Squared Error
RMSET	Root Mean Squared Error above Threshold
SDC	Service Design & Creation
SDN	Software-Defined Network
SELFNET	Framework for self-organized network management in virtualized and software-defined networks
SGMP	Secure Gateway Management Protocol
SLA	Service-Level Agreement
SLO	Service-Level Objective
SML	Service Management Layer
SNMP	Simple Network Management Protocol
SMS	Short Message Service
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TINA-C	Telecommunication Information Networking Architecture Consorcium
tmForum	TeleManagement Forum
TMN	Telecommunications Management Network
UE	User Equipment
UDP	User Datagram Protocol
VIM	Virtual Infrastructure Manager
VNF	Virtual Network Function

1

Introduction

This chapter describes the context and the motivation that supported this work and the writing of this dissertation. The concept of network management is explained, as well as the usefulness of management mechanisms for 5G networks. The general objectives and the main contributions of this work are presented, as well as an explanation of the document structure.

1.1 Context and motivation

Due to the complex and large-scale services provided by today's networks, it is not trivial to define the current scope of network management and its capabilities. In 1996, Saydam and Magedanz [1] said that "Network management includes the deployment, integration and coordination of the hardware, software and human elements to monitor, test, poll, configure, analyze, evaluate and control the network and element resources to meet the real-time, operational performance and Quality of Service requirements at a reasonable cost".

In 2009, Boutada and Jin [2] took a more functional approach to define network management: "A network management system is an application. Its aim is to analyze, control and manage network and service infrastructure in order to ensure its configuration correctness, robustness, performance quality and security". This view updates the previous one by considering network management as an application with the aim of managing a service infrastructure, excluding external factors like human elements or the cost of the network.

Management mechanisms are fundamental for the effective operation of a network. There are various KPIs of extreme importance to monitor in a network, such as latency, jitter, packet loss or availability. Recently, newer and advanced KPIs were proposed [3] to be able to monitor the networks with the increased requirements of the new generations of cellular mobile communications (e.g. 1 000 000 devices per km^2, 20 Gbit/s of download peak data rate). With more complex KPIs, there is a need to design and implement management mechanisms capable of dealing with them and to act accordingly in the network. The ultimate goal of the management mechanisms is to keep track of the network state (with the KPIs) and to optimize the network resources, even in abnormal situations.

In recent years, mobile network traffic has seen an exponential increase, surpassing fixed network traffic. Mobile broadband subscriptions have been following this trend[1], driven by the growing use of mobile computing devices to access the internet and to consume network demanding services, such as online gaming or high-resolution video.

[1] `https://www.ericsson.com/en/mobility-report/reports/june-2019`

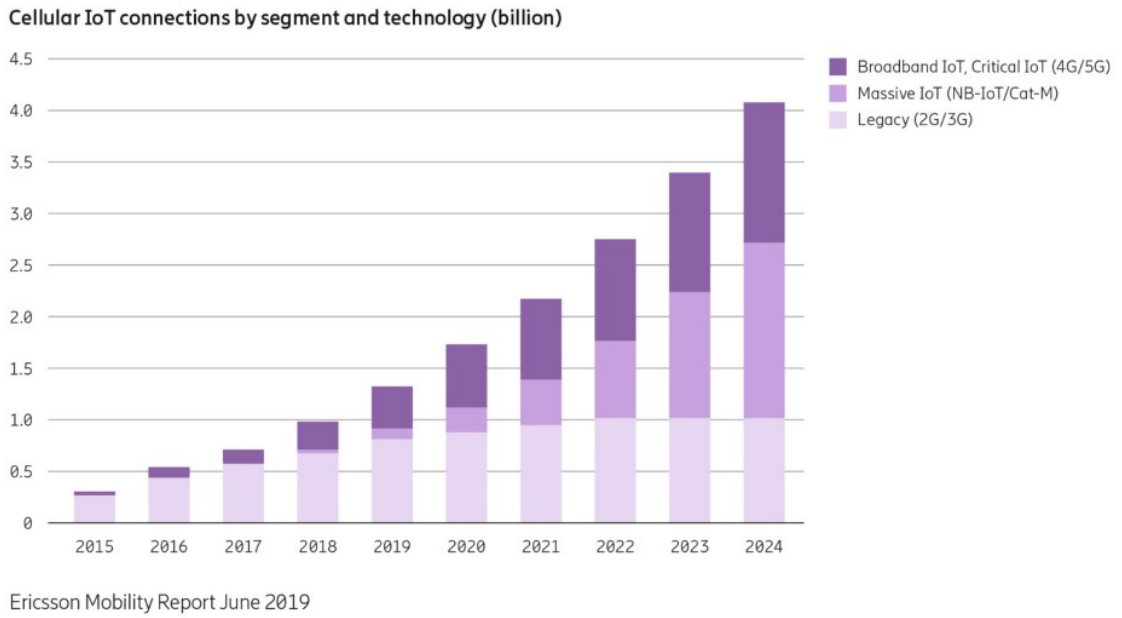

Figure 1.1: Ericsson projection for the number of cellular IoT connections in the next years[1].

With the exponential increase of network traffic around the globe, new requirements for the mobile communication architecture started to emerge. The predicted Internet of Things (IoT) boom (Figure 1.1) and its application in several industries, such as automotive, agriculture, energy or health care, create new challenges that current mobile communication architectures have trouble in answering. The effort made in recent years by the research community and the telecom industry in defining a new network architecture (Fifth Generation of Cellular Mobile Communications (5G)) that supports the new set of requirements is finally reaching the market.

The urge for efficient monitoring of those networks is decisive for their successful management. Due to its dynamic load and flexible topology, prediction of the network state is a must to assure that the user requirements are met. Automation of the network management is also mandatory, only possible due to the advances in virtualization, mainly in Software-Defined Network (SDN) and NFV. There is some early work that shows the advantages of introducing automation in network management [4], [5].

1.2 Objectives

This dissertation will contribute to the accurate monitoring of network metrics. The current operator network predictors implement linear statistical models invented in the 1980s. However, recent studies show that with the advance of Deep Learning techniques, newer approaches have higher accuracy than the ones currently used in network monitoring.

For the network metric predictors to be used, they must be integrated with the management frameworks used by the network operators. For an easier integration without any capability loss, it will also be designed and implemented a prediction architecture with a uniform interface, to ease the installation of various network predictors with different management frameworks.

Two use cases of policy implementation with the use of network predictions will be tested, to measure the impact of accurate predictions on the management of a network.

Finally, for better pro-active management of the network, it will be done a study about what network KPIs are the most important when forecasting reduced network accessibility in a 4G operator

network, to understand what are its causes and how to mitigate them.

In summary, the goal of the dissertation is fourfold:

- to implement different forecasting approaches and to build models to forecast the network KPIs in different datasets and with different goals;
- to create a distributed real-time prediction architecture to allow the network operators to perform predictions in real-time using a uniform interface, with increased modularity and enabling horizontal scaling for prediction of different network KPIs; it also allows to add or remove prediction modules on-the-fly, as well as to perform online training with recent network behavior;
- to implement dynamic network policies to improve the reliability, stability and quality of service of the network based on the predicted KPIs in the network;
- to understand what network KPIs cause reduced network accessibility in a 4G network.

1.3 Contributions

The work developed in this dissertation led to the following contributions:

- analysis and implementation of two network prediction models using statistical and machine learning techniques; the analysis includes exploration of the datasets with time-series techniques and tests varying the hyper-parameters and external features;
- implementation and tests of two network prediction models where the goal is adapted to specific use cases: predicting higher value observations and predicting on anomalous time intervals;
- design and implementation of a generic prediction architecture able to contain various predictors for different metrics with a uniform API, enabling horizontal scaling;
- implementation of dynamic policies for two use cases in a network using the forecasts made, to improve the network behavior;
- analysis of the root cause KPIs of low network accessibility in a 4G network.

A journal about the network prediction approaches used and the tests done will be submitted to "Advances in Artificial Intelligence and Machine Learning for Networking" (IEEE Journal on Selected Areas in Communications (J-SAC))[2]. A conference paper about the real-time distributed time-series prediction architecture and its implementation will be submitted to an international conference, as well as a conference paper about the root cause KPIs of low network accessibility.

1.4 Document structure

This work is structured as follows:

- **Chapter 2 - Background and Related Work** - it presents the most interesting network management frameworks and network management protocols in network telecommunications; it also presents the 5G networks and its main characteristics (concepts, architecture, slicing and management frameworks); finally, it explains what is machine learning, what is its purpose and the internals of the forecasting techniques used in this dissertation;
- **Chapter 3 - State of the art** - it presents the state of the art in the application of machine learning for the management of networks, with a special focus on network forecasting;

[2] https://www.comsoc.org/publications/journals/ieee-jsac/cfp/advances-artificial-intelligence-and-machine-learning

- **Chapter 4 - Forecasting of the number of sessions in a vehicular network** - it explores the dataset with the number of sessions in the network of the public buses in Porto, and tests different approaches to achieve the best model that forecasts the number of sessions in the next hour;
- **Chapter 5 - Forecasting of Specific Characteristics in Network Metrics** - using the same dataset as in the previous chapter, two different forecasting scenarios are presented in this chapter: to forecast accurately higher value observations, allowing for larger errors in smaller value observations, and to forecast accurately on anomalous time intervals;
- **Chapter 6 - Forecasting the number of connected users in a 4G network** - using a dataset from a Portugal Internet Service Provider (ISP) with various network KPIs, it tests different approaches to forecast accurately the number of connected users in the network in the next hour;
- **Chapter 7 - Root Cause Analysis of Low Network Accessibility** - using the same dataset as in the previous chapter, the goal of this chapter is not only to forecast the network accessibility, but also to understand what network KPIs better indicate a potential future decrease of network accessibility;
- **Chapter 8 - Real-time Distributed Time-Series Prediction Architecture** - it explains the distributed real-time prediction architecture and its implementation, along with each module requirements;
- **Chapter 9 - 5G Network Slicing with Network Metrics Forecasts** - using some of the forecasts made in the previous chapters, the use cases of congestion management and resource splitting in a network with a limited number of resources will be addressed to create dynamic network policies to improve the network performance;
- **Chapter 10 - Conclusion & Future Work** - presentation of the final conclusions of the dissertation work and suggestions for future improvements related to the presented work.

2

Background and Related Work

This chapter presents the background and related work in four fields of study. First, it presents network management frameworks. A network management framework is a model that represents the interactions between different functional abstract and conceptual blocks with the goal of managing a network. In this dissertation, it is proposed to develop a generic block that can be integrated with most network management frameworks. For that, it is crucial to understand the needs and the rationale behind the most influential network management frameworks.

Second, this chapter presents the best known network management protocols. The network management protocols implement a set of tools for collecting information about the network and its devices and for modifying the devices' configurations. They are relevant for this work to understand how the data collection is made, what type of data is collected and how to act on the network devices.

The third field is 5G networks, mainly focusing on SDNs, NFVs, network slicing and management frameworks. To understand how to contribute to the implementation of a component for a network management system, it is needed first to understand the next generation networks, its characteristics and its requirements. Analyzing the differences between the 5G networks and the previous generation networks is essential not only to figure out what KPIs are important to predict in a 5G network, but also to understand how to design a prediction system compatible with the needs of 5G networks.

Finally, the last section focuses on machine learning. Being fundamental for the autonomous management of a network, machine learning techniques have greatly evolved in the past years, particularly in the Deep Learning subfield. The most accurate network forecasting models take advantage of Deep Learning to build a model of the network behavior.

2.1 Network Management Frameworks

Operation, Admnistration, Maintenance and Provisioning (OAM&P)

Operation, Admnistration, Maintenance and Provisioning is a telecommunication network management framework widely adopted by large service providers in their network management systems in the early 1980s and 1990s. It has its origins in the wired telephony world. As the name suggests, it is divided into four main categories [6]:

- **Operation** - activities that are taken to keep the network and its services to run as supposed to, mainly monitoring the network and suppressing potential problems;
- **Administration** - support procedures necessary for day-to-day operations. It involves discovering and monitoring the network resources;

- **Maintenance** - this function is focused on solving network issues that affect service or network operation. It also involves corrective and preventive measures to make the managed network to run more effectively;
- **Provisioning** - configure the network to provide new services, setting up new equipment or installing new hardware.

FCAPS

FCAPS was introduced in the early 1980s and it was standardized by International Organization for Standardization (ISO) [7]. Today, it still is the most popular conceptual framework for network management, mainly because it can be applied effectively to almost all networks. The FCAPS model divides network management into five functional areas:

- **Fault management** - enforces the detection, isolation and correction of abnormal situations in the telecommunication network;
- **Configuration management** - network operation is monitored, and the addition, modification and removal of network elements are coordinated;
- **Accounting management** - provides the metrics of network services needed to determine the costs to the service provider and the charges for each customer;
- **Performance management** - manages the overall performance of the network, maximizing the throughput and minimizing possible bottlenecks;
- **Security** - prevention and detection of security breaches, as well as to take recovery methods when a security breach happens. This area is common to all the other functional areas.

Telecommunications Management Network (TMN)

The TMN protocol model is a set of standards defined by International Telecomunication Union Telecommunication Standardization Sector (ITU-T) [7] with the goal of supporting the management of a telecommunications network. It creates a hierarchical structure that allows the interconnection of various operating systems and telecommunication equipment, using a well-defined architecture with normalized protocols and interfaces. In spite of the commercial importance of TMN being decreasing, being replaced by eTOM (as explained further ahead), this management model is a reference framework for telecommunication management related topics.

The network management tasks are grouped into management layers, although in practice the software implementing the TMN model may not clearly separate these layers:

- **Network Element Layer (NEL)** - this layer defines interfaces for network elements, instantiating functions for device instrumentation;
- **Element Management Layer (EML)** - this layer provides management functions for network elements on an individual or group level. It also deals with abstracting the functions provided by the Network Element layer;
- **Network Management Layer (NML)** - this layer's main tasks are the configuration, control and supervision of the overall network, having to coordinate different multi-vendor network elements. It also deals with network fault detection and optimization;
- **Service Management Layer (SML)** - this layer is responsible for the contractual services provided to users, for example Quality of Service (QoS) management or Service-Level Agreement (SLA) availability;
- **Business Management Layer (BML)** - this layer is responsible for enterprise metrics. Includes the functions related to business trends and to provide financial reports.

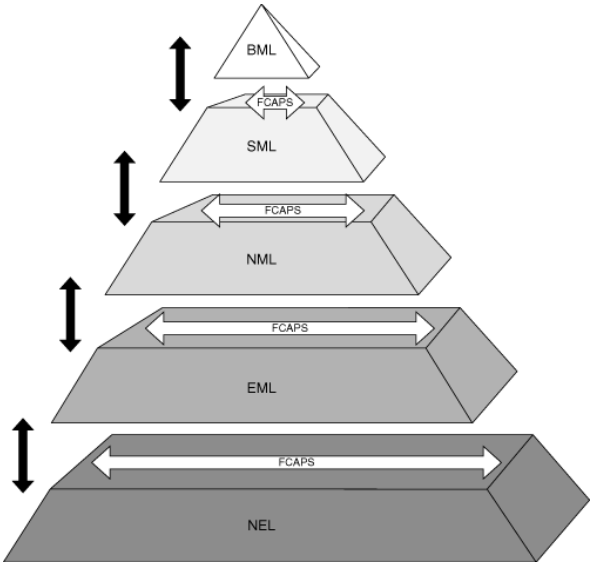

Figure 2.1: Illustration of the FCAPS model[1]

As shown in Figure 2.1, the FCAPS logical layers are compatible with all layers of the TMN model. While TMN presents its reference model with management layers, FCAPS has its architecture organized into functional areas.

Telecommunication Information Networking Architecture Consorcium (TINA-C)

TINA-C was formed in 1992 with the goal of defining, designing and implementing a software architecture for the telecommunications infrastructure. TINA-C made an effort to achieve a uniformization between Intelligent networks (INs) and the TMN model, with a conceptual model that captured essential aspects of telecommunication operations, like resources, networks, software, services and participants [2]. Due to the lack of adoption of the architecture in the market, TINA-C is inactive since 2000.

eTOM

In 2001, eTOM was released (initially only Telecommunication Operation Map(TOM)) as an operating model framework for the telecom service providers in the telecommunications industry, with the goal of replacing OAM&P and complementing TMN [8]. It is currently a standard maintained by TeleManagement Forum (tmForum)[2]. The main difference between TMN and eTOM is that while the TMN approach was built based on the assumption that it would be applied to large network operations, eTOM was built based on the need to support processes of the entire service provider enterprise.

eTOM focuses on modeling enterprise processes according to their priorities for the business. It is technology and service independent, which means that it can be used in other contexts besides telecommunications service providers. It defines three main blocks (Figure 2.2):

- **Enterprise Management** - contains the high-level business processes, with functions like marketing and financial management;
- **Strategy, Infrastructure and Product** - contains the processes needed for the correct management of product lifecycle;

[1] `http://etutorials.org/Networking/network+management/Part+I+Data+Collection+and+Methodology+Standards/Chapter+3.+Accounting+and+Performance+Standards+and+Definitions/Architectural+and+Framework+Standards+The+TMN+FCAPS+Model+ITU-T/`

[2] `https://www.tmforum.org/business-process-framework/`

[3] `https://en.wikipedia.org/wiki/Business_Process_Framework_(eTOM)`

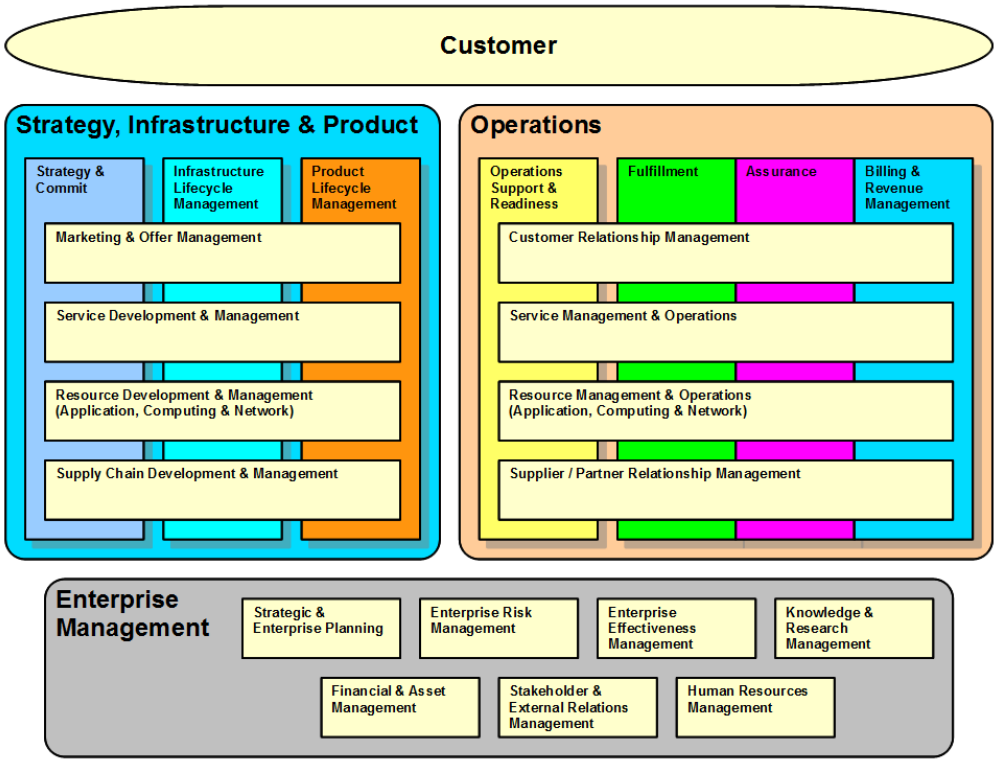

Figure 2.2: Illustration of the eTOM process model[3]

- **Operations** - includes the processes that support customers, network operations and management. The core of operations is the Fulfillment, Assurance and Billing (FAB) model: fulfillment is concerned with delivering products to the customer; assurance has the objectives of system monitoring, performance monitoring and general maintenance; billing determines the charges to the customers, according to the services consumed.

Information Technology Infrastructure Library (ITIL)

ITIL was developed in the late 1980s by the Central Computer and Telecommunications Agency (CCTA) and provides a framework of best practice guidance for IT service management. It has gone through several revisions in its history, and while in the beginning there were more than 30 books, currently ITILv3 (2007) comprises five books [8] [9]. The original goal was not to create a product, neither a standard: it was to create a comprehensive, publicly available guidance for IT service management, mainly because of the lack of standard procedures that were increasing costs and allowing errors to perpetuate[4].

The five volumes defined in ITILv3 are[5]:

- **Service strategy** - the center and origin point of the ITIL service lifecycle. It focuses on the approach to turn IT service management into a strategic asset. Includes defining the services that will be offered, to whom, or how the company will measure the quality of the service;
- **Service design** - is concerned with creating new services to serve the customers, including their architecture, processes, policies and documentation, having as basis the service strategy;

[4] `https://www.cio.com/article/2439501/infrastructure-it-infrastructure-library-itil-definition-and-solutions.html`

[5] `https://www.howtonetwork.org/tshoot/module-1/an-overview-of-network-management-models/`

- **Service transition** - focuses on the new proposed services and delivers them into operational use;
- **Service operation** - manages the current services offered to the customers;
- **Continual Service Improvement** - iterational and incremental large-scale improvements to the current services, ensuring his quality.

Cisco Prepare, Plan, Design, Implement, Operate and Optimize (PPDIOO)

The Cisco PPDIOO is a proprietary network life cycle, consisting of six steps that guide the network administrators to network design, implementation and update to the creation of a highly efficient network infrastructure[6]. The approach is in the form of a continuous loop, with the steps being[7]:

- **Prepare** - establish the business requirements, develop a network strategy and propose a high-level conceptual architecture identifying best-suited technologies;
- **Plan** - audit the exiting network and validate that it can support the proposed system;
- **Design** - based on the business requirements already defined in the prepare phase and the technical information defined in the planning phase, begin to design the new network topology;
- **Implement** - configure and setup the new equipment specified in the design phase, integrating services without disrupting the existing network or creating points of vulnerability;
- **Operate** - support and monitor the network implemented in the previous phase;
- **Optimize** - with the monitoring done to the network, identify possible changes to improve performance and resolve issues.

2.2 Network Management Protocols

Secure Gateway Management Protocol (SGMP)

SGMP was defined in 1987 [10] and is a simple protocol that can retrieve and assign values of variables. It can monitor network interfaces, interface status, routing protocols and other network-related information. It was quickly replaced by SNMP.

SNMP

SNMP is used for collecting information from network devices and configure them. Developed in 1988 and approved as an Internet standard in 1990 [11], it is a simple protocol for network management that replaced SGMP. In the SNMP architecture, there are three types of entities: managed devices, agents and Network Management Systems (NMSs). A managed device is a network node, such as a switch, router or host, that contains an SNMP agent. The agent is a network management software module that resides in a device. The NMSs execute the applications that interact with the managed devices to monitor them. These entities and their interactions are depicted in Figure 2.3.

The information served by SNMP agents is described by a Management Information Base (MIB). There are also two traditional ways to monitor the network. The NMS can periodically ask the agent for new information (polling), or the agent can send a message only when an event occurs (trap).

SNMP has been widely deployed in Local Area Networks (LANs), mainly because it is simple to implement, it is an open standard and the pooling method is adequate to LANs. However, it has some drawbacks: it hardly scales to large ISP networks, there can be a large overhead due to polling, and its specific semantics makes its integration with other approaches difficult.

[6] https://www.cisco.com/web/partners/services/promos/accelerate/downloads/lifecycle_services_sg.pdf
[7] https://ccie-or-null.net/2011/05/09/the-cisco-ppdioo-life-cycle/

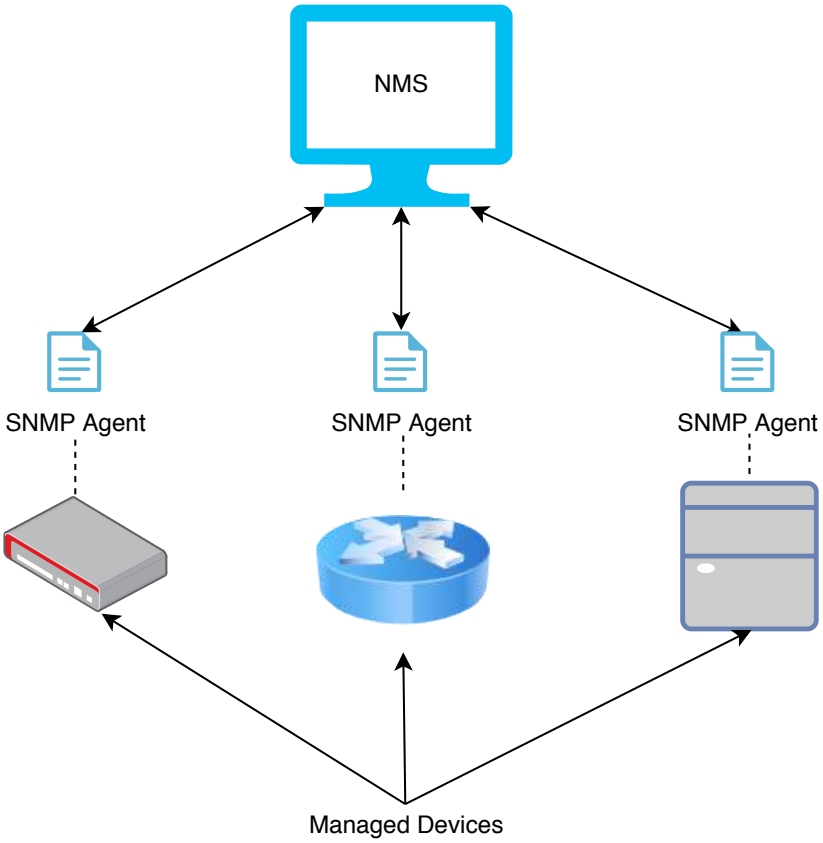

Figure 2.3: Illustration of the SNMP entities.

Common Management Information Service (CMIS)/Common Management Information Protocol (CMIP)

Common Management Information Service is a service interface specified by ITU-T used by Open Systems Interconnection (OSI) network elements for network management. It was designed in competition with SNMP. The protocol CMIP [12] provides an implementation for CMIS. CMIP was designed in the 1980s to be the unifying protocol in large networks. It is a key part of the TMN framework, enabling cross-organizational and cross-vendor network management. However, its management information model with an object-oriented approach based on managed objects, its heavy specification and the complex resource requirements of CMIP agents and management systems lead to most network elements supporting SNMP and not CMIP, making SNMP dominate the market. Today, CMIP still has some relevance in the telecommunication market.

Common Open Policy Service (COPS)

The Common Open Policy Service protocol was defined in RFC 2748 [13]. It enforces a policy-based model, in which the goal is to globally manage the network and not its elements. It is possible to define policies to inform the network of what to do in terms of service access, differentiation and charging. Those policy rules are then translated into equipment configuration changes.

In a policy management system, there are four main entities: the policy management stations, the policy repository, the policy consumers (or Policy Decision Point (PDP)) and the policy targets (or Policy Enforcement Point (PEP)). The policy management stations control and send policy rules to

the policy repository. The policy repository mantains an updated list of all the policies sent by the policy management stations. The PDPs retrieve the policies from the policy repository and translate them into hardware-specific rules to be sent to PEPs. SDN can be seen as a variation of this concept.

The COPS protocol is a question/answer protocol to assure the communication between the PDPs and PEPs. There are two types of models:

- **RSVP (outsourcing model)** - PEPs contacts PDPs when a decision is needed;
- **DiffServ (configuration model)** - PDPs configure PEPs with specific equipment information.

Remote Monitoring (RMON)

RMON is a set of standardized MIB variables, developed by Internet Engineering Task Force (IETF) [14], to monitor networks. All previously defined MIBs monitored only nodes. Typically, an RMON implementation operates in a client/server model. Monitoring devices are called probes and they contain software agents that collect information and analyze packets. The main difference between RMON and SNMP is that, while SNMP is used for individual network element management, RMON focuses on flow-based monitoring.

RMON is divided into nine groups: Statistics, History, Alarm, Host, HostTopN, Matrix, Filter, Packet Capture and Event.

NetFlow

NetFlow is an industry-standard protocol introduced in Cisco routers in 1996 [15] that provides network administrators IP flow information from their data networks. A flow is defined as a unidirectional sequence of packets with some common properties between two devices in the network. The NetFlow monitoring setup consists of three main components:

- **Flow Exporter** - a network element, such as router or switch, that aggregates packets as they enter or exit an interface, and exports flow records to flow collectors;
- **Flow Collector** - a computing device that receives the flow records from the flow exporters, stores them and pre-processes the flow data;
- **Analysis application** - application that queries the flow collector for flow data records.

IP Information Flow Export (IPFIX)

IPFIX is an IETF protocol [16] created with the need for standardizing an export protocol similar to NetFlow. It is based on Netflow v9, and many functionalities are common to both systems.

Packet Sampling (PSAMP)

The PSAMP working group started after the IPFIX group creation [17]. In high-end routers, where monitoring every packet is impossible, the monitoring must be done by sampling subsets of packets with statistical methods. The goal of PSAMP is to define ways of configuring packet selection, sampling and exporting processes. The information is exported by packet, ignoring the notion of flow. For the exportation of PSAMP packet information, it is used the IPFIX protocol.

2.3 5G networks

Since 2012 there is active research on defining the next generation of cellular mobile communications (5G). The standard was finalized and presented in 2018 [18], and commercial deployments are expected by early 2020. The IoT explosion, the growing number of users and its high demands, along with the new wave of software virtualization (SDNs and NFVs), have the potential of disrupting the mobile telecommunications world, making it more flexible and capable of satisfying the users' needs.

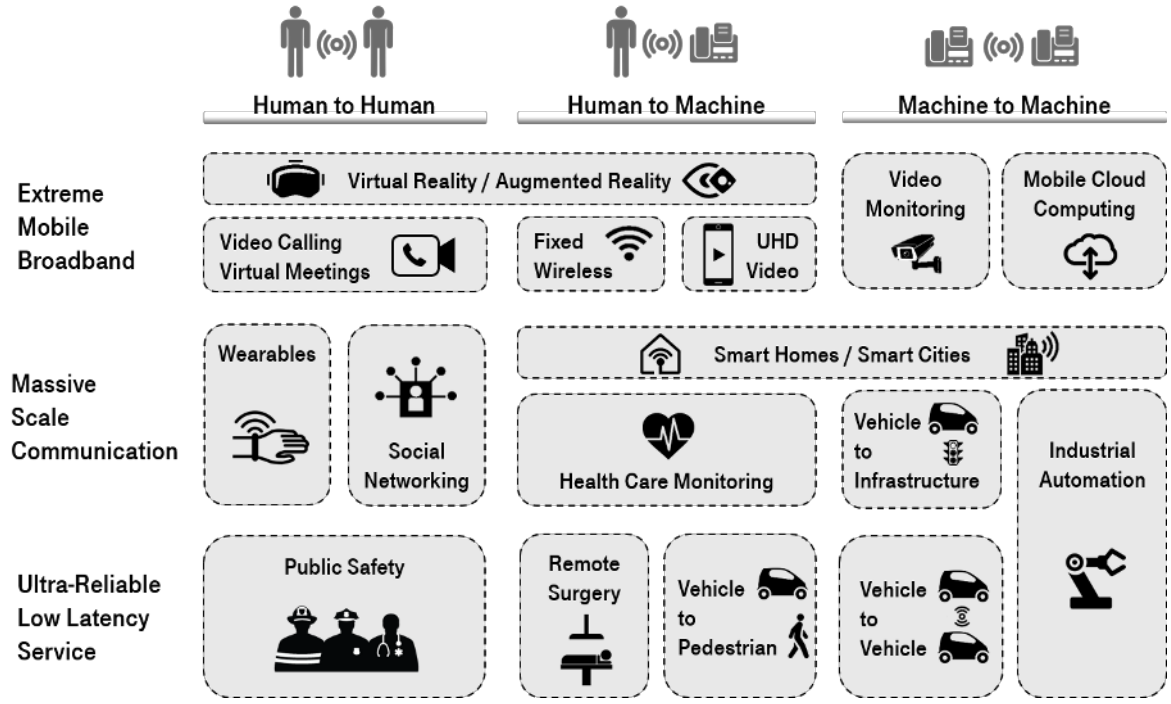

Figure 2.4: Some 5G use cases grouped by the type of interaction and the range of performance requirements [20].

In 3GPP [19], over seventy use cases for 5G were identified, which were divided into three categories (Figure 2.4) [20]:

- **Extreme Mobile Broadband** - this category includes use cases such as high-resolution video streaming, virtual reality or video monitoring. It is designed for real-time data responsiveness and aims at the growth of mobile devices connected to the internet that consume high-speed data. It is characterized by high-speed data access, higher coverage capacity and higher user mobility[8];
- **Massive Scale Communication** - this category includes use cases such as smart homes, vehicle-to-infrastructure platftorms or health care monitoring. It is driven by the exponential growth of Internet of Things devices and as such, its main characteristics are the high number of devices connected in a small area. These devices are usually low-cost devices with long battery life and that send data sporadically;
- **Ultra-Reliable Low Latency Service** - this category includes use cases such as vehicle-to-vehicle communications, remote surgery or public safety platforms. It is designed for safety-critical missions, being characterized by low latency communications, high availability and high reliability.

Malandrino and Chiasserini [21] showed the benefits of existing mobile applications having their demand served on a 5G network, arguing that high-data rate and low-sparseness apps have the most to gain from 5G integration.

To achieve a flexible and dynamic environment in the network, SDN and NFV will play a major role in the 5G architecture deployment and management.

2.3.1 Software-Defined Network

The advances in cloud computing and the need to reduce Capital Expenditure (CapEx) and Operational Expenditure (OpEx) [22] led to a new technology that enables network softwarization as

[8] `https://5g.co.uk/guides/what-is-enhanced-mobile-broadband-embb/`

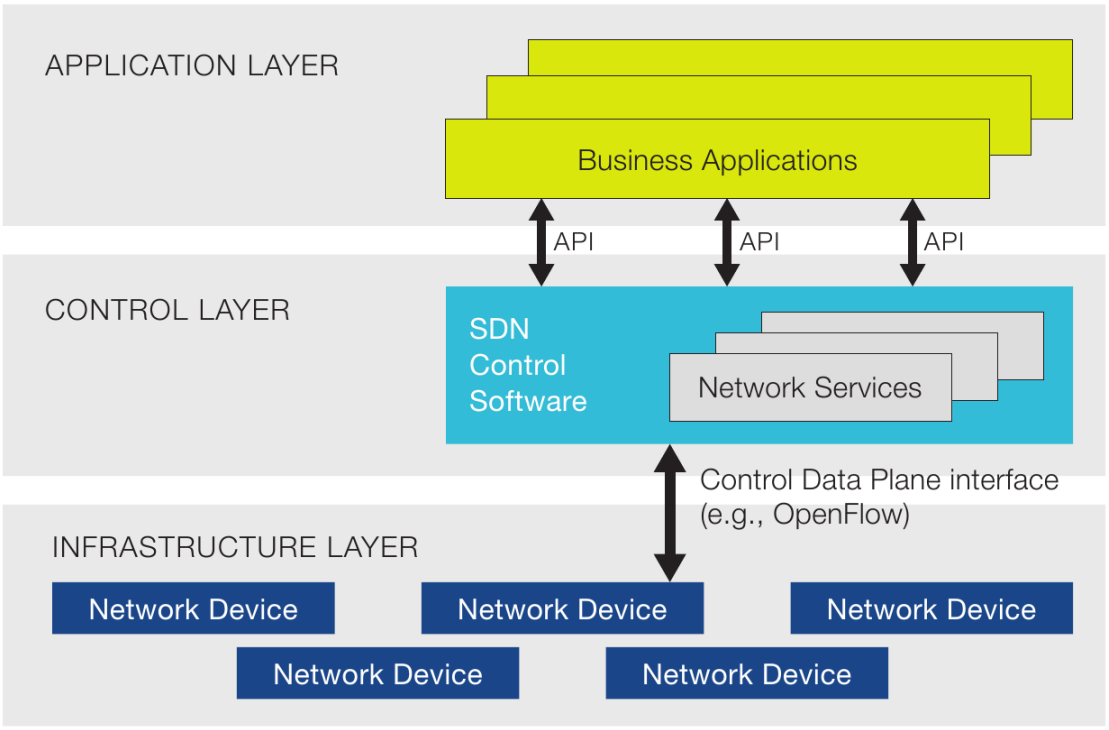

Figure 2.5: Software-Defined Network architecture [28].

an emerging trend, where its objective is to decouple the hardware from the software implementation.

The SDN concept has been discussed over twenty years. In 1996, D. Tennenhouse and D. Wetherall presented an active network architecture [23] [24]: a highly programmable network, where its modules are programmable (e.g. switches), and it is possible to inject customized programs into the nodes of the network. In 1998, the Tempest framework [25] was the first network programmable framework to explore the concept of a separated control plane (to decide how to handle the traffic) from the data plane (to forward network traffic according to the instruction received from the control plane). In 2008, OpenFlow was proposed as a communication protocol to control the forwarding plane of a network switch or router [26]. OpenFlow is currently the standard communication interface between the control and forwarding layers of an SDN architecture, defined by the Open Networking Foundation [27].

The SDN architecture [28] is separated into three layers (Figure 2.5):

- **Infrastructure layer** - contains the hardware components that are being managed by the SDN controllers;
- **Control layer** - contains one or more SDN controllers. The SDN controller is a software that serves as the middleware between the physical network and the application layer, abstracting it from all the hardware details of the network. The interface between the control layer and the infrastructure layer is made using the Openflow protocol;
- **Application layer** - contains the programs that interact with the network with a view defined by an API in the control layer, by the SDN controller.

Gelberger, Yemini, and Giladi [29] showed that, in spite of the flexibility achieved with the SDN architecture, there is a penalty in the raw performance of the network (the analyzed metrics were throughput and latency), although that penalty does not depend on the complexity of the SDN infrastructure.

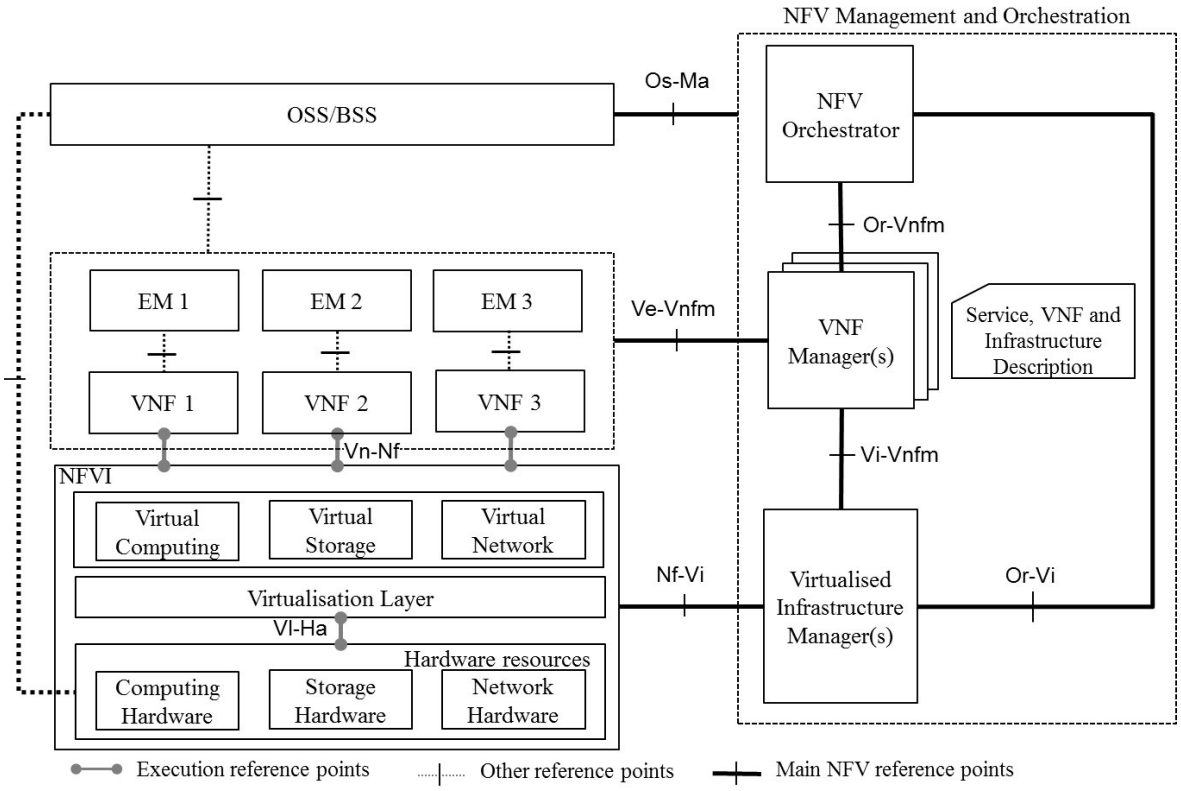

Figure 2.6: NFV reference architectural framework [32].

2.3.2 Network Functions Virtualization

Another highly complementary technology to software-defined functions is network function virtualization. The aim of NFV is to implement Virtual Network Functions (VNFs) that can be instantiated in generic servers or cloud infrastructure without the need for specialized hardware, decoupling the software from the hardware appliances. Some benefits of enhancing the network with NFV are reduced CapEx and OpEx, lower Time to Market and encouraging open vendors to release its products [30].

Network functions that can be virtualized include [31]:

- packet core functions such as the mobility management entity, serving gateway, and packet data network gateway;
- baseband processing unit functions, including medium access control (MAC), radio link control (RLC), and radio resource control (RRC) procedures;
- switching function;
- traffic load balancing;
- operation service centers.

The NFV reference architecture was standardized by the European Telecommunications Standards Institute (ETSI) in [32] (Figure 2.6). It is described at a function level, not proposing any specific implementations.

The NFV architecture has three major components. The NFV Infrastructure (NFVI) contains the hardware (computing, storage and network) resources, that are virtualized with a common interface for the VNFs. The VNFs run on the NFVI and they are available to be accessed by the OSS/BSS. The Element Management System (EMS) performs the typical management functionality for one or several VNFs. The NFV management and orchestration comprises resource provisioning modules to ensure the VNF operation.

SDN and NFV are not dependent on each other, but the combined use of them is beneficial for each other. Ordonez-Lucena, Ameigeiras, Lopez, *et al.* [33] present a scenario that combines SDN and NFV to monitor and analyze the behavior of the network infrastructure.

2.3.3 Network Slicing

In previous cellular mobile generations up to 4G, the different services (Short Message Service (SMS), Multimedia Messaging Service (MMS), internet, voice chat, etc.) were hosted on the same mobile network architecture. For the new variety of use cases and its demanding requirements, it would be complex and expensive to host them on the same mobile architecture without any division between them [34]. The solution is network slicing: configuration of logical end-to-end slices for network traffic with the use of virtualization (SDN and NFV), to meet a certain set of requirements for each slice. Each slice can be treated logically as a separate network. The formal definition of network slicing is given by 3GPP [18], "The network slice is a composition of adequately configured network functions, network applications, and the underlying cloud infrastructure (physical, virtual or even emulated resources, Radio Access Network (RAN) resources etc.), that are bundled together to meet the requirements of a specific use case, e.g., bandwidth, latency, processing, and resiliency, coupled with a business purpose".

To better understand the value of network slicing, specific use cases can be analyzed. In a remote surgery scenario, it is demanded low latency communications and high availability. On the other hand, if in the same network there are users playing a virtual reality game that consumes high-speed data from the internet, the traffic from the virtual reality game can cause congestion on the network that delays the remote surgery communications. If each scenario had its own slice of the network, no slice can interfere with another, so that congestion would only be noticed on the slice of the virtual reality game. In Figure 2.7, it is presented a representation of three logical network slices, each one of them for a specific use case with unique requirements, running on top of the same physical infrastructure.

In a joint study from Ericsson and BT[9], it was shown that with the introduction of network slicing

[9] https://www.ericsson.com/assets/local/digital-services/doc/Scalable-Network-report.pdf

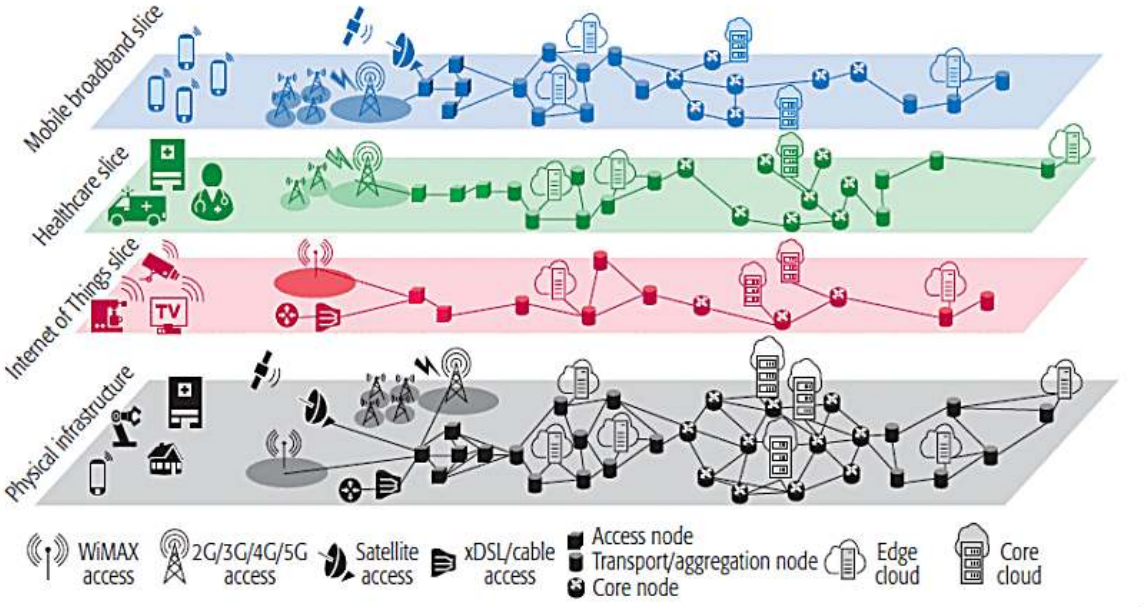

Figure 2.7: 5G network slices example. Above a multi-vendor and multi-access physical infrastructure, there are logical slices independently managed, each one of them addressing a specific use case [33].

in an IoT environment, there was a 35% increase in revenue over five years in comparison with a multi-service network. It was also shown that it reduced the OpEx by 40% and had a total economic benefit of 150%.

2.3.4 5G Network Management frameworks

With 5G networks, new management frameworks arise with the need to support new network business models (e.g. Internet as a Service (IaaS), Platform as a Service (PaaS), Network as a Service (NaaS)) and new users with complex requirements. The perfect management framework works without the intervention of network administrators, dealing autonomically with network failures, security attacks and inefficient resource utilization. With a fully autonomous management framework, it is possible to lower the OpEx, reduce the time to solve network failures or even predict those failures before they can happen.

In 4G, only some restricted mechanisms are automated, mainly at a local level. This is also linked with the fact that previous mobile network generations adhere to the traditional FCAPS model, which makes it hard to get a coherent picture of the holistic network status [4]. Early work by Mwanje, Decarreau, Mannweiler, *et al.* [35] also shows that, with the heterogeneity of network elements in 5G, the systems will demand an enhanced level of automation that is far beyond what 4G management systems can provide. In 5G, machine learning will be determinant to take fast and accurate decisions about the network status as a whole.

In a report by Aviat Networks[10], it was shown that the OpEx cost was three times the CapEx cost, and it keeps rising. The high OpEx cost combined with the new 5G network requirements forces the mobile networks to have a different approach to management frameworks, making them more intelligent and capable of automatically act on the network [36]. Therefore, autonomic networking will emerge as an important trend in the management of the next generation networks.

In the following subsections, it will be presented management frameworks designed for 5G networks, with the network automation phase already incorporated.

SELFNET

The EU H2020 SELFNET project [36] has the aim of implementing "an autonomic network management framework to achieve self-organizing capabilities in managing network infrastructures by automatically detecting and mitigating a range of common network problems that are currently still being manually addressed by network operators, thereby significantly reducing operational costs and improving user experience"[11]. This framework takes advantage of techniques such as SDN, NFV, machine learning and self-organizing networks to implement the framework, that has the following use cases:

- **Self-protection** - detect distributed cyber-attacks and restore the regular state of the network, with NFVs distributed across network elements;
- **Self-healing** - detect and predict network failures, such as hardware failures, software failures or power supply outages, and recover from them in an autonomic way;
- **Self-optimization** - dynamically improving the performance of the network resources and, with that, improve the Quality of Experience (QoE) of the users.

The SELFNET architecture (Figure 2.8) layers are briefly explained below:

[10] `https://aviatnetworks.com/media/files/Top_10_Pain_Points_For_Ops.pdf`
[11] `https://selfnet-5g.eu/`

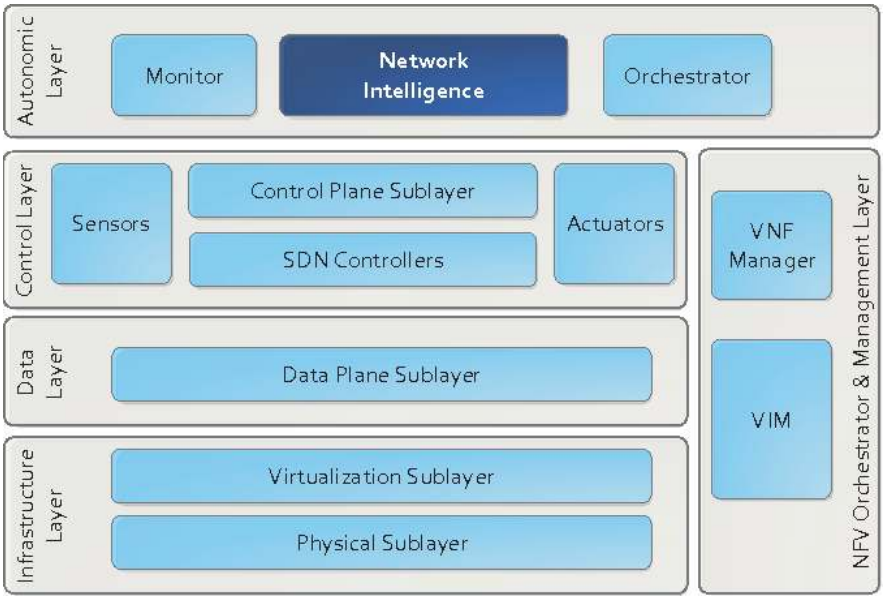

Figure 2.8: Reference architecture of SELFNET framework [36].

- **Infrastructure Layer** - this layer encompasses the physical and virtual network resources, represented in the figure by the physical and virtualization sublayer, respectively. The physical sublayer provides access to physical resources, while the virtual sublayer provides virtualization on top of the physical resources;
- **Data Layer** - to respect the SDN architecture that separates the control plane from the data plane, this layer abstracts the resources from a data view to the upper layers;
- **Control Layer** - this layer contains four internal modules: the control plane sublayer and the SDN controllers are responsible for communicating with the data layer; the sensors and actuators are responsible for collecting data from the network infrastructure and enforcing actions, respectively;
- **Autonomic Layer** - this layer contains the network intelligence of the infrastructure. The monitor extracts features related to the network. Those features are received by the network intelligence to decide which actions to be taken. Those actions are then passed to the orchestrator, that manages the actuators to execute the action;
- **NFV orchestrator & Management Layer** - this layer controls the VNF with the VNF manager, as well as the virtual resources with the Virtual Infrastructure Manager (VIM).

The SELFNET framework deals with an anomaly in the network in the form of a closed loop (Figure 2.9). The sensors send data to the monitor, that extracts the features and sends them to the network intelligence. If the network intelligence recognizes a problem, takes an action decision and sends it to the orchestrator, which coordinates the physical and virtual resources to enforce the action.

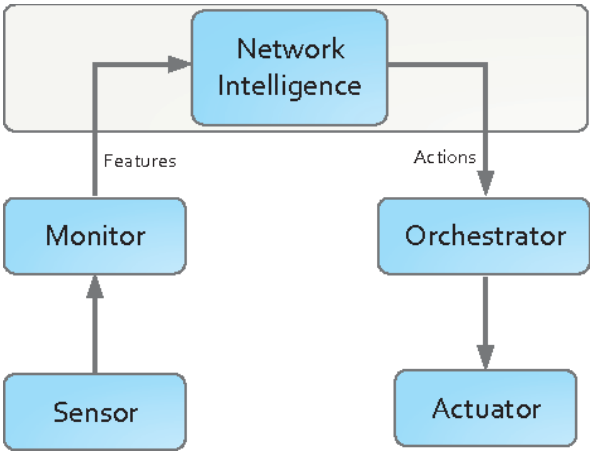

Figure 2.9: Selfnet control loop [36].

COMPA

COMPA is an autonomic networking architecture presented by Ericsson. This modular framework is based on a control loop composed of three tasks [37]:

- **Control, Orchestration and Management** "for executing control decisions";
- **Policy** "responsible for intelligence and dynamic governance to achieve control decisions";
- **Analytics** "providing real-time insights to influence control decisions".

The control loop (Figure 2.10) is subdivided into three minor loops. The functional loop, represented by the red arrows, starts by analyzing (A) the incoming data with the use of machine learning and statistical analytics to discover and understand trends and patterns in the event stream. The patterns and predictions are then passed to policy (P), that recommends or decides to perform actions automatically. COM performs semantic, functional and non-functional validation of recommendations and requests.

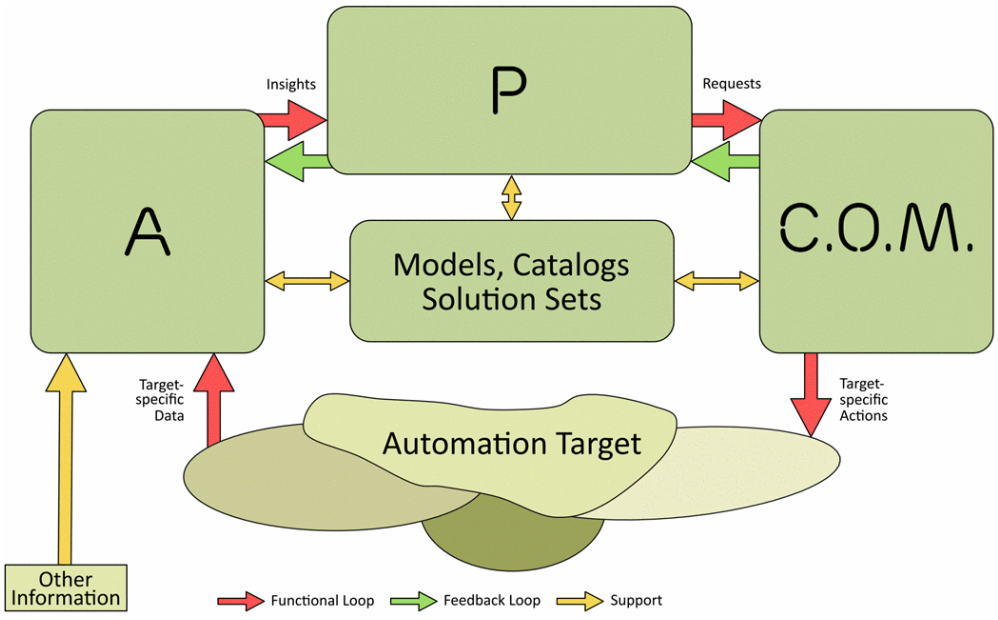

Figure 2.10: COMPA Autonomic control loop [37].

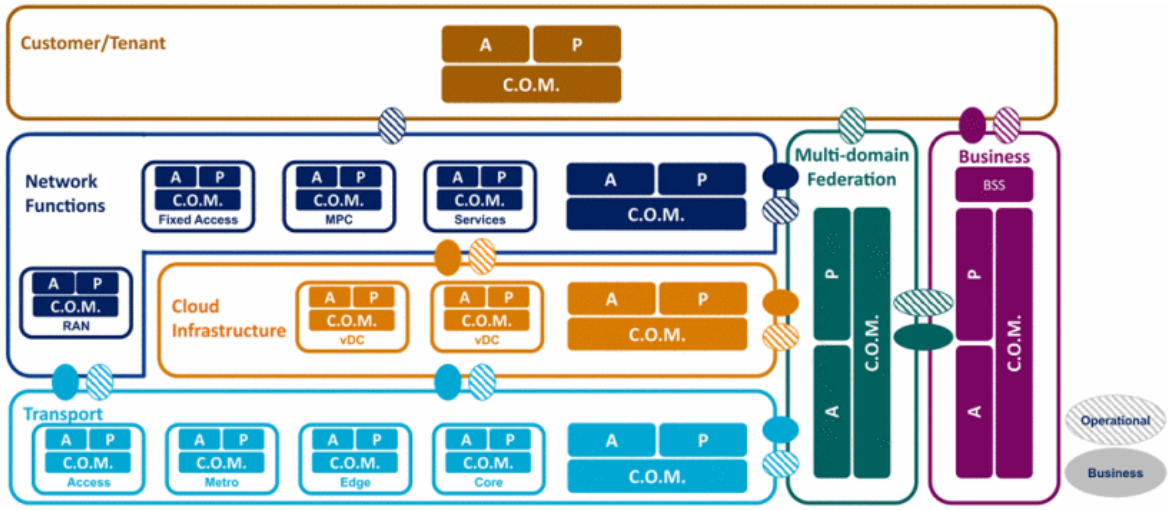

Figure 2.11: COMPA Architecture & Responsability domains [37].

The feedback loop (in green) goes the reverse way and can be useful to tune analytics and complex event processing. The support flow (in yellow) to and from the "Models, Catalogs, Solution Sets" supports the operations with useful data.

The COMPA adaptative control loop is common at different levels of the network. The COMPA architecture defines six federated domain layers with specific tasks: customer/tenant, network functions, cloud infrastructure, transport networks, multi-domain federation and business layers. Inside each layer, the COMPA control loop is applied (Figure 2.11).

ONAP

ONAP[12] is the result of the merging of two of the largest open-source networking initiatives, Enhanced Control, Orchestration, Management & Policy (ECOMP) and Open Orchestrator Project (OPEN-O), into one open-source project hosted by Linux Foundation. Its first release was in February of 2017.

Taking advantage of the benefits from both projects, the platform aims at automating, orchestrating and managing VNFs and network services. The main advantage of ONAP is the flexible and scalable unified architecture, which supports new components, enabling end-users to create their own VNFs.

The ONAP architecture can be split into two main categories: the Design-Time Framework and the Execution-Time Framework. The design-time framework defines the workflow for instantiating, monitoring and managing VNFs and services, and sends those design rules to the execution-time framework. The execution time framework is an autonomic framework that manages the lifecycle of VNFs and services, and also delivers a view on the available resources and services [38]. The Casablanca version of the architecture (November 2018) (Figure 2.12) is explained below.

The design-time framework has two main subcomponents [39]:

- **Service Design & Creation (SDC)** - the environment that describes how VNFs or services are managed, based on metadata description;
- **Policy Creation** - defines control, orchestration and management policies with a set of rules.

The execution-time framework has the following subcomponents[13] [38] [39]:

- **Active and Available Inventory (AAI)** - provides a real-time view of the topology and the available resources;

[12] `https://www.onap.org/`
[13] `http://wiki.onap.org/`

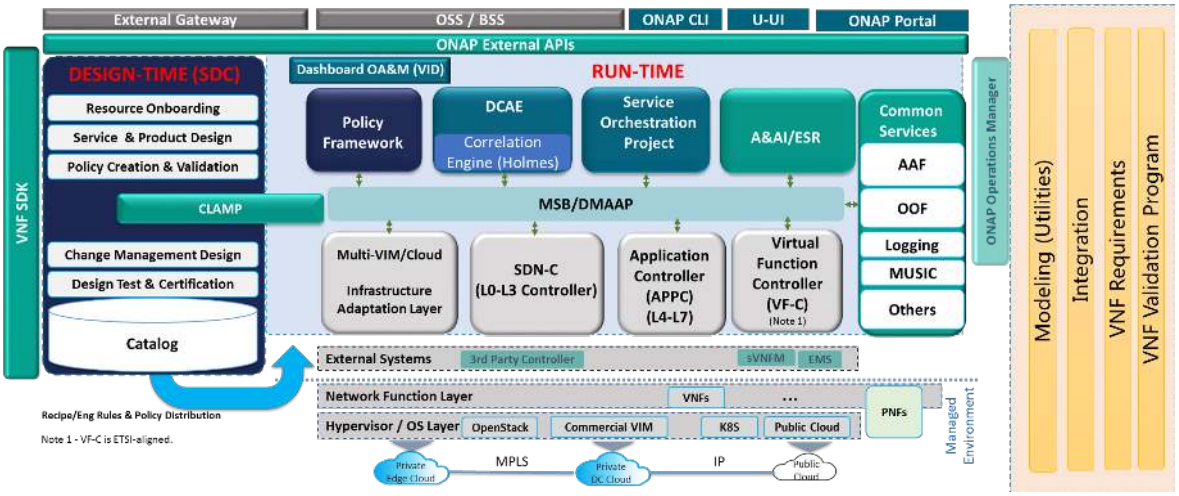

Figure 2.12: ONAP Casablanca architecture[5], November 2018.

- **Controllers** - each controller manages the state of a single resource. Executes the resource configuration and installation, and does the resource management, such as control loop actions, migration and scaling. There are three types of controllers: network controllers (configures the network and manages VNFs), application controllers (manages more complicated VNFs and services) and infrastructure controllers (manage the cloud's infrastructure resources). The orchestration by the controllers is a lower-level orchestration when compared to the orchestration done by the Master Service Orchestrator (MSO);
- **Master Service Orchestrator (MSO)** - uses the controllers to provide the capabilities of end-to-end service provisioning;
- **Data Collection, Analytics and Events (DCAE)** - collects metrics from VNFs, allowing analytic applications to detect network anomalies and act on the network;
- **Dashboard (Portal)** - unifies the execution time framework data to show metrics to the user;
- **ONAP Optimization Framework (OOF)** - policy-driven and model-driven framework for creating optimization applications for various use cases;
- **Security Framework** - provides security functions to two main aspects of the platform: security of the platform itself and security of the cloud services.

Just like COMPA, ONAP also defines a control loop - Closed Loop Automation Management Platform (CLAMP). The control loop is used to provide automation to proactively respond to network and service conditions without human intervention. A high-level view of the control loop has seven phases: Design, Create, Collect, Analyze, Detect, Publish and Respond [38].

5G NORMA

5G NORMA is a project under the 5G Infrastructure Public Public Private Partnership (5G PPP) horizon 2020 framework, and its main objective is to develop a novel, adaptative and future-proof 5G network architecture. 5G NORMA architecture design has three major objectives [40]:

- **Service-aware resource sharing**, by enabling the construction of fully decoupled end-to-end networks on top of shared infrastructure, that can be used to host heterogeneous services;
- **Network customization by an adaptative allocation of network functions**, using NFV;
- **Network programmability for flexible network control**, using SDNs.

5G NORMA functional architecture is designed in a modular manner, to provide the opportunity to vendors, mobile service providers and infrastructure providers for fine-grained offerings to their

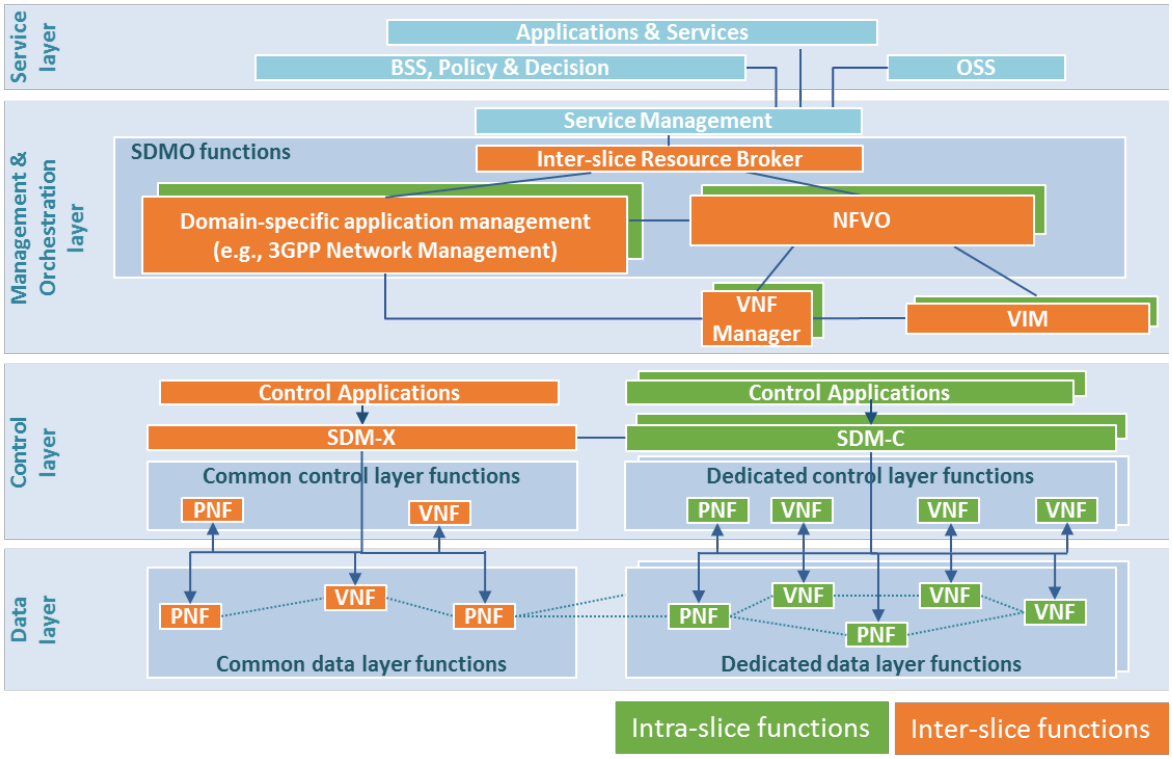

Figure 2.13: 5G NORMA functional architecture [41].

customers. The architecture has to support the 5G requirements, like multiple telecommunications services, heterogeneous KPIs and sharing the same infrastructure. The four layers of the architecture (Figure 2.13) are explained below [41]:

- **Data Layer** - this layer comprises the VNFs and the Physical Network Functions (PNFs) needed to process user data traffic;
- **Control Layer** - this layer is responsible for the control of the virtual functions of the network. There are two main controllers: the software-defined mobile network coordinator (SDM-X) for the control of shared Network Functions (NFs) and the software-defined mobile network controller (SDM-C) for dedicated NFs;
- **Management & Orchestration (MANO) Layer** - this layer is responsible for the NFV management and orchestration. For that reason, it comprises the Virtual Infrastructure Manager (VIM), VNF manager and NFV orchestrator components. Furthermore, the layer accomodates application management functions from various domains, and also contains the Inter-slice Resource Broker (ISRB) to configure and enforce policies for cross-slice resource allocation;
- **Service Layer** - this layer is not in the scope of the project. It contains the Business Support Systems, business-level policy and decision functions and the applications and services operated by the tenant.

CogNet – Building an Intelligent System of Insights and Action for 5G Network Management

CogNet (from Cognitive Networks)[14] is a project under the 5G PPP horizon 2020 framework that proposes an autonomic self-managing network based on NFV and machine learning. A key point in the architecture is the capability of enabling batch and real-time machine learning solutions to an ecosystem that can scale horizontally or vertically in various 5G scenarios [42].

[14] `https://5g-ppp.eu/cognet/`

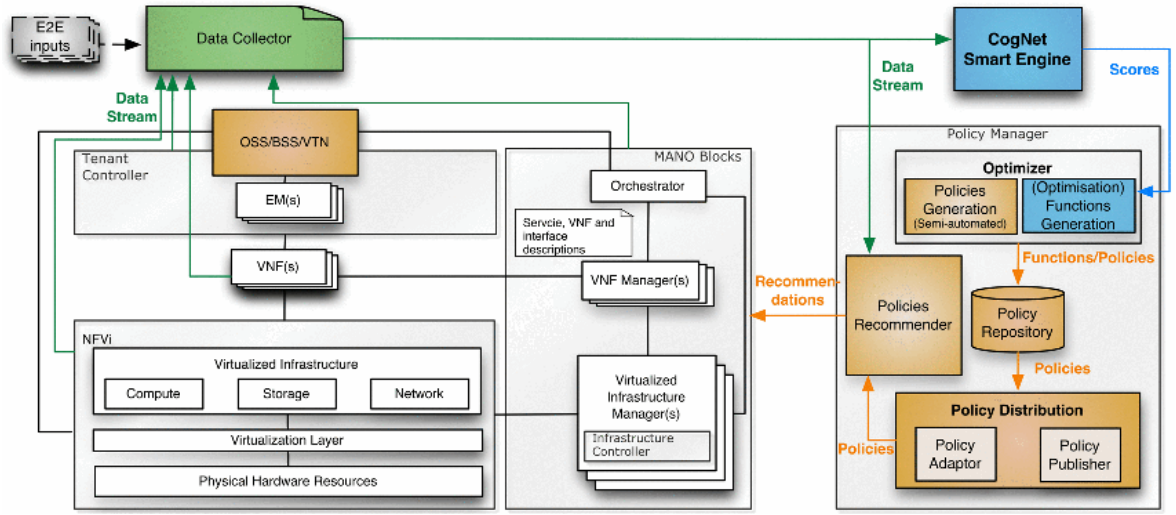

Figure 2.14: Architecture of the CogNet Project [42].

The high-level architecture is presented in Figure 2.14. The Cognitive Smart Engine (CSE) will process the collected records from the network in (near) real-time or periodically for various purposes, such as dynamic resource allocation, demand prediction or security threats. The output of the CSE will be sent to the policy manager for policy generation. The policy manager has two functions: translate the rules from CSE into policies and recommend the policies to the NFV architectural framework [42]. Finally, the NFV framework decouples the software of a given network function from the hardware, allowing for a hardware abstraction when applying network configurations.

T-NOVA

T-NOVA is a framework that "aims at designing and implementing an integrated management architecture, including an Orchestrator platform, for the automated provision, management, monitoring and optimization of Virtualised Network Functions over Network/IT infrastructures"[15]. The T-NOVA system architecture is made of four main layers [43]:

- The **NFV Infrastructure layer** encompasses the network elements where network services are deployed;
- the **NFV Infrastructure Management layer** has the entities responsible for the management of the NFV infrastructure layer;
- the **Orchestration layer** is the core component of T-NOVA. His main role is to manage the VNFs lifecycle operations over network elements. It also includes a "Network Function Store", that is a repository for all VNFs;
- the **Marketplace layer** has interfaces and modules that communicate with the outside, exposing all the functionalities of the layers underneath. It facilitates the integration among various business activities.

Other research works

In this subsection, it will be presented other proposals for 5G management frameworks.

CROWD [44] is a collaborative project funded by the European Commission that has the goal of creating a network framework designed to provide mechanisms to tackle the high densification and heterogeneity of wireless networks. The CROWD architecture follows an SDN approach, with a

[15] `http://www.t-nova.eu/`

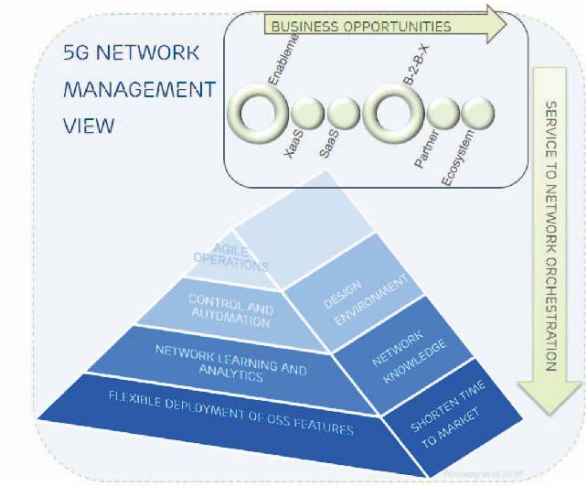

Figure 2.15: 5G cognitive management architecture [4].

two-tier SDN controller hierarchy: a local controller that takes fast decisions with a limited vision of the network, and a regional controller, that takes long time-scale decisions with a broader vision of the network [44].

Bosneag and Wang [4] proposed a holistic management perspective of the network. In comparison with the TMN model, aspects of the next generation networks are introduced, like virtualization, automation or analytics. The architecture also adopts a strategy of decomposition and layering (Figure 2.15):

- **Flexible deployment of OSS features layer** - this layer is the core layer for the architecture. It consists of a flexible deployment, where the nodes dynamically adjust their VNFs according to the network conditions and information coming from other layers;
- **Network learning and analytics layer** - this layer will gather metrics from the network and learn with them, with machine learning and other analytics techniques. The knowledge obtained in this layer will be used on the upper layer;
- **Control and automation layer** - with the knowledge from the underneath layer, this third layer will make decisions and act on the network to make them more robust;
- **Agile operations layer** - the upper layer allows the flexible implementation of operations, using the underneath layers to understand the current state of the network.

Celdrán, Pérez, Clemente, *et al.* [45] proposed a 5G oriented architecture that considers the SDN and NFV control plane. The architecture is able to monitor and orchestrate the life-cycle of services using information from the network control plane, such as the number of gathered flows per second or the percentage of CPU. The architecture is composed of five layers:

- **Virtualized Infrastructure (VI)** is in charge of managing the life cycle of the virtual machines, instantiated over the physical network elements;
- **Virtualized Network Functions (VNF)** contains the VNFs, that are controlled by the VNF manager, that has the duty of creating and monitor VNFs on the virtual machines exposed by the layer underneath;
- **Software-Defined Networking (SDN) paradigm** is at the same level as the previous layer, and the main job is to decouple the control plane from the data plane;
- **Control and Data Plane Management** handles the data coming from the layer underneath, already separated in control plane and data plane;

- **Operations and Business Support Systems (OSS/BSS)** has a set of rules and policies to react and orchestrate the decision-making results.

The model manages the network at run-time using management-oriented policies. The main categories of those policies are SDN policies (allow managing the elements in the SDN paradigm), VNF policies (capable of managing the internal behavior of VNFs), Virtual Infrastructure policies (manages the virtual network infrastructure automatically) and Physical Infrastructure policies (controlling the physical resources of the network infrastructure).

2.4 Machine Learning

The name *machine learning* was invented by Arthur Samuel in 1959 while at IBM [46], as *"the field of study that gives computers the ability to learn without being explicitly programmed"*. The current definition of *machine learning* is more technical and it was given by Tom M. Mitchell [47]: *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"*. Machine learning is a subset of the artificial intelligence field that tries to achieve human-like intelligence with the use of statistical techniques that enable machines to improve at tasks with experience. The experience is the data that machine learning algorithms use to create a mathematical model. According to the type of learning and the desired objective, a performance metric has to be defined to evaluate the output of the algorithm to be optimized.

There are three broad categories of learning algorithms, according to the type of learning that is done by the mathematical model:

- **Supervised learning** - uses labeled training datasets. This learning technique is employed to learn to identify patterns or behaviors in a training dataset where the output is known. Typically, this approach is used to solve classification and regression problems;
- **Unsupervised learning** - uses unlabeled training datasets to create models that can recognize patterns in the data. It is widely used for clustering problems;
- **Reinforcement learning** - is an iterative process where an agent interacts with the external world and it learns by exploring the environment. The actions taken by the agent can be rewarded or penalized, and its goal is to maximize (or minimize) a performance metric.

In this dissertation, it will only be used supervised learning algorithms to solve prediction and classification problems. Specifically, Artificial Neural Networks (ANNs) will be used and compared with classical machine learning algorithms and other statistical algorithms.

With the increase of the amount of data available, Deep Learning is becoming increasingly important and an essential field in machine learning. Deep Learning is the subset of machine learning that uses ANNs with more than one hidden layer to model a non-linear function.

ANNs are widely used for modeling and predicting time-series data due to their capacity for learning complex patterns. An ANN is inspired by the biological neural networks that constitute animal brains. The neural networks are theoretically able to estimate any function where there is a relationship between the input and the output. The parameters of the neural network are determined only by the dataset and are not limited to any analytical model, making this approach able to predict non-linear relationships in the data and capable of achieving state of the art performance with a high amount of data, when compared with classical machine learning algorithms.

Three types of ANNs will be described in the next subsections. A statistical model used for forecasting, ARIMA, will also be described.

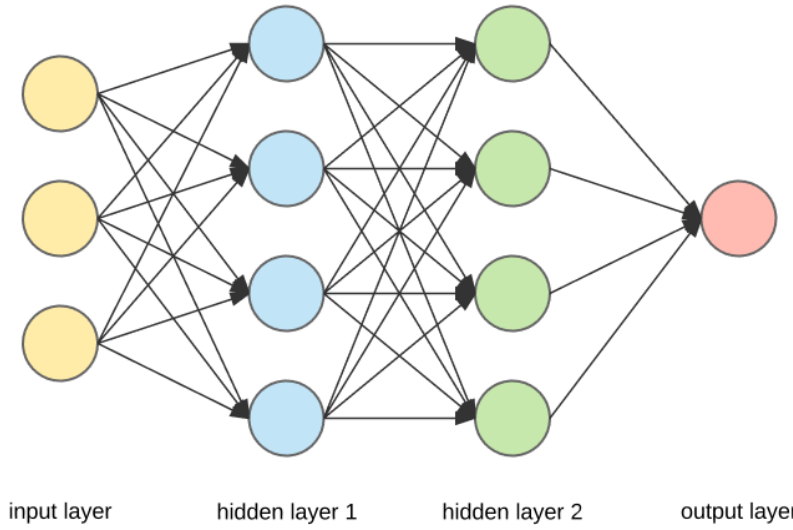

Figure 2.16: Example of an architecture of an ANN with two hidden layers with four neurons each, with 3 neurons in the input layer and only one neuron in the output layer [16]

2.4.1 Feed-Forward Neural Networks

Feed-forward neural networks are the type of ANNs most used. In feed-forward neural networks, the neural network architecture is composed of nodes, called neurons, each one with an activation function that defines the output of the neuron. The neurons are connected between themselves, and each connection has a weight. There can be several layers of neurons. The neurons of each layer are connected with all the neurons in the previous and next layer (Figure 2.16).

In the training phase of an ANN, the training set is presented to the network and the weights of the neurons are adjusted to predict the correct value for the input set. The backpropagation algorithm [48] is used to update the weights according to the output error of a loss function, using an optimizer function. There are two essential hyper-parameters in the training phase. The batch size is the number of training examples in one forward and backward pass in the network. After the number of samples specified in a batch size is analyzed, the internal parameters of the neural network are updated using the backpropagation algorithm. The number of epochs defines the number of times that the learning algorithm will work through the whole dataset. If the number of epochs is too low, the neural network will not be able to learn the mapping between the input and the output, and the model will be underfitting. However, if the number of epochs is too high, the neural network will overfit the training data (if the network is big enough). In the prediction phase, the input is presented to the first layer of the ANN and the output is calculated using only a forward pass.

2.4.2 Recurrent Neural Networks

A recurrent neural network is a type of ANN that takes advantage of internal memory to maintain a state with information about the previous inputs (Figure 2.17). It is possible to maintain or to reset the internal state between the processing of two different inputs. If the state is reset, the recurrent network will only maintain a state depending on the input values in the input sequence.

With the use of recurrent neural networks, in theory, it is possible to store information from arbitrarily long time ago. This is an advantage over the feed-forward neural networks if there are

[16] https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

[17] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

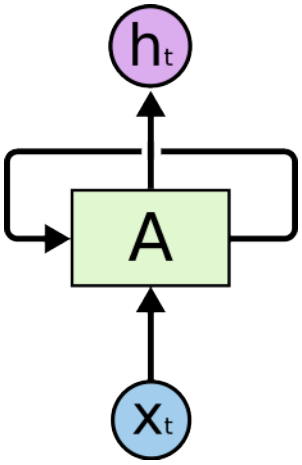

Figure 2.17: General architecture of a recurrent neural network[17]

any long-term dependencies that cannot be learned by time-lagging the data and use it as input. However, if that is not the case, the recurrent neural networks will perform the same or worse than the feed-forward neural networks.

The LSTM is a particular type of recurrent neural network that prevents the recurrent neural network from the exploding/vanishing problem [49], a difficulty when training general recurrent neural networks that makes the update of the network weights too big or non-existing [50]. However, the LSTM also has some drawbacks. It is more expensive and hard to train the network due to the complex internal architecture of each neuron, making it more time consuming and harder to find an optimal solution.

The internal architecture of an LSTM cell is composed of an input gate, an output gate and a forget gate (Figure 2.18). The forget gate, represented by σ_1, outputs a number between 0 and 1 for each number in the cell state C_{t-1}, taking into account the current input x_t and the previous output h_{t-1}. The forget gate specifies how much of the previous information will be saved in the current cell state. The input gate, represented by σ_2, decides what values will be updated to the current cell state, according to a vector of new candidate values created by the $tahn_1$ activation function. Finally, the output gate is represented by σ_3 and decides which values from the cell state will be used for the output, after a transformation with the $tanh_2$ function, to shrink values between -1 and 1.

The formulas for the LSTM are described in the Equation 2.1 [49]. The σ_x represents the gates, while $\sigma(x)$ and $tanh_x$ represent the application of the sigmoid and hyperbolic tangent functions, respectively. The matrices W_x, U_x and b_x contain the learned weights of the connections.

$$
\begin{aligned}
\sigma_1 &= \sigma(W_1 X_t + U_1 h_{t-1} + b_1) \\
\sigma_2 &= \sigma(W_2 X_t + U_2 h_{t-1} + b_2) \\
\sigma_3 &= \sigma(W_3 X_t + U_3 h_{t-1} + b_3) \\
C_t &= \sigma_1 * C_{t-1} + \sigma_2 * tanh_1(W_4 X_t + U_4 h_{t-1} + b_4) \\
h_t &= \sigma_3 * tanh(C_t)
\end{aligned}
\tag{2.1}
$$

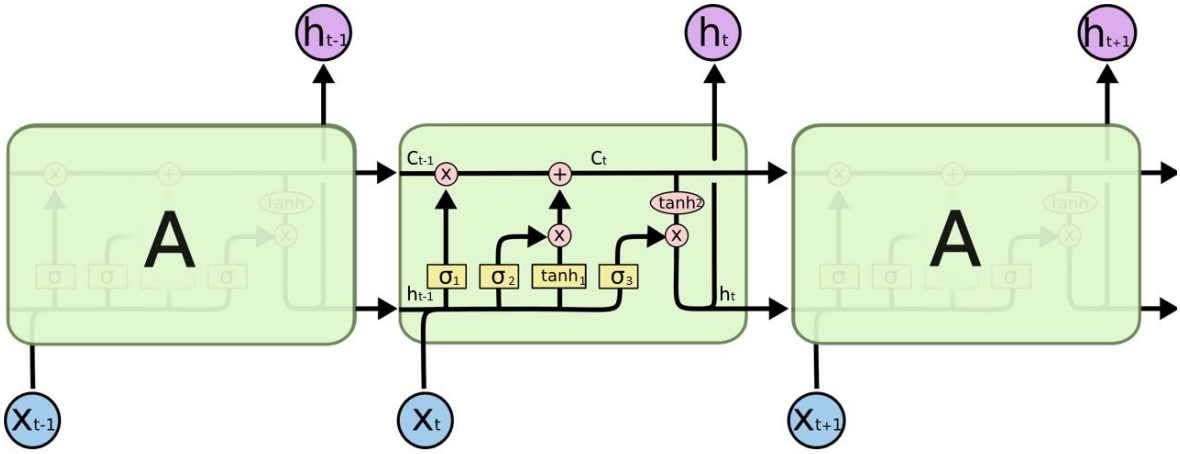

Figure 2.18: Internal architecture of an LSTM cell[3][Adapted].

2.4.3 1-D Convolutional Network Layers

Convolutional layers have proven to be very effective in the computer vision area, where the spatial locality of the pixels can be used to reduce the computations needed to achieve state-of-the-art performance in image recognition and image classification tasks.

Similarly, convolutional networks (convnets - ANN with convolutional layers) can be used in sequence processing, extracting features from local input patches and allowing for representation modularity and data efficiency. The property of locality used in computer vision problems can also be helpful in sequence processing, with time being treated as a spatial dimension, like the height or the width of a 2D image.

Depending on the dataset, 1-dimensional convnets can be competitive with recurrent neural networks or feed-forward neural networks in the prediction task, with a much cheaper computational cost. A visual explanation of a 1-D convolutional layer can be seen in Figure 2.19. Each output timestep is obtained from a temporal patch in the input sequence. In the specific case of predicting the number of sessions at each timestep, the number of input features per timestep is one. Each neuron can learn a different pattern to recognize, being able to use it at different positions of the input.

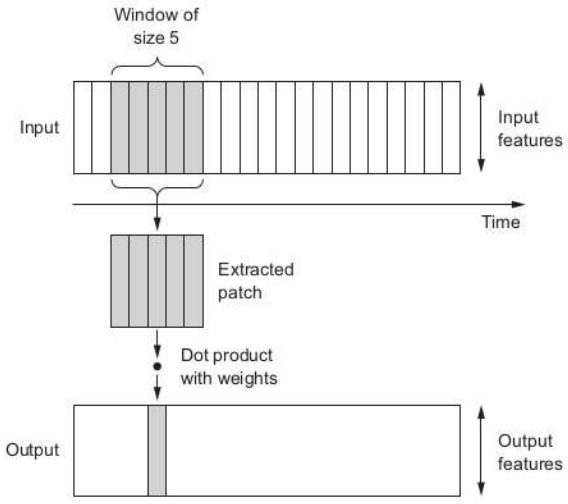

Figure 2.19: Example of a 1-D convolutional layer [51].

2.4.4 ARIMA

The ARIMA model [52] is one of the most popular statistical methods used for modeling and forecasting linear time-series. Although it is not considered to be a machine learning algorithm, it will be used to compare its results with other machine learning algorithms. The model is based on a linear combination of three components:

- AR (AutoRegression) - linear combination of the past values of the variable:

$$\hat{y}_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t, \tag{2.2}$$

  where ϕ_t is the coefficient associated with the value timestamp t, there are p lag observations included in the model, c is a constant and ε_t is white noise;
- I (Integrated) - differentiation of raw observations to make the time-series stationary; in this case, the differentiation will be a seasonal differentiation, as represented in equation:

$$\hat{y'}_t = \hat{y}_t - y_{t-d}, \tag{2.3}$$

  where $\hat{y'}_t$ represents the differenced predictions at time t and d is the differentiation parameter;
- MA (Moving Average) - linear combination of the past forecast errors in the variable:

$$\hat{y}_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}, \tag{2.4}$$

  where θ_u is the coefficient associated with the timestamp $t-u$, ε_t is the white noise at timestamp t, c is a constant and q is the size of the moving average window.

Therefore, taking into account the three components, the ARIMA equation can be written as

$$\hat{y'}_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t, \tag{2.5}$$

where y' is the differenced series. The equation includes both lagged values and lagged errors. This model has three hyper-parameters that must be set by the user:

- p - number of lags included in the model;
- d - degree of differentiation;
- q - size of the moving average window.

2.5 Summary

The first two sections of this chapter described the network management frameworks of the previous generation networks and the network management protocols. While for historical reasons and teaching purposes the OAM&P, FCAPS and TMN frameworks are important, eTOM and ITIL are the industry *de-facto* standards for network management reference models. Since eTOM and ITIL have different goals, they can be combined for the construction of a network management architecture [53]. The third section presented the most important 5G concepts, as well as the 5G management frameworks. Currently, ONAP is the choice for the management framework of 5G operator networks, mainly due to its open-source core, active support by the Linux Foundation and active development of new features. However, because the 5G networks are still not extensively deployed around the world, there is no standard when it comes to 5G management frameworks. In the last section, the machine learning concepts were introduced, and three types of neural networks were explained more in-depth, as well as ARIMA, a statistical model for forecasting. These algorithms will be used in this dissertation to predict network KPIs.

In the next chapter, the state of the art of machine learning approaches in management networks is presented, for a better understanding on how to use machine learning to perform the management of a network.

3

State of the art

Machine learning is an important trend in network management frameworks, given that almost all of the presented network management frameworks for 5G (Subsection 2.3.4) include an automation part to implement the monitoring and acting on the network without the need for human intervention. Machine learning enables the capability of improving network performance, reliability, stability, among other factors.

Moysen and Giupponi [54] have identified the classes of problems that are usually addressed when managing autonomously the network and that can be solved using machine learning techniques:

- **Variable estimation or classification** - traffic estimation or classification of the network KPIs are two examples of these tasks. Usually, to solve them it is applied regression or classification with supervised machine learning techniques;
- **Diagnosis of network faults or misbehaviors** - this class aims at detecting issues in the network. Usual machine learning techniques to solve this problem fall under the category of unsupervised learning techniques;
- **Dimensionality reduction** - this class encompasses the problems where the goal is to reduce the noise from the data generated by the network. It is a transversal problem in the machine learning field and it can be solved with unsupervised learning techniques;
- **Pattern identification, grouping** - this class of problems has the objective of identifying similar patterns in network data and group them. Automated network slicing is an example of such a task, as well as self-configuration use cases. Usually, clustering algorithms based on unsupervised learning techniques are used to solve them;
- **Sequential decision problems for online parameter adjustment** - this class is the most common in autonomous management, where control decision problems are posed to adjust the network parameters. The most appropriate decision is learned online, based on the reaction of the environment to the actions the network is taking. The self-optimization problems fall under this category. The approach usually taken is to use a reinforcement learning technique.

In this chapter, it will be presented the state of the art of machine learning techniques applied to four network problems: network traffic forecasting, traffic matrix forecasting, network traffic classification and prediction of service-level agreement breaches. The four problems are being researched in the context of 5G networks. In this dissertation, the network traffic forecasting problem will be approached in the following chapters (Chapters 4, 5 and 6).

3.1 Network traffic forecasting

Traffic forecasting, along with classification, were two of the earliest machine learning applications in the networking field. Forecasting is a sub-discipline of prediction in which the predictions done are about the future, on the basis of time-series data. In this dissertation, the terms *network prediction* and *network forecasting* will be used interchangeably. Network traffic is characterized by self-similarity, multiscalarity, long-range dependence and a highly nonlinear nature [55], which proves not only that Poisson and Gaussian distributions cannot accurately model network traffic, but also that the network traffic has strong correlations and predictability, which makes it a suitable candidate to be modeled using machine learning techniques.

Having an accurate traffic forecasting is essential for many traffic management decisions in a network, such as traffic accounting, short-time traffic scheduling, long-term capacity planning, network design or network anomaly detection. For example, upon congestion in a network, traditional routing protocols cannot react in real-time to avoid congestion. A proactive prediction-based approach would be faster to detect a possible congestion and to act accordingly in the network to avoid it [56]. Security attacks like Distributed Denial of Service (DDoS) can also be detected and predicted, and measures can be taken to provide more reliable service even when under attack.

Ding, Canu, Denoeux, *et al.* [57] classify the forecasting types as:

- **Long-term forecasts**, corresponding to yearly evolution analysis, and often needed for strategic decisions;
- **Middle-term forecasts**, typically issued several months in advance;
- **Short-term forecasts**, associated with a lead time varying from one day to several hours, crucial for optimal control or detection of abnormal situations;
- **Real-time forecasts**, when the predicted period does not exceed a few minutes; it requires an online forecasting system.

Depending on the data available and the sampling period (the time between samples), it may be adequate to use a different class of forecasting. For example, if the data samples have a sampling period of a day, it may be useful to use middle-term or long-term forecasts. If the data has a sampling period of a few seconds and it is only available from a period of a month, it is more adequate to perform short-term or real-time forecasts.

Network traffic forecasting methods can be divided into linear forecasting or non-linear forecasting. The Autoregressive moving average (ARMA)/ARIMA algorithm and the Holt-Winters algorithm are the most widely used traditional linear forecasting methods [56]. The ARIMA model is a generalization of the ARMA model, which is a stationary stochastic process, in terms of two polynomials: an autoregression polynomial and a moving average polynomial (Subsection 2.4.4). The Holt-Winters algorithm is a forecasting technique from the family of Exponential Smoothing methods. Non-linear forecasting methods usually involve an Artificial Neural Network (ANN). An ANN is a machine learning model based on the human brain. The ability to learn non-linear functions in an efficient and stable manner makes ANNs one of the most used techniques in machine learning today. The most common type of ANNs are Feed-Forward Neural Networks, where the information flow is unidirectional and there are no feedback loops (Subsection 2.4.1).

In the work of Feng and Shu [58], linear and non-linear traffic predictors were compared (ARIMA, FARIMA (Fractional ARIMA) [59], ANN and wavelet-based predictors) for traffic forecasting. An ANN with only one hidden layer showed the best results, along with FARIMA algorithm, that has the disadvantage of having a higher degree of complexity.

Cortez, Rio, Rocha, *et al.* [60] conducted an experiment to analyze and predict TCP/IP traffic using three algorithms: ARMA algorithm, the Holt-Winters algorithm and an ensemble of ANNs. An ensemble of ANNs is a method that consists of training various different ANNs and the final prediction is given by the average of the individual predictions. In general, ensembles are better than individual learners. In this case, it was trained five different ANNs. A naive benchmark method was also used to serve as a baseline for other forecasting approaches. The forecasting classes considered in the study were real-time forecasts (every five minutes) and short-term forecasts (hourly aggregated values), using data from two large ISPs. The experiment showed that the ensemble of ANNs outperformed the other three methods, obtaining a 1 to 3% error for the five-minute lookahead forecasts, 3 to 5% error for the one lookahead forecasts and 12 to 23% error for the 24-hour look-ahead forecasts. It was also concluded that the ARIMA methodology was impractical for online forecasting systems, due to the computational effort.

Barabas, Boanea, Rus, *et al.* [61] compared the forecasts of the transfer rate in a network link. The forecasts were produced using different types of ANNs and statistical prediction time-series algorithms (ARMA model, ARAR model, Holt-Winters algorithm). The ANNs applied used multi-task and multiresolution learning. Multi-task learning means that the model has extra tasks that are trained simultaneously with the main task, sharing the hidden layers of the ANN. By learning multiple tasks, the ANN can improve its prediction accuracy. In the work done in [61], two extra tasks were added: for the same link, if the main task was to predict the transfer rate at time t+1, the extra tasks were to predict the transfer rate at time t and at time t+2. Multiresolution learning means that the traditional training is decomposed in several stages, each involving a dataset of a certain resolution. To create multiple representations of the training data, the Haar wavelet [62] was applied to the network traffic, to perform multiresolution decomposition. The goal was to train the ANN with the coarsest resolution, followed by finer ones, and finally learning the original resolution of the dataset. The results of the paper showed that non-linear traffic forecasting approaches based on ANN outperformed linear forecasting models. The best results were obtained using the ANN predictor with multiresolution learning.

A different approach is taken by Cui, Yao, Zhang, *et al.* [63], where a distributed network traffic forecasting system is built, using Hadoop, to meet the requirements of network operators of operating massive amounts of data in real-time. An Echo State Network (ECN) [64] is used as the ANN model. An ECN is a recurrent neural network with a simple training process and excellent modeling performance, consisting of three layers: the input layer, the reservoir layer and the output layer. In this study, five mobile network services were chosen to forecast the traffic volume in a city: MMS, web, streaming media, QQ (the most popular instant messaging application in China) and XunLei (the most popular Peer-to-Peer (P2P) application in China). With a sampling interval of one minute, the first five days were used as training data and the sixth day was used as the test set. The results confirmed the good accuracy of the ECN in network traffic forecasting.

The goal of the work in [65] was to forecast the traffic per hour on a regular day in a laboratory. The approach used was to apply a combination of a genetic algorithm [66] and a wavelet ANN. A wavelet ANN is an ANN structure based on the theory of wavelet transformations. When the ANN topology is determined, the genetic algorithm is used to optimize the network connection weights. The experimental data was decomposed into low-frequency traffic component and high-frequency traffic component. Although an experimental measure was not employed in the study, the paper states that the learned model could accurately predict the high-frequency and low-frequency traffic.

Chen, Wen, and Geng [67] proposed a hidden Markov model based on kernel Bayes rule and

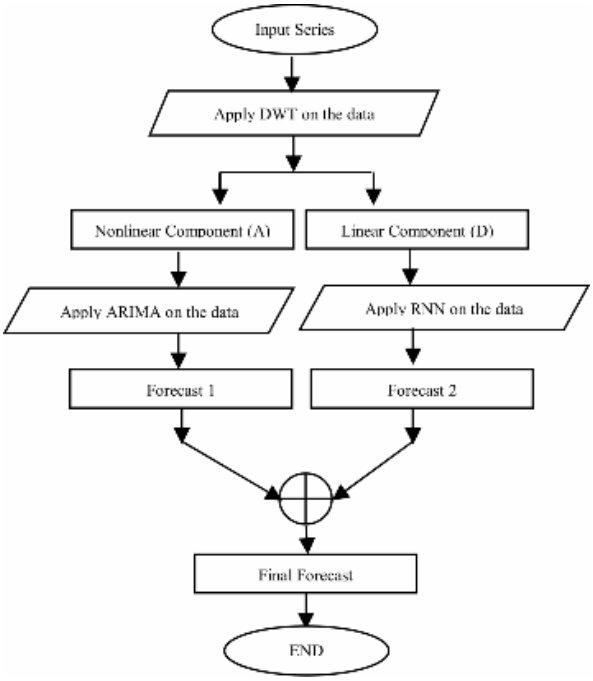

Figure 3.1: Approach proposed by Madan and Mangipudi [68], where the network traffic is divided in two components before the predictions are applied for each component, and then are added again to obtain the final forecasts.

a recurrent neural network to predict the traffic volume using simple flow statistics which are not expensive to collect. The aim of the paper was to forecast the traffic volume without measuring the actual traffic volume, mainly because in large-scale networks the measurement of the actual traffic volume is too expensive to be feasible, and is not scalable in terms of complexity and storage. The metric collected used to forecast future network traffic is the flow count. With a hidden Markov model that describes the relationship between the flow count and the flow volume, it is possible to use machine learning algorithms like kernel Bayes rule or recurrent neural networks to train the model to forecast the traffic volume. In the study, the kernel Bayes rule performed better than the forecast using recurrent neural networks. The paper showed promising results in the use of flow-level statistics to forecast the traffic volume.

The Madan and Mangipudi [68] approach takes advantage of the network traffic decomposition that can be done, in linear and non-linear components. That decomposition is done using a Discrete Wavelet Transform (DWT). After that, the linear component is forecasted using the ARIMA model, because the ARIMA model is best suited for stationary data, and the non-linear component is forecasted using a Recurrent Neural Network (RNN), more adequate to non-linear data. The forecasts of both components are then averaged to achieve the final forecasts (Figure 3.1). The proposed technique outperforms the forecasts of ARIMA and RNN individually.

Hua, Zhao, Li, *et al.* [69] work focuses on reducing the training time of LSTMs (Subsection 2.4.2). Besides being the best approach for network traffic forecasting, they take too much time and computational resources to train. To reduce the training complexity, the neuron connections are done in a stochastic manner rather than fully-connected. The results showed that even only 35% of neural connectivity shows a satisfactory performance in the traffic forecasting, while being much easier to train.

3.2 Traffic matrix forecasting

A traffic matrix is a matrix with size $N * N$, where N is the number of nodes in the network. Each entry $y_{i,\,j}$ in the matrix represents the traffic volume flowing from node i to j. Traffic matrix forecasting can be seen as a general case of a traffic forecasting problem, since traffic forecasting can be calculated by adding all entries in the traffic matrix. Traffic matrix forecasting is generally a harder problem than traffic forecasting.

According to various network management tasks, different kinds of granularities of traffic matrices are needed. The original and destination nodes can be prefixes, links, routers or Points-of-Presence. In a large-scale IP backbone, it can be difficult to obtain the network matrix with the granularity needed, due to the high number of flows in the network.

Because of the high deployment cost of flow-level measurement infrastructure, direct measurement on traffic matrices has been impractical in the past decade. There are statistical inference methods to estimate traffic matrices from network measurements, such as partial flow-level traffic measurements, SNMP-based link load and routing tables. The first generation methods of matrix estimation introduced additional constraints to model the traffic (e.g. Poisson distribution, Gaussian distribution), while the second generation methods (tomography model, gravity model and independent flow model) made assumptions about the independence of the flows between them. The third generation methods (Fanout, Kalman, PCA) rely on priors obtainer from direct measurements [70].

Traffic matrix forecasting is important for network management to identify anomalous flows before they disrupt the whole network. For example, in the case of a DDoS attack, it can be possible to detect which flows are disrupting the network and take preventive actions only affecting those flows.

In the work of Liu, Hong, Ou, *et al.* [70], it is presented an approach to forecast the traffic matrices in an IP backbone network. Three different methods were proposed to forecast the traffic matrix: Independent Node Prediction, that assumes that the node output traffics are independent and fits the simplest curve fitting method; Total Matrix Prediction with Key Element Correction, that takes the total traffic volume of the network as a concert, and the matrix elements are forecasted by apportioning the prediction of the total traffic; and Principal Component Prediction with Fluctuation Component Correction, that uses PCA to consider the inner increasing momentum in the network and ARIMA to predict the fluctuation component. The sampling period is one day, and the forecasting objectives are three: forecast the traffic of an individual flow, forecast the output traffic of one node and forecast the total network traffic. The results showed that the Principal Component Prediction with Fluctuation Component Correction has the lowest overall forecasting error for most of the flows. This study also showed other interesting trends, such as: the three nodes with more overall traffic in the network have more than 50% of the whole traffic in the network; the top five network flows are 30% of the total traffic in the network, which means that they play an important role in the network performance; the proportion of traffic between flows and nodes in the network is stable across time.

Nie, Jiang, Guo, *et al.* [71] present a deep belief network method to forecast and estimate the traffic matrix in a large-scale IP backbone network. The authors claim that in many state of the art network estimators the systems try to obtain a predictor of the traffic matrix at first, and then calculate an accurate estimator for the traffic matrix using the predictor. Firstly, a deep learning architecture is used to learn the properties of network traffic. Based on those properties, a deep belief network is used to forecast a traffic matrix. Using the deep belief network for forecasting, a network traffic estimation method is proposed, relying also on the link counts and routing information.

NeuTM is proposed by Azzouni and Pujolle [56]: an LSTM to forecast a large scale traffic matrix in an SDN. The tests were done using short-term forecasts (fifteen minutes) and results showed that

the mean squared error was much lower with the LSTM approach than with the traditional forecasting approaches (FeedForward ANN, Holt-Winters algorithm, ARMA model and ARAR algorithm).

In [72], different recurrent neural networks are evaluated with the goal of forecasting network traffic matrices. The compared networks are regular recurrent neural networks, LSTMs, Gated Recurrent Units (GRUs) and Identity Recurrent Units (IRNNs). A GRU is a type of recurrent neural network with the aim of reducing the computational cost of LSTM [73]. An IRNN is a recurrent neural network with the weight matrix initialized with an identity matrix. It is proved that, with a specific initialization, IRNNs can achieve similar performance to LSTM networks [74]. The results showed that the performance of both GRU and IRNN networks is comparable to the performance of the LSTM. Moreover, the GRU computational cost is similar to the computational cost of the LSTM networks.

3.3 Network Traffic classification

Network traffic classification can help the network operators to know what type of services are being used in the network, so they can react quickly in support of their various business goals. It can be used to detect patterns indicating an attack, automate the reallocation of network resources for premium customers to provide higher QoS or identify the customer use of the services provided by the network (for example, useful in case of an issued warrant to intercept information about a "person of interest"). Most QoS control mechanisms have a traffic classification module to prioritize different applications across limited bandwidth [75]. Much information about a current network flow can be learned and anticipated just by knowing its network service.

Initial traffic classification systems for network flows were based on port numbers. Successive IP packets having the same protocol type, source address:port and destination address:port were considered as belonging to the same flow. Most applications use well-known port numbers, and they could be identified by those numbers. However, applications are not forced to use the well-known port numbers, which means that they can easily trick the system by changing the port number. A study in 2006 by Madhukar and Williamson [76] showed that port-based analysis was unable to identify from 30% to 70% of the internet traffic flows investigated.

The next generation of network traffic classification methods was Deep Packet Inspection (DPI). DPI consisted of looking inside the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) payload to find application-specific data. In a study made by Sen, Spatscheck, and Wang [77], it was showed that deep packet inspection for classification of P2P traffic could identify the P2P application with less than 5% false positive and false negative ratio, by examination of only the first packets in a P2P connection. However, this method has been decreasing in its usage because of various factors. In a large-scale network, it can be impractical or even impossible to look through every payload of the packets in real-time, due to the complexity of the process. The user can easily encrypt the packet contents, forcing the system to fail in his search through application-specific data. Besides, privacy regulations from the governments have restricted the ability of third parties to inspect the payload of a packet. Finally, since the system relies on a knowledge-based approach, new attacks (zero-day traffic) could not be detected, and the knowledge-based system must be periodically updated to contain the most recent attack signatures. In a study made by Kim, Claffy, Fomenkov, *et al.* [78], it was showed that zero-day traffic is the major portion of unrecognized data, making up to 60% of flows in a network traffic dataset.

Nowadays, newer approaches to traffic classification without needing to rely on the packet payload are being studied. Machine learning techniques have been applied to recognize statistical patterns in the externally observable attribute of the traffic (such as the packet size, the time between arrivals

and other metrics). The goal is to cluster traffic flows into similar groups [79]. The surveys made by Nguyen and Armitage [79] and Deebalakshmi and Jyothi [80] show that traffic classification based on machine learning is still an open field of research with great interest.

Various machine learning techniques can be used to approach the traffic classification problem. It can be used a supervised learning approach, where the training data is already labeled and separated into clusters, and the machine learning model learns to classify traffic according to the data in each set. This approach is beneficial if the classification is between known classes or applications, but it requires a pre-processing step to label the data. For this model to be successful, it is important that the training data classes are similar to the real network traffic classes. The big drawback of this approach is the failure in recognizing zero-day traffic, lowering the algorithm accuracy. Unsupervised learning is another approach, where the network traffic is automatically clustered by the machine learning algorithm, without any prior guidance, eliminating the need for pre-processing the data and outlying the zero-day traffic. One disadvantage of this approach is that the created clusters will still need to be labeled to be relevant for analysis. Another issue with unsupervised learning is that the clusters do not necessarily map 1:1 to the applications we want to identify [79]. Recently, semi-supervised learning techniques have been used to classify network traffic. Semi-supervised learning techniques combine supervised and unsupervised learning to recognize zero-day traffic and label known traffic.

In the next subsections, supervised learning approaches to network classification will be presented, followed by unsupervised learning approaches and, finally, semi-supervised learning approaches.

3.3.1 Supervised learning approaches

Amaral, Dinis, Pinto, *et al.* [81] proposed a simple architecture for data collection in SDNs. A traffic classification use case is tested, in which several applications' (Bittorrent, Dropbox, Facebook, HTTP, Linkedin, Skype, Vimeo and Youtube) traffic is classified using several ensemble learning classifiers: Random Forests, Stochastic Gradient Boosting and Extreme Gradient Boosting. All classifiers presented a high accuracy rate. The study shows that it is possible to classify accurately applications based only on network traffic control data.

In the work by Lu, Huang, Lin, *et al.* [82], three different classifiers with different design objectives were proposed to make traffic classification, using only the message size sequence and message distribution as features: the Message Size Distribution Classifier (MSDC), the Message Size Sequence Classifier (MSSC) and a hybrid solution. The MSDC has an offline and an online phase. The offline phase collects a set of network flows and extracts the flow information and the packet size distribution for all flows. The online phase compares the online flows with the offline labeled flows and classifies them into an application with the minimum similarity distance. At the same time, the session grouping stage tries to cluster the flows based on port locality. After this stage, each flow is classified as an application, and flows with adjacent ports are classified into the same session. If a flow is classified into the same session but to a different application, there is an additional stage, the application arbitration stage, that has the goal of resolving conflicts of flow classification. The drawback of this classifier is that while its accuracy is 99.98% by inspecting less than 300 packets, its throughput is only of 400Mbps, making it hard to operate in real-time in a network.

The MSSC is a lightweight classifier designed for real-time classification. It is also composed of offline training and online classification. In offline training, the message size sequence of each flow is stored. In online classification, online flows are compared with each message size sequence of the captured flows, and the classification of flows as belonging to each class is done with the loose longest common subsequence algorithm, instead of the sum of euclidean-like distance, as proposed in MSDC. The resulting classifier can quickly make a decision on flow classification, having an accuracy of 99.94%

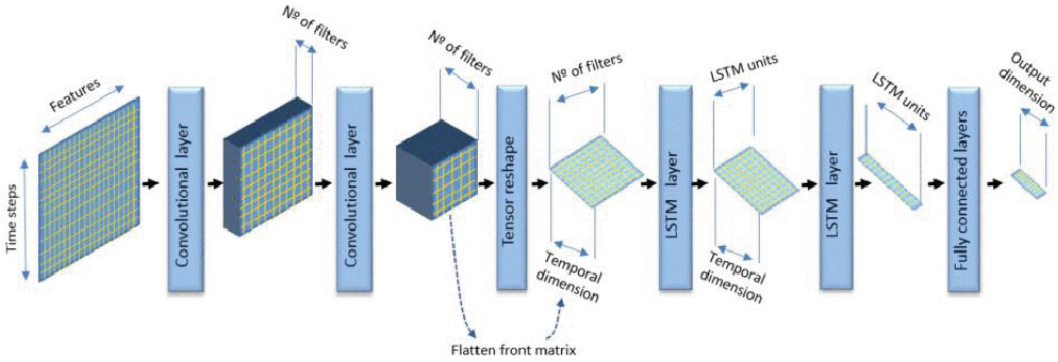

Figure 3.2: Combination of convolutional layers and LSTM layers in the proposed architecture in [85].

by looking only at the first 15 packets, having a throughput of 800 Mbps. Incorrect classifications may happen due to incomplete packet captured or similar protocol states between different applications. A hybrid solution was also proposed, using both classifiers in parallel. If the MSSC is not able to make a decision in a short time that matches more than 90% to an application, the decision is later made by MSDC. The hybrid solution proves itself to be a good balance, having a classification accuracy of 99.97% and an overall system throughput of 723 Mbps.

In the work of Ducange, Mannara, Marcelloni, *et al.* [83], the goal is to classify internet traffic, but also to create a classifier with high interpretability to a human. To do that, a novel approach using Multi-Objective Evolutionary Fuzzy Classifiers (MOEFCs) and fuzzy rule-based classifiers is taken. The algorithm PAES-RCS is used to design the rule base and tune the parameters of the fuzzy sets. The results show that, besides the model low complexity, the accuracy is satisfactory and this approach achieves high interpretability.

The work by Ertam and Avcı [84] uses a kernel-based extreme learning machine to classify internet traffic. The system GA-WK-ELM reaches an accuracy rate of over 95%, using a wavelet activation function and choosing its parameters with a genetic algorithm. The system has a quick training and testing time, which is important for a real-time scenario.

Lopez-Martin, Carro, Sanchez-Esguevillas, *et al.* [85] proposed a network traffic classifier composed of LSTMs and convolutional layers. Various architectures were tested, using only convolutional or LSTM layers, and the architecture with the best results had two layers of both types: two convolutional layers, followed by a Tensor reshape and two LSTM layers; the last layer was fully connected (Figure 3.2). The dataset used to evaluate the performance of the classifier is from a Spanish academic and research backbone network and contains 108 distinct labeled services with a highly unbalanced frequency distribution. To train and evaluate the model, the assignment of the ground-truth to each flow was done with nDPI tool[1], that applies deep packet inspection techniques to classify the flows. From each packet, six features were derived (source port, destination port, packet direction, bytes in payload, interarrival time and window size), and each flow is identified using only the first 20 packets in a time-series fashion. The matrix formed by the time-series of feature vectors is treated as having the same locality properties as an image, to be inputted to the convolutional layer. The final results showed an overall accuracy higher than 96% for all classes.

3.3.2 Unsupervised learning approaches

McGregor, Hall, Lorier, *et al.* [86] published one of the earliest works in traffic classification with unsupervised learning, using the Expectation Maximization (EM) algorithm. The work tries to cluster

[1] `https://www.ntop.org/products/deep-packet-inspection/ndpi/`

network traffic from various applications (HTTP, FTP, SMTP, IMAP, NTP and DNS). The EM algorithm groups the traffic flows into small clusters. Features that do not have a large impact on the clusterization are identified and removed, and the process is repeated. The work showed that the approach could successfully cluster different types of traffic based on various parameters (traffic length, inter-arrival statistics, idle time and others).

Zander, Nguyen, and Armitage [87] showed that the unsupervised learning algorithm AutoClass [88], an unsupervised Bayesian classifier, could reach an accuracy of 86.5% in the clustering of network traffic applications. Some applications were hard to identify using only network traffic data. Similar work was done by Erman, Arlitt, and Mahanti [89], where three clustering algorithms were tested to classify network traffic: AutoClass, K-means and DBSCAN. The work shows that all three algorithms create accurate clusters with the data provided, but all have different characteristics. The AutoClass algorithm produces the best results, but is much slower than the other two. While the DBSCAN has the worst accuracy, it creates a small number of clusters, which can be important in the flow analysis. The K-means algorithm has a marginally lower accuracy than the AutoClass algorithm, but it has a faster model building time. This study was only possible because of the small data set size, otherwise it would have been expensive to manually analyze and label all the clusters.

In the paper work of Fan and Liu [90], a supervised (Support Vector Machine (SVM)) and an unsupervised (K-means) learning approach were compared to classify network traffic. The dataset was composed of traces from a research campus in a 24-hour period, and ten broad traffic categories were used to label the dataset (HTTP, mail, bulk traffic, services, P2P, database, multimedia, attack, interactive traffic and games). The supervised approach had a higher overall accuracy, as expected (98%). Zero-day traffic was not included in the dataset, which could lower the accuracy of the supervised method and increase the accuracy of the K-means algorithm.

In 2017, Hochst, Baumgartner, Hollick, *et al.* [91] used a neural autoencoder to classify traffic flows from an office network. A neural autoencoder is an ANN trained to reconstruct an original input vector from a smaller representation. It is widely used for dimension reduction. The traffic flow type was not separated by application, because current applications have HTTP(S) traffic as a basis for many applications other than browsing the web. Instead, the extracted classes were divided into browsing, interactive, download, live stream, video stream, call and upload. A novel time-interval based feature vector construction was also approached. The flows were separated by exponentially growing time periods because it was observed that the most important criteria for clustering was available shortly after the beginning of the flow. The obtained results showed that seven different classes of mobile traffic flows are detected with an average precision of 80%.

3.3.3 Semi-supervised learning approaches

In recent works, the challenge is not only to classify network traffic, but to handle zero-day traffic and incorporate it into the model for future learning using semi-supervised learning. SeLeCT (Self-Learning Classifier for internet Traffic) [92] is a classifier that can detect and cluster internet traffic, including zero-day traffic, using semi-supervised learning methods. The authors state that the goal is to provide deeper network visibility to operators, exposing traffic classes that are very specific and possibly not already known by the operator. The main control loop of SeLeCT starts by collecting unlabeled traffic from the network and applying the K-means algorithm to cluster the flows. An iterative clustering approach is taken, filtering on each iteration the clusters that have flows with the same server port. Those clusters are removed from the next iteration, and the clustering and filtering phase is repeated for a number of iterations to improve the homogeneity of the clusters and to reduce its number.

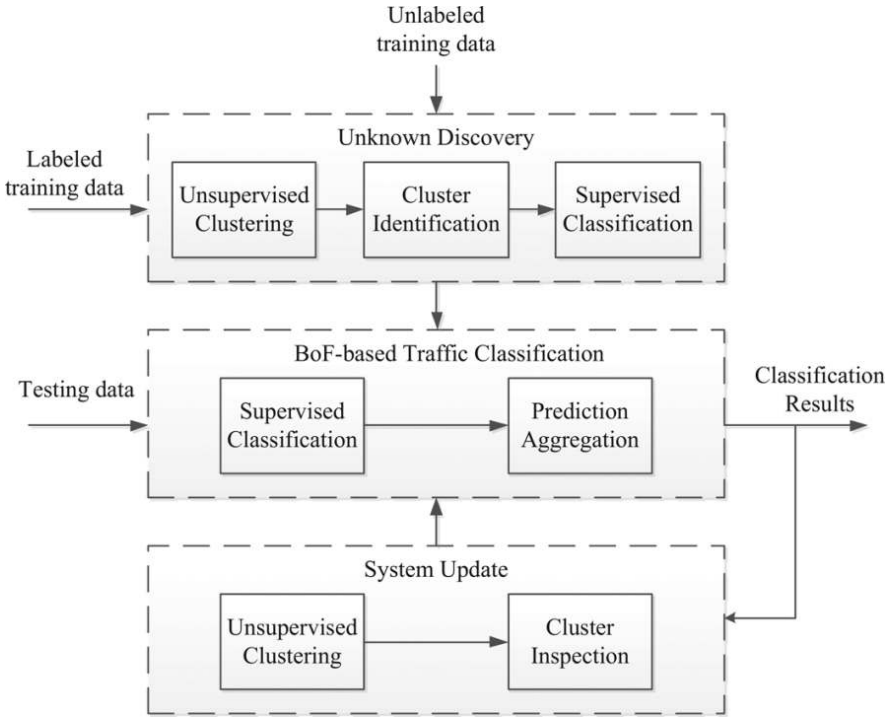

Figure 3.3: RTC framework [75].

The next phase applies the K-means algorithm to the other flows that do not have the same server port. Then, a labeling algorithm is applied to the formed clusters to identify them, using a simple majority voting scheme. If all flows are labeled as "unknown" in a cluster, the system administrator should manually label the cluster. Finally, from each cluster, a number of samples are taken to feed the next iteration of the main loop with labeled data. From each cluster, the number of extracted samples is proportional to the cluster size, ensuring that at least one sample of each cluster is chosen. The chosen feature set was the server port, the length of the first n segments with payload and their corresponding inter-arrival time. Such simple features can be acquired at the beginning of the flow and the system can run in real-time, constantly monitoring the traffic by creating batches of flows. The results show that SeLeCT overall accuracy is higher than 98.8%, much better than the K-means overall accuracy. Besides that, the number of clusters is often reduced to less than 150. More interesting is the fact that SeLeCT was able to correctly classify samples that were initially wrongly labeled by a deep packet inspection tool in the selected dataset.

Zhang, Chen, Xiang, *et al.* [75] also propose a system to tackle the problem of detecting zero-day applications in a network combining supervised and unsupervised learning methods. The RTC framework (Figure 3.3) has three main layers: the unknown discovery, the Bag of Flows (BoF)-based traffic classification and the system update.

The goal of the unknown discovery layer is to find new samples of zero-day traffic in a set of unlabeled network traffic collected from the target network. It is composed of three inner modules: Unsupervised Clustering, Cluster Identification and Supervised Classification. The labeled and unlabeled training data are initially clustered by the Unsupervised Clustering module, using the K-means algorithm. The Cluster Identification module will identify each cluster based on the labeled flows that each cluster contains. The authors claim that, if the number of clusters chosen is high enough, it can be obtained high-purity traffic clusters. If a cluster does not contain any labeled samples, it is classified as a zero-day traffic cluster. Because of the high number of clusters created, the Unsupervised Clustering

module will lead to a high true positive and true negative rate of unknown detection and it cannot be used for zero-day traffic classification. The Supervised Classification module takes advantage of the Random Forest algorithm to classify all the known classes and the unknown class with the labeled data from the Cluster Identification module.

The aim of the BoF-based traffic classification is to take the prelabeled training samples and zero-day traffic samples to build a classifier for robust traffic classification. To do that, a classification method using flow correlation is employed. The System Update layer analyzes the zero-day traffic and constructs new classes to extend the system's knowledge. It starts by applying unsupervised clustering using the K-means algorithm to the zero-day traffic. Then, it randomly selects a number of flows from each cluster and a user manually inspects the selected flows. If all the selected flows indicate the same application/protocol, it creates a new class and uses the flows in the cluster as its training data. With the System Update layer, the system is able to learn new classes of traffic. The results of this approach showed that the RTC system performs much better than other state of the art algorithms, like SVC, Random Forest or BoF with Random Forest.

Zhao, Zhang, and Chang [93] also use a semi-supervised approach to classify network traffic, with a tri-training framework. A three-layered architecture is designed to solve the problem. The first layer is the clustering stage. It consists of two K-means algorithms, but with different parameters and approaches. One of them focuses on finding clusters of unknown flows. The other one has the goal of finding clusters that can be labeled with a class. The next layer is the tri-training classifier. This layer trains three weak classifiers: C4.5, Naive Bayes and Random Forest. From the labeled clusters formed in the previous stage, the classifiers will be trained to recognize the different classes of traffic. For every training sample of the unknown clusters, if two classifiers classify them as belonging to a class, the third classifier will be trained with the unlabeled data, now labeled by the other two classifiers. The process will repeat itself for every classifier until there is no more unlabeled data that two classifiers agree on the same label. When compared with other classification algorithms, the authors show that this approach has better accuracy. However, it cannot classify zero-day traffic accurately, since there must be at least one training example for a class to be created.

3.4 Prediction of service-level agreement breaches

A Service-Level Agreement (SLA) is an agreement between a service provider and one or more customers that states the service standards that the service providers must enforce, and the obligations that the clients have with the service provider. Typically, an SLA defines the responsibilities of the service provider according to a set of measurable conditions that define the delivered service, called Service-Level Objectives (SLOs), like the availability uptime, minimum performance benchmarks and maximum resolution time for a problem. The responsibilities of the clients can be, for example, a monthly download limit. If at some point, any part of the contract does not comply with the SLOs, penalties can be applied, regularly involving extra payments and indemnifications.

An iterated SLO violation can have side effects in a service provider, mainly in terms of loss of revenue and reputation. However, it is hard for service providers to predict and prevent an SLO violation before it happens, due to the network complexity and the data available.

There is not much research in prediction of SLA breaches in networks. In the work by Hani, Paputungan, and Hassan [94], a support vector regression is used to predict SLOs violations. The SLOs addressed were throughput and response time. The prediction accuracy in the dataset used was more than 80%. This work handles the SLO violation represented by numerical static thresholds, turning the problem into a regression problem.

Ahmed, Johnsson, Yanggratoke, *et al.* [95] analyze a video-on-demand scenario, where the goal is to predict SLA breaches on the video transmission between a server and a client, using only server device statistics, such as CPU utilization, free memory and other metrics, and the classification is given by the client, using frames per second as a threshold. The scenario is enhanced with concept drift, which means that the SLAs and the system behavior could change over time and the predictor has to learn and adapt to it. Three online algorithms were tested: Gradient Ascend Logistic Regression, Very Fast Decision Tree and Online Accuracy Updated Ensemble. Three offline algorithms were also tested to serve as a baseline: Logistic Regression, Decision Tree and Random Forest. The results show that for a regular traffic scenario, the performance of the offline and online classifiers is similar, with an accuracy of over 90% in the prediction of violation in SLAs. In the scenario of a flash crowd, the online methods maintained consistently high performance, especially Online Accuracy Updated Ensemble, while the accuracy of the offline methods dropped abruptly.

Bendriss, Yahia, and Zeghlache [96] present a framework for SLA enforcement for VNFs and SDNs. The most important block of the architecture, the Cognitive Smart Engine, is composed of three blocks: Forecasting, Violation Prediction and SLA enforcer. The forecasting block has as input the network features at time t, and it will predict the same features at time t+x. In this work, the prediction for each feature was made using an LSTM for each feature. Those feature predictions will be sent to the violation prediction block, to check if the predictions violate any current SLO. To do so, this block has access to the low-level repository of SLO specified by the network administrator. The algorithm to classify the SLA violations is the ID3 (Iterative Dichotomiser), a decision tree learning algorithm. The advantage of a decision tree classifier is that the internal decisions can be easily seen as a white box. Different SLO violations can be detected with a multi-class classifier. The SLA enforcer will receive the classification of violation predictions and act accordingly in the network. In this study, the SLA enforcer problem was not addressed. While the study shows a good performance, the evaluation is only made separately to the forecasting and the violation prediction block. In a real-world scenario, it would be complex to fetch the low-level SLO for all the flows and predict the SLO violations.

3.5 Conclusion

This chapter presented the recent advances in the autonomous management of networks using Machine Learning, focusing on four problems: network traffic forecasting, traffic matrix forecasting, network traffic classification and prediction of service-level agreement breaches. There is currently active research in the four areas, and there are many approaches being studied for each one of them.

Due to the wide range of approaches for each problem, in this dissertation it will only be explored the network traffic forecasting problem as a way of performing active monitoring of the network. In many of the works presented, the dataset used was from local networks. In this dissertation, the dataset used will be from operator networks, a vehicular and a mobile operator, with more network traffic and different network patterns. Those patterns can be useful not only for the monitoring of the network, but also for the management of a city.

Several techniques presented in Section 3.1 will be implemented, such as linear forecasting approaches (ARIMA) and non-linear forecasting approaches (ANN, LSTM, etc.). To achieve better results, combinations of neural networks and ensemble algorithms will also be tested. It will be performed short-term forecasts, with one hour as lead time to forecast.

In the next three chapters, it is described the process of implementing, testing and tuning a model for the prediction of network KPIs. In the Chapter 4, the KPI is the number of sessions per hour in a vehicular network; in the Chapter 5, the same KPI is used (number of sessions per hour) in the same

network, but with different prediction objectives (predict accurately higher value observations and predict accurately in anomalous time intervals); in the Chapter 6, the KPI is the maximum number of users per hour in a 4G network.

4

Forecasting of the number of sessions in a vehicular network

Recently, free public internet access has been an upward trend in public transportations of developed cities. In Porto, in the largest mesh network of connected vehicles in the world, more than 200 000 people enjoy free Wi-Fi every day in buses, taxis and municipal service vehicles [97].

In a 5G network, a slice for this mesh network may be needed, for dedicated monitoring and provisioning. In this chapter, it will be described the process of implementation, testing and tuning of a forecasting model for the number of user sessions in this mesh network. A user session is counted from the moment that a User Equipment (UE) has access to the internet until its disassociation. The number of user sessions is the metric chosen to forecast because it can be used as a rough estimate of network traffic volume.

The first three sections of this chapter are dedicated to data analysis. In the first section, it is described the vehicular dataset; in the second section, it is performed deeper time-series analysis of the data; in the third section, the original data is transformed and time-series analysis techniques are applied to understand what data transformation is more adequate to use in the forecasting task. Data exploration is essential for any machine learning project: there could be valuable insights into the data that can be used to build a better machine learning solution. Besides that, data preparation must be done to remove the outliers and ease the learning process.

The fourth section focuses on the implementation, testing and tuning of some forecasting techniques (described in Section 2.4). Firstly, a benchmark result is obtained with the persistence model. Then, the ARIMA model is tested, followed by several machine learning techniques, such as classical machine learning algorithms or different types of neural networks. Combinations of the models are also tested. At the end of the fourth section, the models with the best results are used to compute the final error in the test set.

4.1 Dataset exploration

The dataset from the mesh network of connected vehicles (vehicular network dataset) contains the sessions and its durations, between the period of 2018-07-09 at 08:03:17 to 2019-02-04 at 10:53:43 (in the ISO 8601 format, YYYY-MM-DD at hh:mm:ss). There are 3 014 264 sessions. A session is characterized by a user login in the mesh network. An analysis of the data shows that there are sessions with the wrong duration. Some durations are negative, and other durations are very large (more than 17808 days, possibly due to miscalculations, because the start date of those sessions is

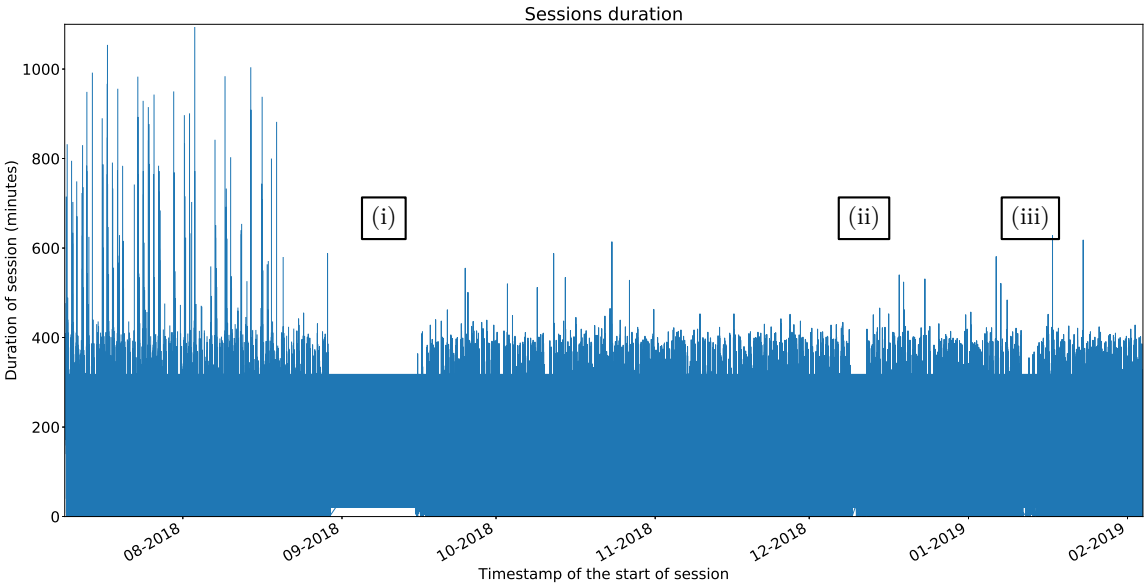

Figure 4.1: Sessions' duration over time, in minutes. The three markers show the periods where there were no sessions in the vehicular dataset.

around Unix time 0, the 1^{st} of January of 1970). There are only 17 sessions with a very long duration, and the sessions with negative duration are 16 073, which represents 0.53% of the total sessions. After the removal of these outlier sessions, the total number of sessions is 2 998 174 (99.46% of the original sessions).

In the Figure 4.1 it is plotted the sessions' duration over time, in minutes (without the outlier sessions). From the 29^{th} of August to the 16^{th} of September there are no sessions (i). The lack of sessions also happens in the periods from the 9^{th} of December to the 12^{th} of December (ii) and from the 11^{th} of January to the 14^{th} of January (iii), due to system maintenance. There was a change in the way that the session duration was calculated before and after the period from the 29^{th} of August to the 16^{th} of September, represented by a visibly different distribution of the sessions' duration.

A closer look at the sessions' duration (Figure 4.2) shows that it is highly variable between sessions. The mean of a session duration is 16 minutes and 54 seconds, and the standard deviation is 28 minutes and 32 seconds.

To uniformize the time interval between the number of sessions in the dataset (the forecasts will be done for a fixed time interval of one hour), it is needed to group the number of sessions per hour (Figure 4.3). There is an outlier on the 15^{th} of August, where the number of sessions per hour exceeds the 10 000 sessions (the maximum number of sessions in the figure is limited for readability purposes; this event is marked with (i)). After the 21^{st} of December (ii), the number of sessions is also significantly different from the previous data, which is expected due to the holiday season, that has much impact on the city transports' due to the university students' and workers' different day-to-day behaviors.

The data used for forecasting must be continuous (which means that there must be no "jumps" in time intervals for the forecasting to work correctly based on the previous values) and must follow a pattern (outliers should be eliminated), which means that the anomalous days in December should not be used for forecasting, neither the data when the number of sessions is zero. The data used for forecasting was the data between the 17^{th} of September to the 8^{th} of December (Figure 4.4).

There are various patterns present in the data. Figure 4.5 shows data from one week. The weekend has significantly fewer sessions than weekdays. In a day, there are two major peaks in the number

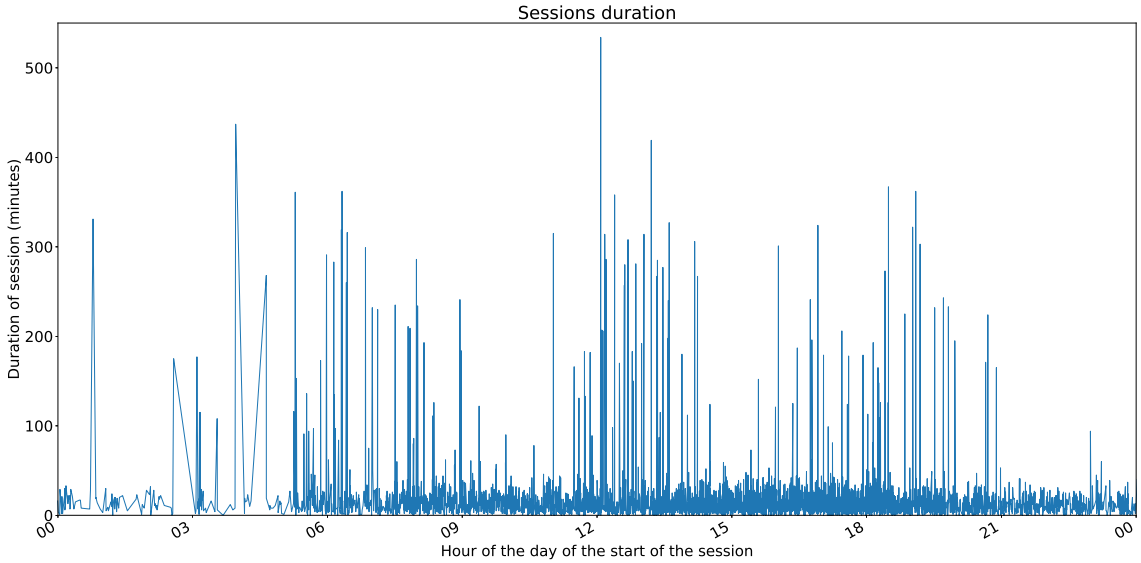

Figure 4.2: Sessions' duration in minutes in the day of 14^{th} of Octoboer of 2018 in the vehicular network.

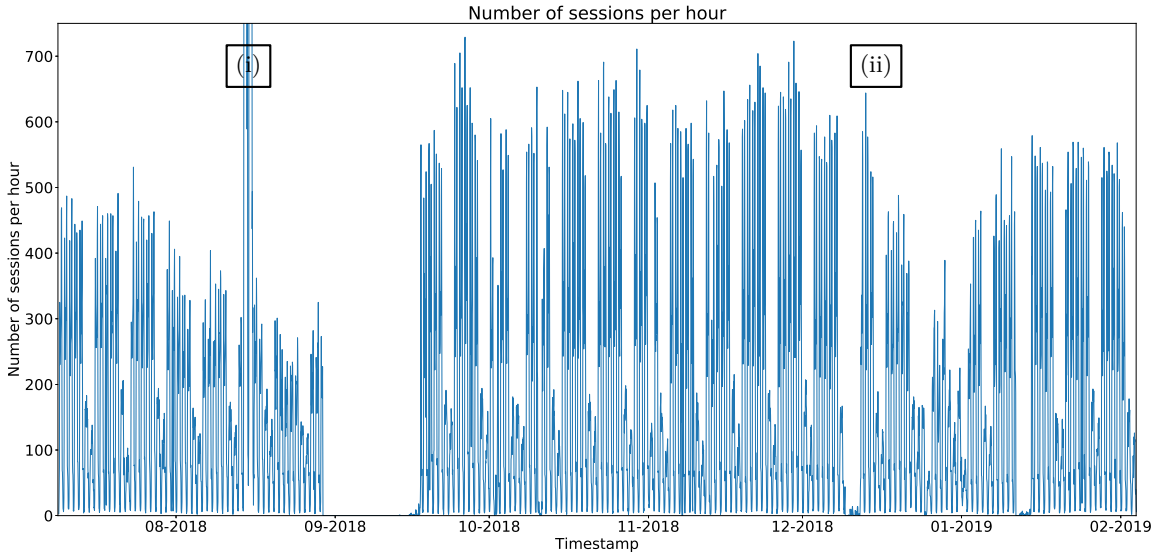

Figure 4.3: Number of sessions per hour in the vehicular network.

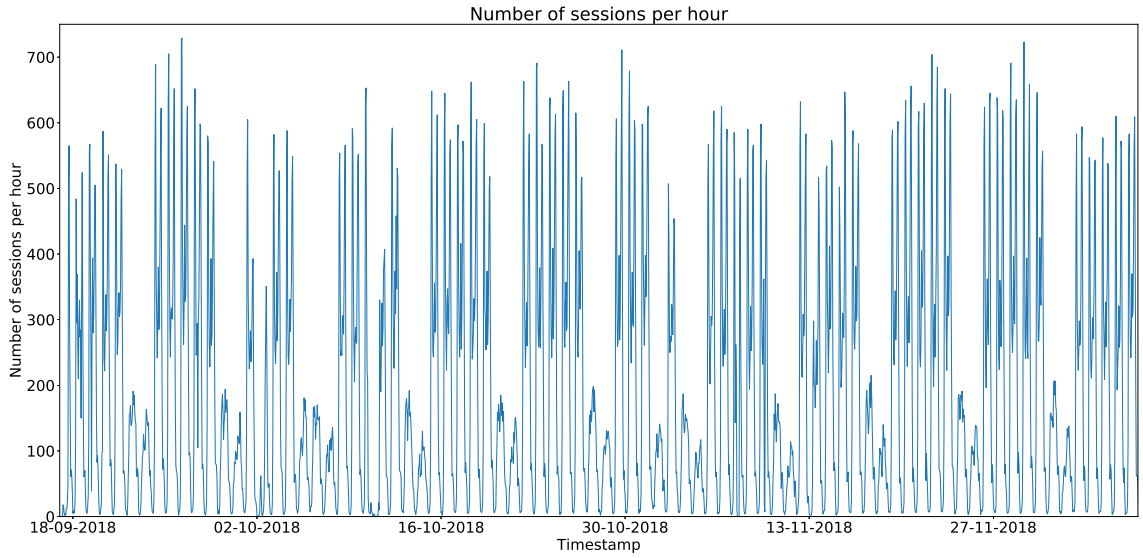

Figure 4.4: Number of sessions per hour limited to the data used for forecasting, in the vehicular network.

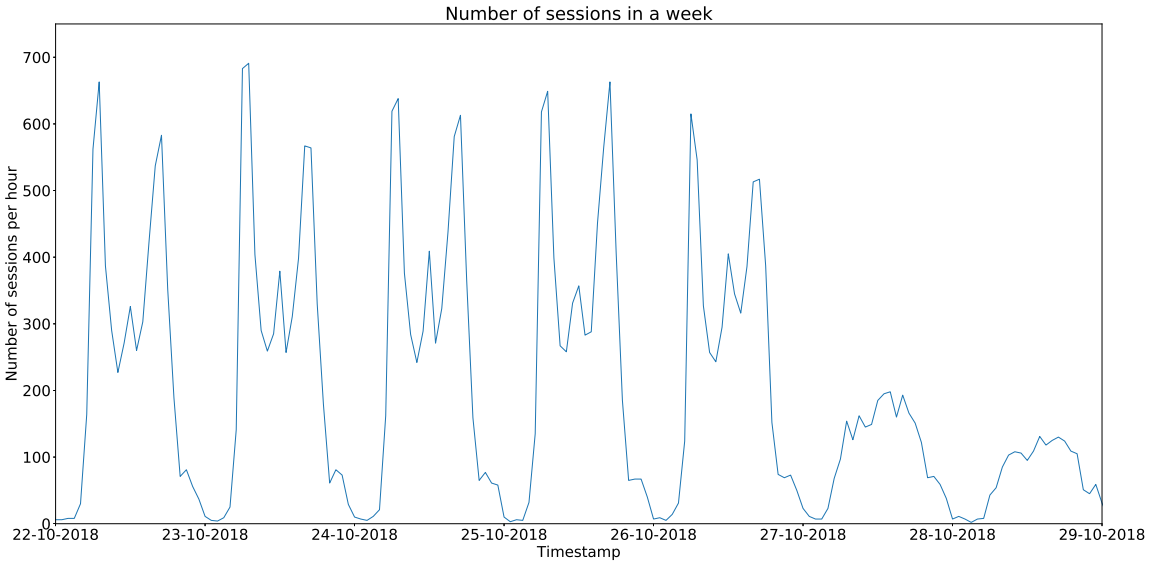

Figure 4.5: Number of sessions per hour of one week in the vehicular network.

of sessions: the first is around 7h00, the second is around 17h00, which is understandable due to the workers' and students' schedules. There is a third minor peak around 12:00, at lunch time. Another relevant observation can be done by observing Figure 4.6, where the 1^{st} of November has a different pattern from the other weekdays, due to being a holiday (i). There is another holiday on the observations, in the 5^{th} of October (not visible in this figure).

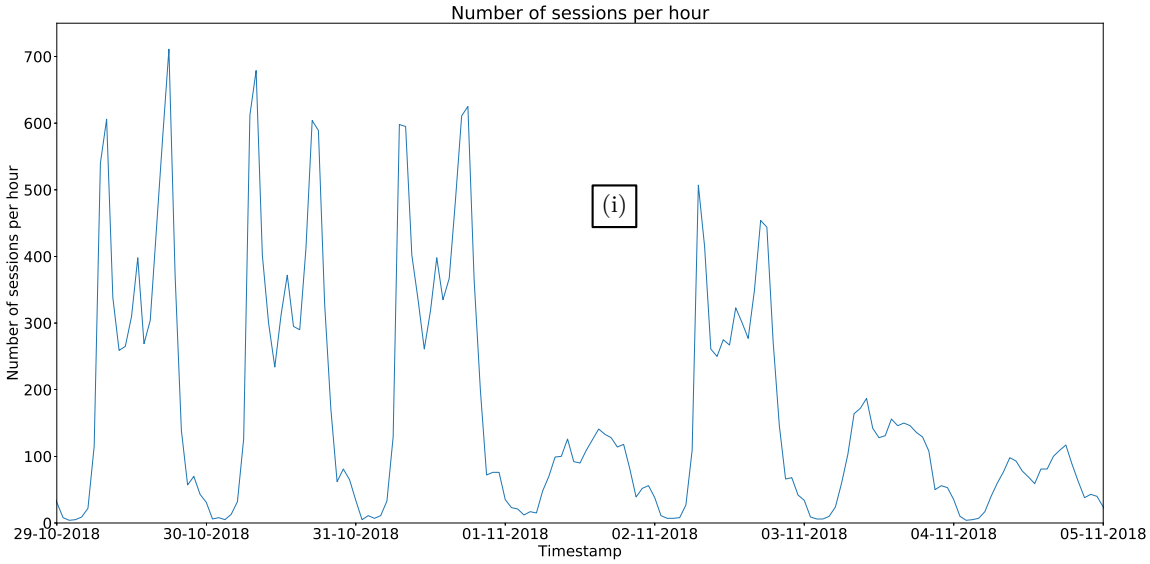

Figure 4.6: Number of sessions per hour of a week with a national holiday in the 1st of November (i) in the vehicular network.

4.2 Time-series analysis

A time series analysis is needed to better understand the characteristics of the data. Particularly, it is needed to approximate the time-series data to a stationary process. A stationary process is a stochastic process whose conditional joint probability distribution does not change when shifted in time [98]. A stationary process can be represented by Equation 4.1, where the function F is periodical and its value is not affected by the time shift τ.

$$F_x(x_{t_1+\tau}, ..., x_{t_n+\tau}) = F_x(x_{t_1}, ..., x_{t_n}) \tag{4.1}$$

Stationary processes are easier to model, especially for linear forecasting models such as ARIMA. If the data is not directly represented by a stationary process, various types of differentiation can be done to approximate a non-stationary process to a stationary process (Section 4.3).

If the data is stationary, its mean is stable across a fixed time-interval and the standard variation is close to zero. In Figure 4.7, the rolling mean and the standard variation of the sessions per hour are represented in a plot, using a rolling window of 24 hours. The mean and the standard deviation are variable over time mostly due to the different range of values presented in the weekends from the week values. The overall mean of the number of sessions is 181.76 and the standard deviation is 180.98.

A test that can be used to detect temporal patterns in time-series data is to calculate the autocorrelation function. The autocorrelation function calculates the Pearson correlation coefficient [99] between the same data distribution, separated by various time lags. The Pearson correlation coefficient can be calculated according to equation 4.2, where ρ is the Pearson correlation coefficient, X and Y are the data distributions, cov is the covariance function and σ_X is the standard deviation of X.

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma Y} \tag{4.2}$$

In Figure 4.8, it can be seen the autocorrelation plot of the time-series data. There are peaks in the plot around the lags that are multiple of 24 hours, indicating that there is a high correlation

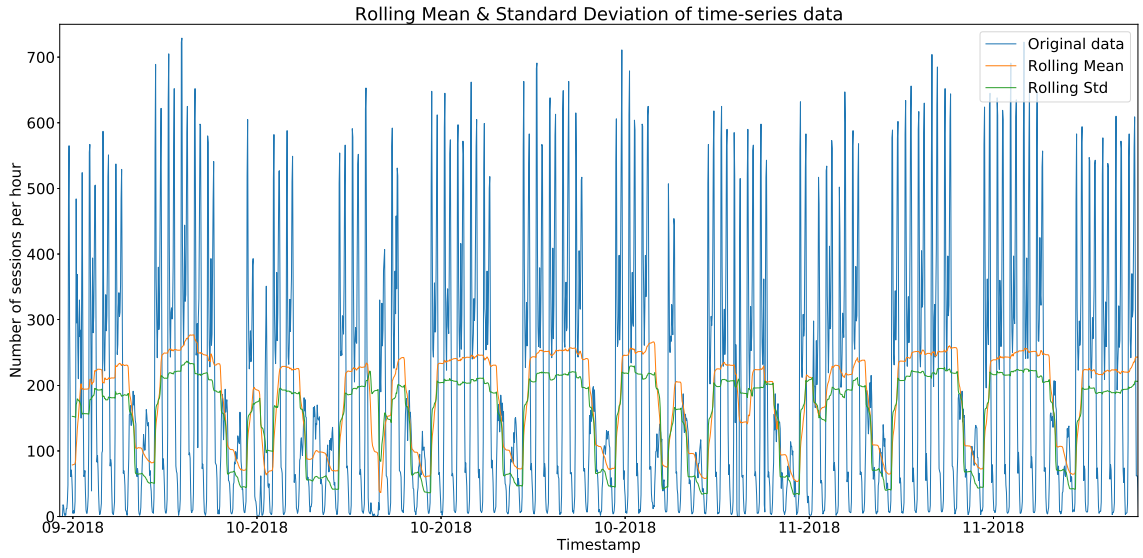

Figure 4.7: Number of sessions per hour over time, with the rolling mean and rolling standard deviation represented, with a rolling window of 24 hours, in the vehicular network.

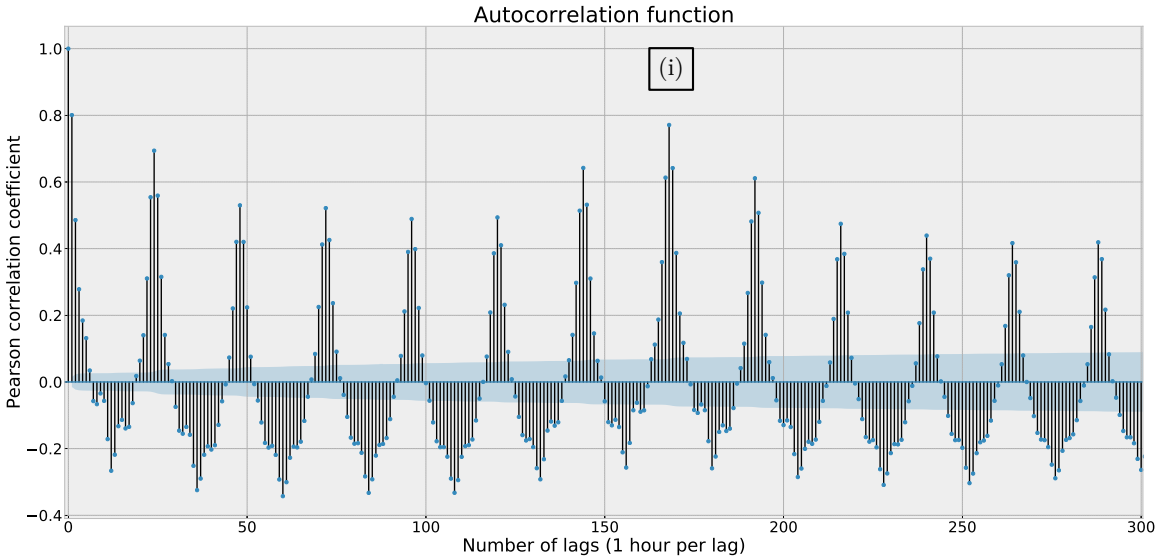

Figure 4.8: Autocorrelation plot of the time series data in the vehicular network.

with the lag of 24 hours (one day). The highest peak (i) has a lag of 168 hours (one week). These observations help to detect patterns in the data: the peaks in the autocorrelation function indicate similarity with the time interval of the number of lags. The number of sessions is correlated with the number of sessions in the previous days at the same hour and in the previous week.

A seasonal decomposition using moving averages was done using an additive model [100] with a frequency of one week. The original data is divided into three components: trend, seasonality and residuals (Figure 4.9). There is no clear trend in the data, but there are daily and weekly seasonality components, confirming the results of the autocorrelation plot. The trend or seasonality component can be removed using differentiation techniques (Section 4.3), to try to model only the residuals, for easier and accurate forecasts.

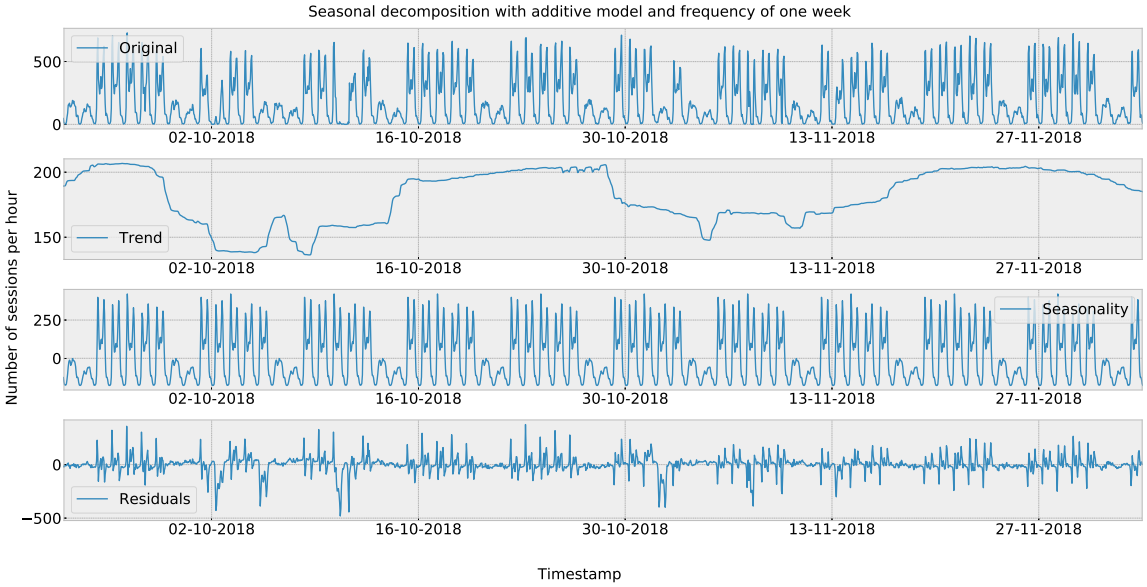

Figure 4.9: Seasonal decomposition using moving averages with an additive model and frequency of one week in the vehicular network. The first plot shows the original data; the second shows the captured trend; the third plot shows the seasonality; the last plot shows the residuals.

The Augmented Dickey-Fuller test [101] is a statistical test that determines how strongly a time-series is defined by a trend, testing the null hypothesis that a unit root is present in an autoregressive model. The alternate hypothesis is that the time-series does not have a unit root, which means that it is not defined by a clear trend. With a p-value lower or equal than 5%, the null hypothesis is rejected, which means that the data does not have a unit root. Otherwise, the null hypothesis is failed to be rejected, and the data has a unit root. The p-value obtained for this dataset was $9.12 * 10^{-8}\%$ with a significance level of less than 1%, which confirms the observations of the seasonal decomposition plot that the time-series data has no clear trend.

Figure 4.10 represents the number of sessions per hour in consecutive hours. From 6:00 to 8:00 the number of sessions per hour has the highest leap of the day, going from around 100 sessions to around 600 (during weekdays, in the circle (i) in the figure). There is another rise in the number of sessions from 16:00 to 18:00 (during weekdays, in the circle (ii) in the figure). The major decay in the number of sessions of the day is represented by the cluster of points in the middle right of the plot (during weekdays, in the circle (iii)), ranging from the hours between 08:00 to 10:00 and between 18:00 to 20:00.

These observations can help to better understand the life habits of the Porto inhabitants. The high number of sessions from 6:00 to 8:00 and from 18:00 and 20:00 in Porto public transports happen due to the working schedule of the inhabitants. The minor peak between 12:00 and 13:00 (Figure 4.5) shows that most Porto inhabitants have their lunch break at that time interval, although many of them prefer not to use public transportations in the lunch break. Although Porto is a city with an increasing number of tourists every year, the majority of the people that use the public transports are Porto inhabitants that follow a working schedule. During the weekends, Porto inhabitants use fewer public transportations than on weekdays.

Figure 4.11 shows the histogram of the number of sessions. For the most hours, the number of sessions is below 100. There are also many hours with around 300 sessions and around 600 sessions. For a machine learning solution, it can be challenging to forecast the number of sessions if the observation

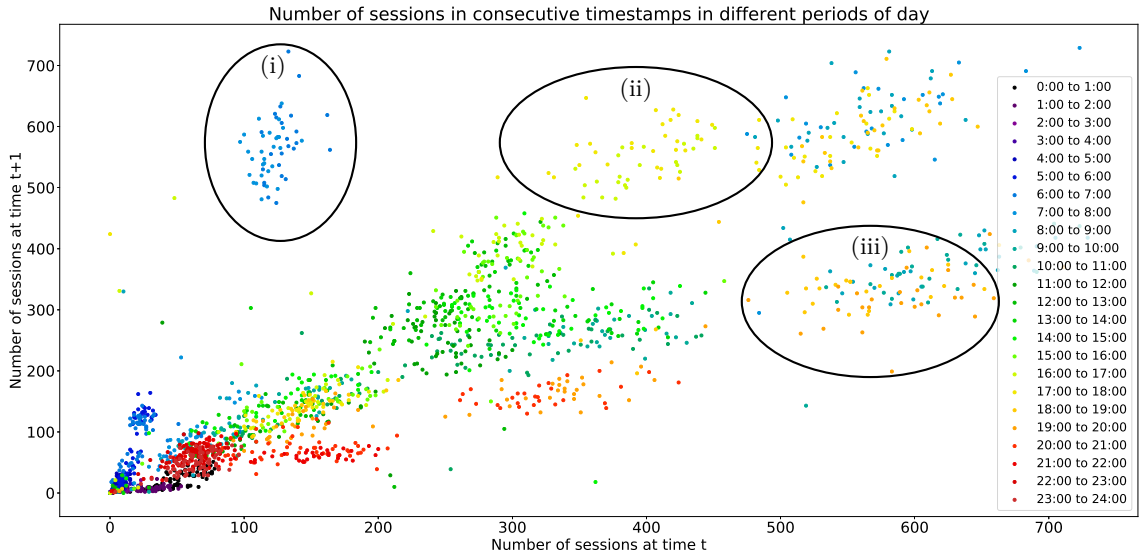

Figure 4.10: Number of sessions per hour in consecutive hours in the vehicular network.

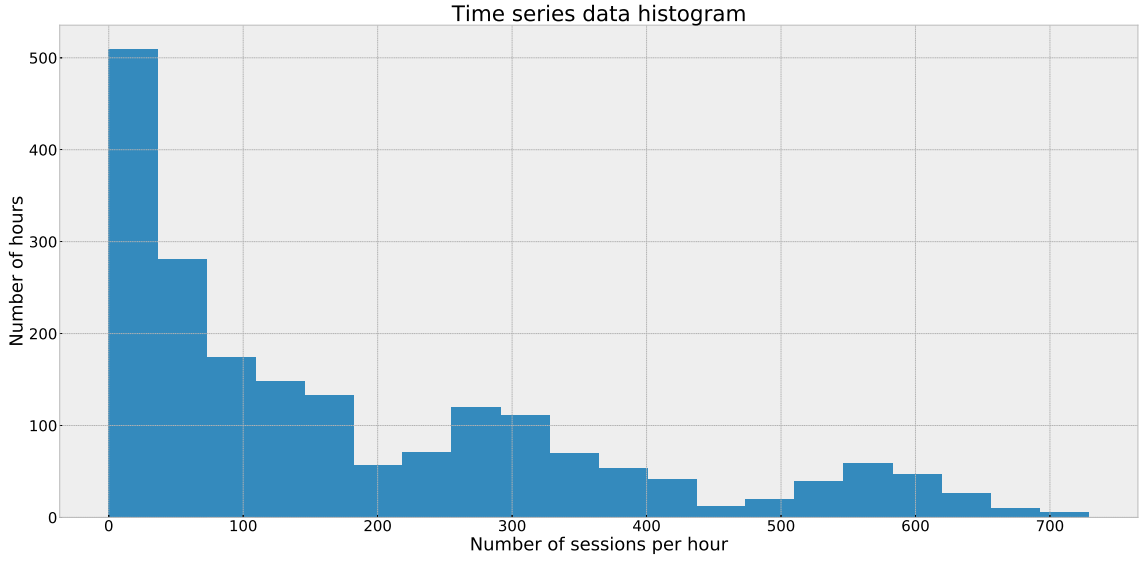

Figure 4.11: Time-series data histogram of the vehicular network.

value is higher than 100, due to the dataset having more observations where the number of sessions is between 0 and 100 (the dataset is biased towards a low number of sessions per hour). Besides forecasting the absolute value, another technique that will be used is differentiation: forecasting the variation of the number of sessions. In that way, the effect of an unbalanced dataset can be attenuated. In the next section, the differentiation transformations are applied to the data and analyzed with the same techniques used in this section.

4.3 Differentiation tests

Differentiation is achieved by transforming the time-series data into the difference of time-series values, lagged by a time interval, with the goal of stabilizing the mean of the time-series and eliminating trend and seasonality, making the data stationary. Differentiation is performed by subtracting two

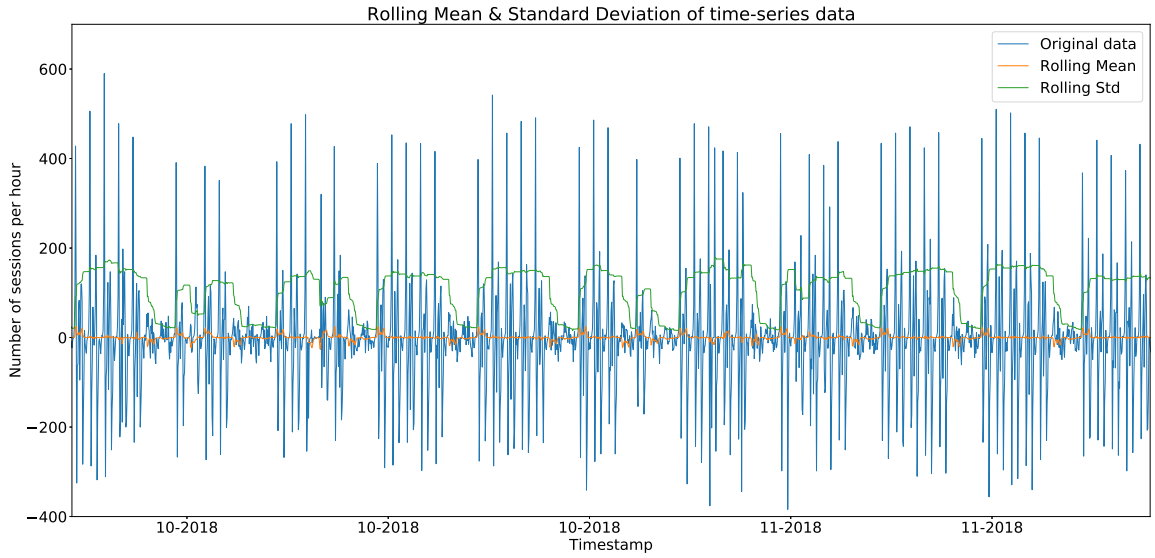

Figure 4.12: Time-series data of the vehicular network with differentiation of one lag, with the rolling mean and rolling standard deviation represented, with a rolling window of 24 hours.

observations, as explained in Equation 4.3.

$$difference(t) = observation(t) - observation(t - lag) \tag{4.3}$$

Three *lag* values will be tested in the differentiation: 1, 24 and 168. Those values were chosen because they represent one hour, one day and one week, respectively. In the autocorrelation function (Figure 4.8), those lags (1, 24 and its multiples, and 168) had higher Pearson correlation coefficient (168 is also multiple of 24, but it has higher Pearson correlation coefficient than any of the previous multiples), proving to be good candidates for the differentiation *lag*.

In Figure 4.12, it is represented the rolling mean and the standard deviation of the time-series data, with a rolling window of 24 hours, with the differentiation of one lag. It can be seen the same seasonality pattern that was present in the non-differentiated data (Figure 4.7). As expected, with this transformation the mean has stabilized around 0. Because the mean is calculated with a rolling window of 24 hours, and the transformation applied calculates the variation of the number of sessions in a day, if the number of sessions in an hour every day was the same, the variation of the previous 24 hours was zero. With the vehicular dataset, because most days are similar, the total variation over the last 24 hours is always close to zero.

Figure 4.13 represents the autocorrelation function of the data with a one-lag differentiation. It is possible to see the peaks around the lags multiple of 24, with the highest peak being at lag 168, indicating that there is still a seasonality pattern. Finally, Figure 4.14 shows that a strong seasonality component is still present. The one-lag differentiation is very effective at removing the trend of the data. However, it has little effect on removing seasonality patterns.

As shown by the autocorrelation plot in Figure 4.8, there is high autocorrelation in the lags multiple of 24, especially the lag 168, which indicates that the time-series data has a daily and a weekly pattern. Both time intervals will be tested as lag values to differentiate the time-series, to try to remove the seasonality pattern.

The data with 24 lags of differentiation is presented in Figure 4.15. There is less variation in the time-series values than in the previous differentiation plot. However, for all weeks, there are two

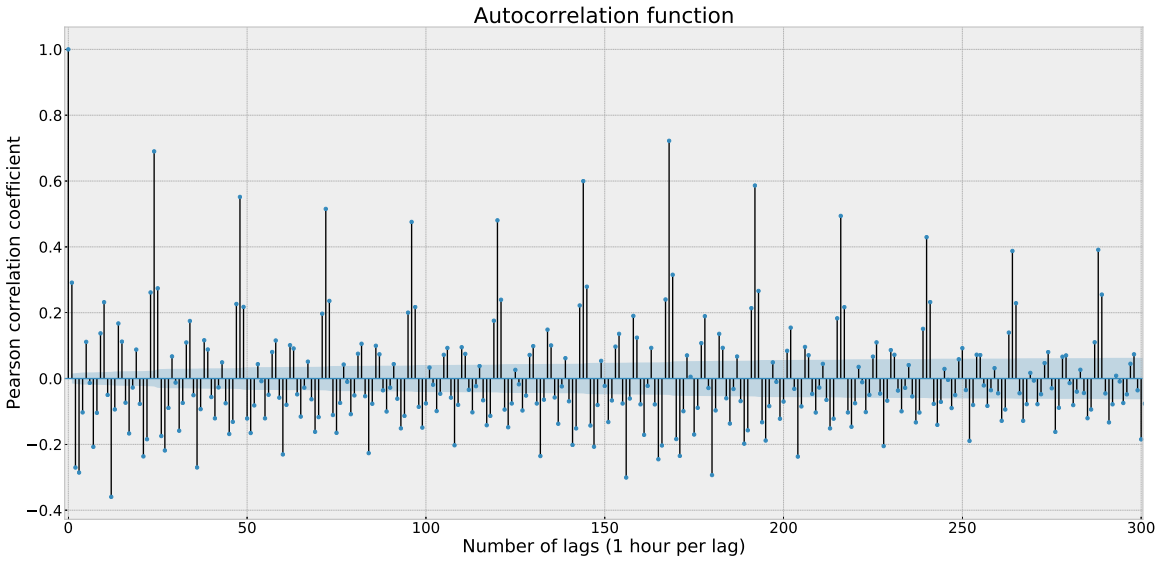

Figure 4.13: Autocorrelation plot of the time-series of the vehicular network data with differentiation of one lag.

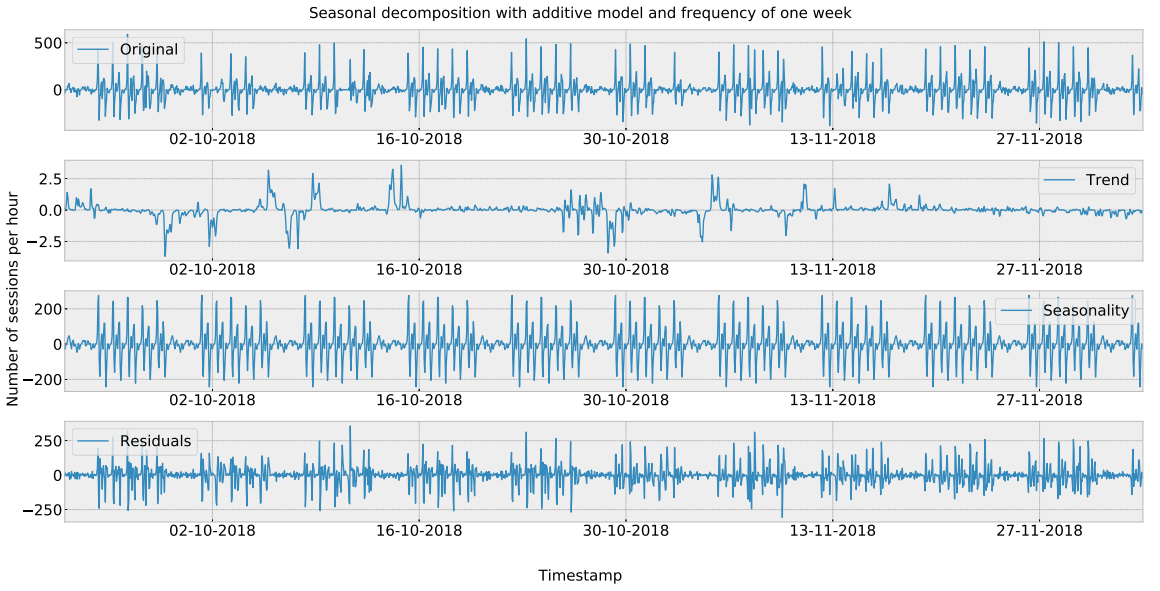

Figure 4.14: Seasonal decomposition of the vehicular network with a one-lag differentiation using moving averages with additive model and frequency of one week. The first plot shows the original data; the second shows the captured trend; the third plot shows the seasonality; the last plot shows the residuals.

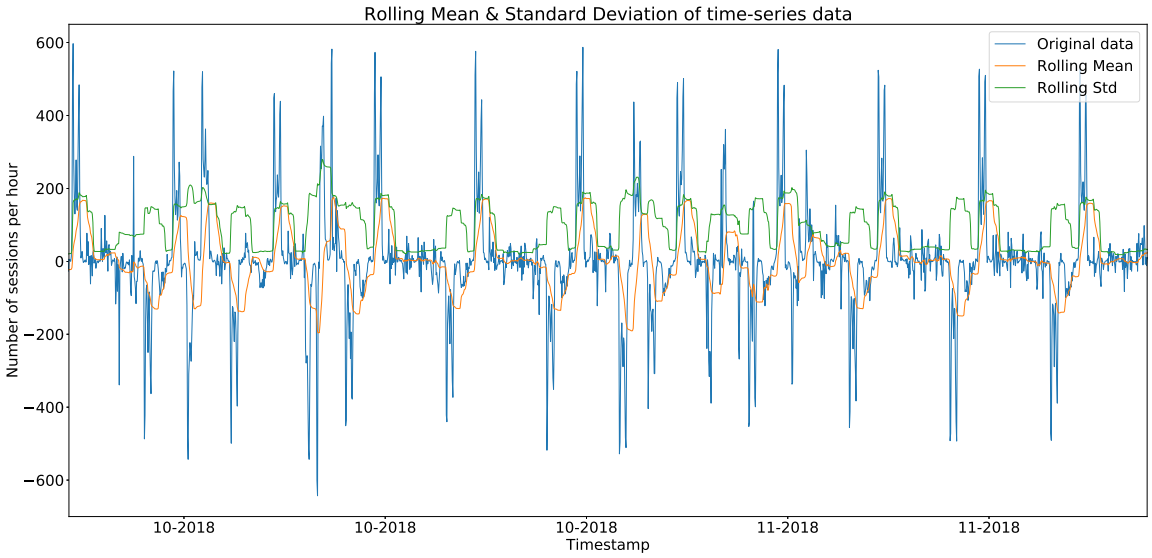

Figure 4.15: Time-series data of the vehicular network with differentiation of 24 lags, with the rolling mean and rolling standard deviation represented, with a rolling window of 24 hours.

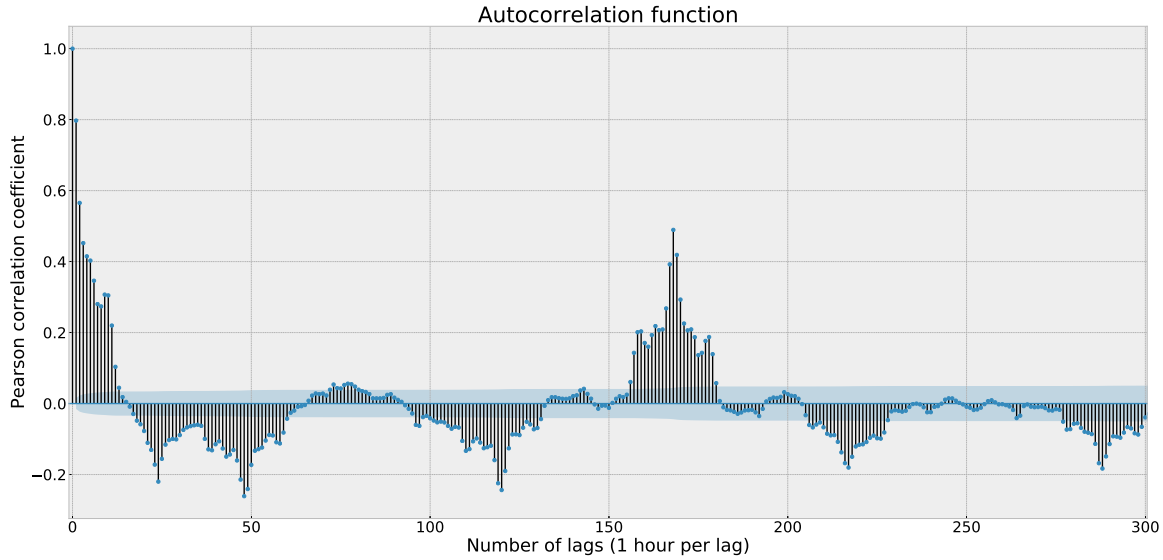

Figure 4.16: Autocorrelation plot of the vehicular network with differentiation of 24 lags.

days with bigger variations in its values. That happens due to the differentiations of weekends when compared with weekdays, and vice-versa. Because their distribution is not the same, their difference is higher than on other days where the values are similar between each other and their difference is closer to zero (the number of sessions in a Friday is not similar to the number of sessions in a Saturday, neither the number of sessions in a Sunday is similar to the number of sessions in a Monday).

The autocorrelation function for the data with 24-lag differentiation (Figure 4.16) shows a positive peak autocorrelation around the lag 168, indicating the similarity of values across different weeks. The seasonality decomposition (Figure 4.17) indicates that there is a strong seasonality pattern around the weekends, as expected.

The time-series data with a differentiation of 168 lags is presented in Figure 4.18. There is no

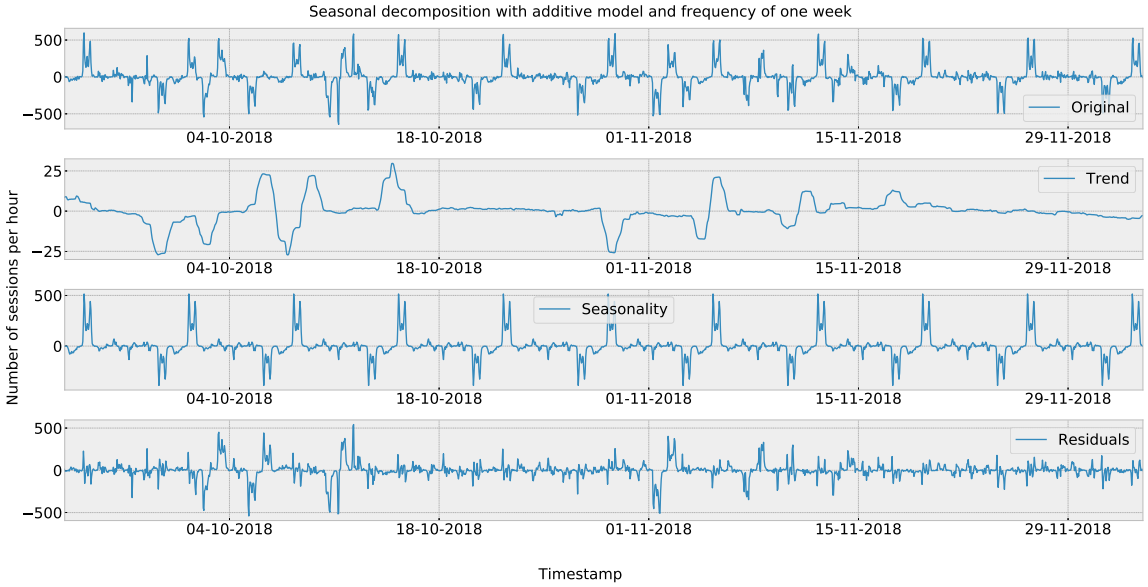

Figure 4.17: Seasonal decomposition of the vehicular network with a 24-lags differentiation using moving averages with additive model and frequency of one week. The first plot shows the original data; the second shows the captured trend; the third plot shows the seasonality; the last plot shows the residuals.

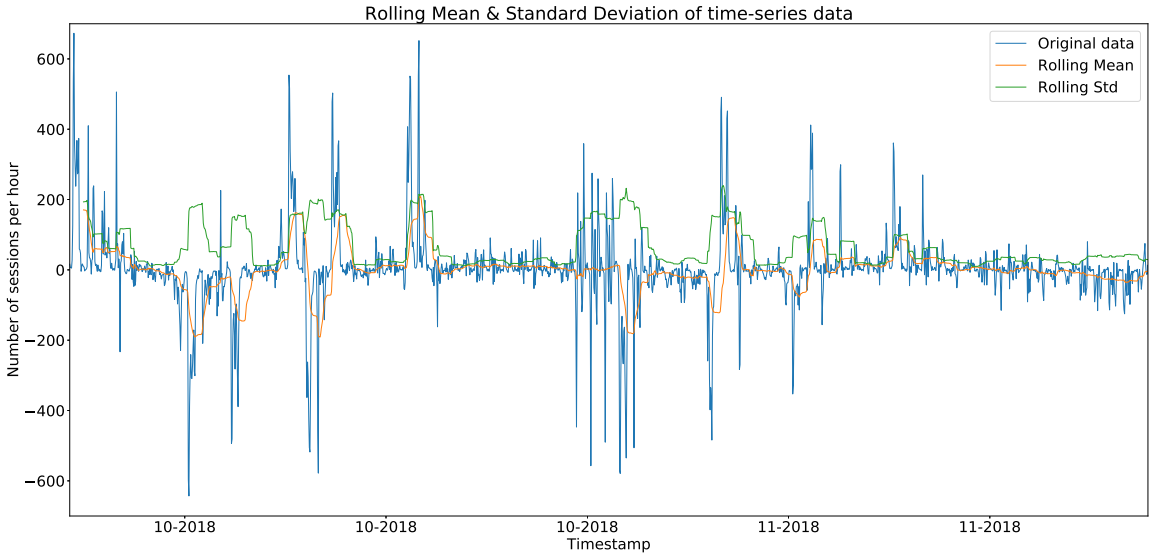

Figure 4.18: Vehicular network data with differentiation of 168 lags, with the rolling mean and rolling standard deviation represented, with a rolling window of 24 hours.

clear pattern in the data. Statistically, the standard deviation is lower than the data with 24 lags of differentiation, which may indicate that the data is closer to being a stationary distribution. The autocorrelation function (Figure 4.19) shows a very negative correlation around the lag 168, which is expected after the 168-lag differentiation. The seasonality decomposition (Figure 4.20) shows that there is still a weak seasonality pattern for every weekday that was not removed.

The 1-lag, the 24-lag and the 168-lag differentiations approximate the time-series data to a stationary process, which is essential for the accurate forecasting of values. Several models such as ARIMA only have the capacity for modeling stationary processes.

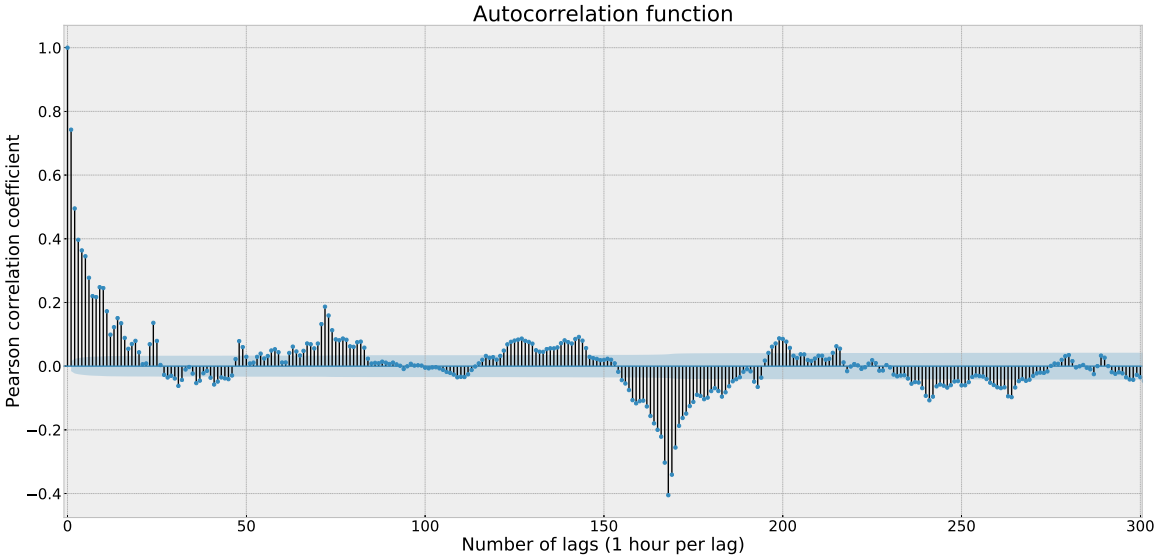

Figure 4.19: Autocorrelation plot of the vehicular network data with differentiation of 168 lags.

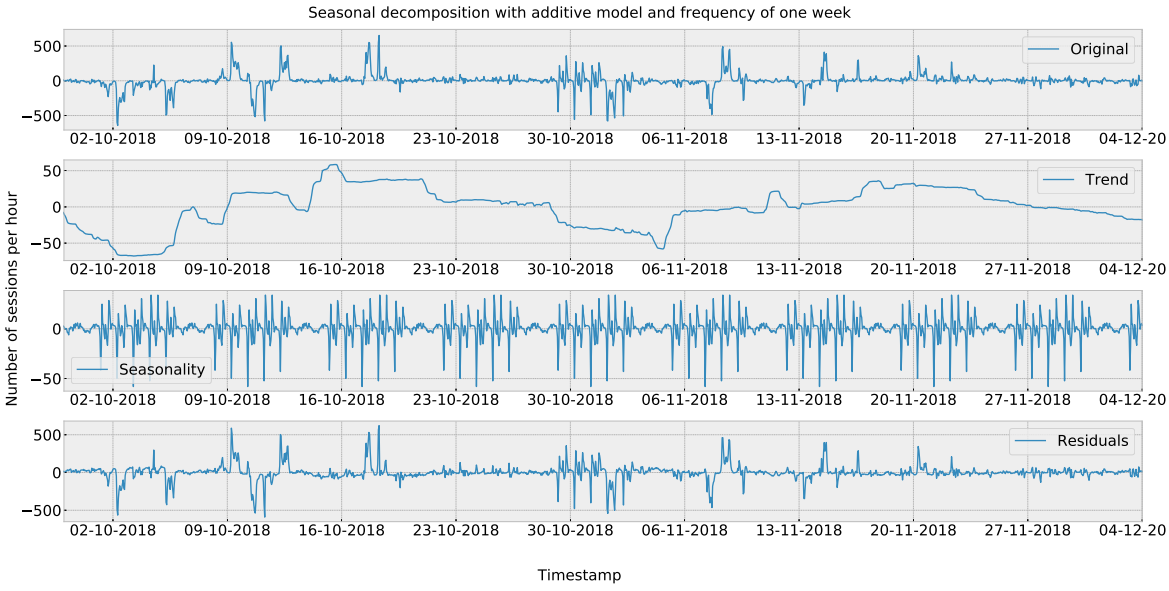

Figure 4.20: Seasonal decomposition of the vehicular network with a 168-lags differentiation using moving averages with additive model and frequency of one week. The first plot shows the original data; the second shows the captured trend; the third plot shows the seasonality; the last plot shows the residuals.

In this section, various analysis techniques were used to prove that approximation. Other differentiations were tested, but the results were not satisfactory, such as the composition of previous differentiations (24-lag differentiation and 168-lag differentiation), differentiation with moving average, differentiation with the exponential weighted average and others. It was not possible to choose one lag number as the best differentiation value because the data distributions generated produce equally good results.

In the next section, the three differentiation lags will be tested as input to the forecasting algorithms, to check what differentiation lag produces most accurate for each algorithm.

4.4 Forecasting task approaches and results

The goal of this section is the creation of a model to forecast the number of sessions in the vehicular network. The forecasting task consists of, given the past values of the number of sessions per hour in the vehicular network $\{y_0, y_1, ..., y_t\}$ and other possible additional features, forecast the value y_{t+1}, where y_t represents the number of sessions in the network at hour t.

With the current dataset, two weeks will be used as cross-validation data (from the 11^{th} of November to the 24^{th} of November) to optimize the forecasting models, and the last two weeks (from the $25th$ of November to the 8^{th} of December) will be used as test data, to perform the final tests and calculate the performance metrics. The previous weeks (from the 17^{th} of September to the 10^{th} of November) will be used as training data. Overall, the dataset will be separated in 55 days for the train set (66.3%), 14 days for the cross-validation set (16.9%) and 14 days for the test set (16.9%) (Figure 4.21).

There were two reasons for this dataset division. Firstly, two weeks were chosen for the cross-validation and test set for every weekday to be present at least two times in each set, to be a representative sample. Secondly, the sets have to be from a continuous time interval (they cannot be shuffled), because the recurrent models can use the information about the previous number of sessions per hour (it can have memory) to make a forecast. If the data was shuffled, the memory about the previous data was not ordered. Because the data is not shuffled, it is also easy to visualize separately the forecasts made by the algorithms in each set.

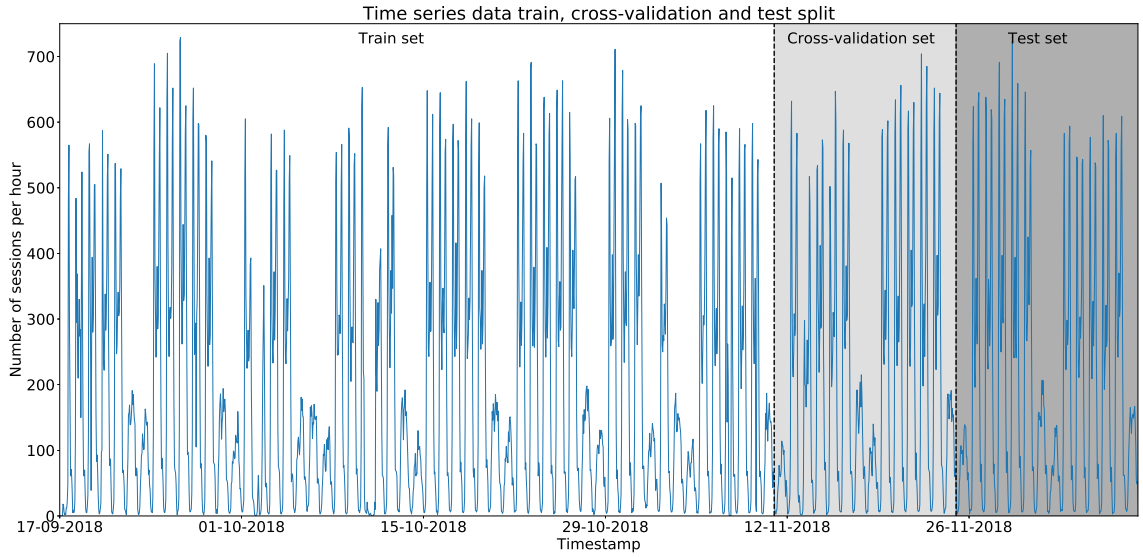

Figure 4.21: Vehicular network dataset split in training set, cross-validation set and test set.

The performance metric used to evaluate the forecasts will be the Root Mean Squared Error (RMSE) (the lower the RMSE, the better). RMSE is the square root of the average of squared differences between the forecasts and the actual observations, as described in Equation 4.4 [102].

Let y be the actual observations, $\hat{y}$ the forecasted values, and n the number of forecasted values,

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}. \tag{4.4}$$

The RMSE units are the same units of the time-series data. In this case, the RMSE units will be the number of sessions per hour.

The RMSE has some advantages when used as a performance metric. It gives high weight to large errors, being useful to apply when large errors are undesirable, it is symmetric and is independent of the absolute value of the forecast or the observation, only depending on the difference between them. However, it has low interpretability, which means that it is not easy to understand if a RMSE value is a low or high error without a reference value for that specific task. To help with the interpretability problem, baseline models will be used to compare the RMSE with a baseline result.

A second performance metric more adequate for interpretability purposes will be used. The Mean Absolute Percentage Error (MAPE) is a measure that calculates the accuracy of forecasts. Its formula is described in Equation 4.5.

Let y be the actual observations, $\hat{y}$ the forecasts and n the number of forecasted values,

$$MAPE(y, \hat{y}) = \frac{1}{n}\sum_{j=1}^{n}\left|\frac{y_j - \hat{y}_j}{y_j}\right|. \tag{4.5}$$

The biggest advantage of the MAPE is the easy interpretability in the form of a percentage of error. Nonetheless, this metric also has some drawbacks:

- if there is an observation with a zero value, the MAPE will result in an error due to a division by zero;
- the measured error is relative to the observation. This leads the final result to be heavily dependent of the observation values and not of the difference between the observations and the forecastings;
- given that the error is relative to the actual observation, it is possible for the MAPE to be above 100%;
- there is a higher penalty on negative errors than on positive errors.

Due to these drawbacks, the MAPE will not be used to minimize the forecasting error. Instead, the RMSE will be the performance metric to be minimized, and the MAPE will be used as an indicator, for easier interpretation of the results.

4.4.1 Persistence model

The first approach used to establish a benchmark for the forecasting task is the persistence model (or naive forecasting method). This approach forecasts the future as the present value. It performs poorly in seasonal data. An improved version of this method is used in this work, taking into account the observations done in Section 4.3 regarding the number of lags and seasonality patterns. Instead of only forecasting the value of the previous hour as the next hour (1-hour lag forecasting), the forecast is given by predicting the value as the observation K hours before, which is the period of a seasonal cycle (Equation 4.6).

K	RMSE (number of sessions per hour)	MAPE (%)
1	115.60	58.68
24	131.95	51.22
168	74.29	29.63

Table 4.1: RMSE and MAPE values of the persistence model with different value of K.

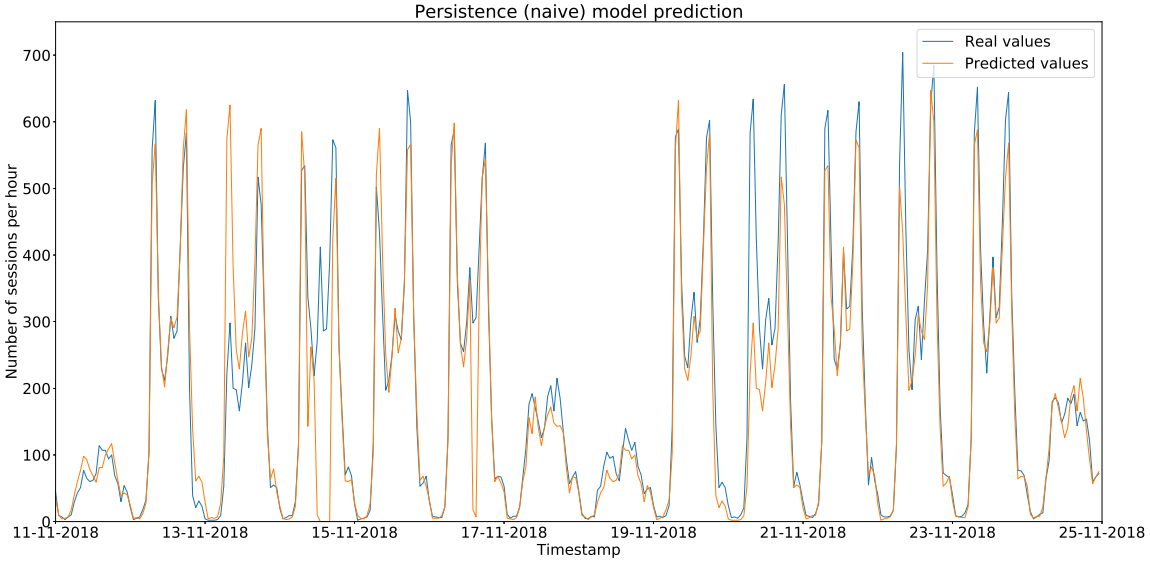

Figure 4.22: Forecasts made by the persistence model with K=168 in the cross-validation set of the vehicular network, compared with the real values.

Let y be the actual observations, $\hat{y}$ be the forecasts, and K the period of a seasonal cycle,

$$\hat{y}_t = y_{t-K}. \tag{4.6}$$

The K values tested were 1 (predict the same values as in the last hour), 24 (predict the same values as in the last day) and 168 (predict the same value as in the last week), as discussed in Section 4.3. The tests were done using the cross-validation set. The results are in Table 4.1. With $K = 24$, the RMSE is the highest, indicating that it is not a good value of K. Due to the high penalization of large errors by the RMSE, the large differences in the number of sessions per hour on weekends are severely penalized, making it the model with higher RMSE. On the other hand, $K = 168$ has the best result, with the lower value of RMSE. Figure 4.22 shows the forecasts made by this model compared with the real observations.

The RMSE of 74.29 sessions per hour will be the baseline for testing other models, corresponding to a MAPE of 29.63%.

4.4.2 ARIMA model

The ARIMA algorithm (described in Subsection 2.4.4) used was provided by *statsmodels*[1], a Python library designed for the testing of statistical models and for conducting statistical tests. Before fitting the model, the data is normalized, removing the mean and scaling the data to unit variance. The performance metrics used are RMSE and MAPE. The tests are conducted using walk-forward validation: after each forecast, the real observation is added to the training dataset, and a new model

[1] `https://www.statsmodels.org/`

d	**RMSE**
0	97.04
1	102.49
24	77.23
168	51.13

Table 4.2: RMSE results for the forecast with the ARIMA model for different d values.

p	**RMSE**
0	102.74
1	78.40
2	75.98
3	74.06
4	71.05
5	71.76

Table 4.3: RMSE results for the forecast with the ARIMA model for different p values.

q	**RMSE**
0	89.01
1	79.47
2	74.00
3	70.94
4	71.68
5	68.83

Table 4.4: RMSE results for the forecast with the ARIMA model for different q values.

is fitted to forecast the next point. It was opted to perform walk-forward validation to perform a fair comparison with the machine learning algorithms. While machine learning techniques can use previous points to forecast the next point (just using it as input, without having to re-train the model), for the ARIMA model to take into account the previous point, it needs to refit a new model with all the previous points.

Extensive tests were done using the ARIMA model to forecast the number of sessions in the cross-validation set. All the combinations of the following three hyper-parameters were tested (as described in the Subsection 2.4.4):

- parameter p for Autoregressive (AR) degree - 0, 1, 2, 3, 4, 5;
- differentiation parameter d - 0, 1, 24, 168;
- parameter q for Moving Average (MA) degree - 0, 1, 2, 3, 4, 5.

The differentiation (d) values tested are the same as the K values in the previous subsection (as discussed in Section 4.3). The parameters p and q must not have a very high value; otherwise the model becomes too complex and it takes too much time to fit and forecast the data. Values between 0 and 5 will be tested for the AR and MA degree (p and q parameters, respectively).

The average RMSE for various differentiation parameters (d) is presented in Table 4.2. With 24-hour differentiation, the average results are better than with no differentiation or with 1-hour differentiation. The results with the 168-hour differentiation are even better than with the 24-hour differentiation.

With the increase in the parameters p and q, the average RMSE seems to slightly improve, as it can be seen in Tables 4.3 and 4.4. However, the improvements are only marginal when compared with the improvements of the tests with various differentiation values (Table 4.2).

Best results	RMSE	p	d	q
1	49.50	3	168	0
2	49.62	2	168	2
3	49.62	1	168	3
4	49.63	2	168	3
5	49.71	4	168	1
6	49.74	1	168	2
7	49.75	4	168	0
8	49.81	1	168	4
9	49.87	3	168	1
10	50.00	1	168	1

Table 4.5: Ten best results of the ARIMA approach in the forecasting.

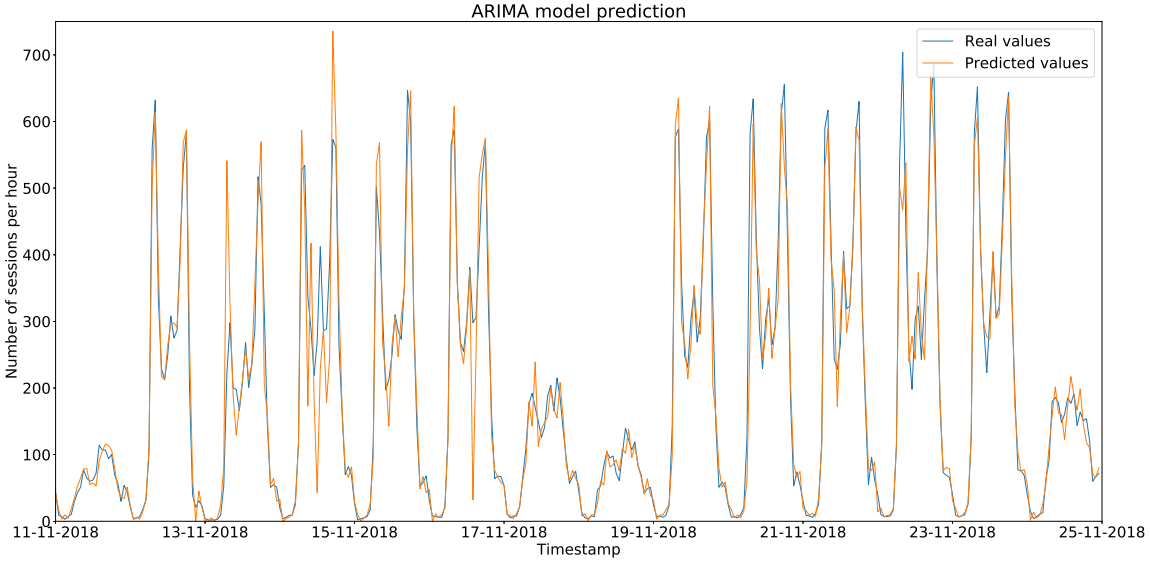

Figure 4.23: Forecasts made by the ARIMA model in the cross-validation set of the vehicular network, with the order parameters (3,168,0), compared with the real values.

The ten best results are in Table 4.5. All best results have as differentiation 168 lags, and the p and q values vary from 0 to 4. The best result is when $p = 3$ and $q = 0$ (the forecasts are in Figure 4.23), with a RMSE of 49.5 sessions per hour and a MAPE of 26.20%, improving in comparison with the persistence model.

4.4.3 Classical machine learning models

Another approach to the construction of a model with the task of forecasting values is to use machine learning algorithms (thoroughly explained in the Section 2.4). In this subsection, various classical machine learning algorithms will be used. By classical machine learning algorithms, it is excluded any kind of neural network algorithm (those algorithms will be tested in the following subsections). The tested algorithms in this subsection are:

- Linear Regression;
- Lasso;
- ElasticNet;
- Stochastic Gradient Descent (SGD) Regressor;
- Support Vector Regression;

- Random Forest;
- Gradient Boosting Regressor;
- AdaBoosting Regressor (AdaBoost);
- eXtreme Gradient Boosting Regressor (XGBoost).

The Linear regression, Lasso and ElasticNet are linear approaches for modeling the relationship between a dependent variable and independent variables. The Stochastic Gradient Descent (SGD) regressor models a linear relationship minimizing a loss function with the use of SGD, an iterative method for optimizing an objective function. Support Vector Regression is based on Support Vector Machines (SVMs), an algorithm designed to create a set of hyperplanes to be used for classification or prediction. It is also possible to create non-linear classifiers or regressors applying the kernel trick. In the tests, the Radial basis function (RBF) kernel was used. Random Forest, Gradient Boosting Regressor, AdaBoost Regressor and XGBoost are ensemble methods that take advantage of the construction of multiple models to obtain better predictive performance. Particularly, while Random Forest is a bagging algorithm that trains multiple decision trees with different subsets of the training set and each one vote with equal weight, Gradient Boosting Regressor, AdaBoost Regressor and XGBoost are boosting algorithms that train new model instances focused on the prediction error of the previous instances.

To implement the tests, it was used the *Scikit-learn* library for Python [103]. It provides efficient and well-tested versions of a large number of machine learning algorithms, as well as a simple Application Programming Interface (API) with a very complete online documentation.

The pre-processing of the data for the input of the machine learning algorithms is essential for the results obtained. This pre-processing will be the same for all the machine learning algorithms used in this subsection, as well as for other machine learning algorithms used in the next subsections. Firstly, the differentiation is made (Equation 4.3). Afterwards, the data is normalized, removing the mean and scaling data to unit variance. For the data to be inputted to the algorithms, it must be in a matrix of size $n * l$, where n is the number of examples in the dataset and l is the number of features to train the models. The output matrix has size $n * p$, where for each example in the input, a set of forecasts of size p is made. In these tests, the input features will be the l values previous to the forecasted value. The previous values will be used to detect patterns and forecast the value of the next point. The number of previous values, l, is a hyper-parameter that must be tested. The forecasting size p is one because the algorithms will try to predict only the next value (the next hour).

The tests will be conducted using walk-forward validation on the cross-validation dataset (described in the Subsection 4.4.2). The following hyper-parameters will be tested for the algorithms described above in this subsection:

- differentiation lag (d): 0 (no differentiation), 1, 24, 168;
- number of previous values (l): 1, 12, 24, 48, 168.

The remaining hyper-parameters for each algorithm will have the default values. Each algorithm has its own hyper-parameters, and tuning all of them would become a time-expensive task and probably not worthy, because the default hyper-parameters of the library *Scikit-learn* are well tested and perform well for most cases.

In Figure 4.24 it is shown the average RMSE for the different algorithms with variation in the differentiation lag. Most algorithms have better results when the differentiation lag is 168. However, three ensemble algorithms (Random Forest, Gradient Boosting Regressor and XGB Regressor) have better results when there is no differentiation. These ensemble algorithms are able to learn non-linearities in the data with the use of several machine learning models, instead of algorithms that are

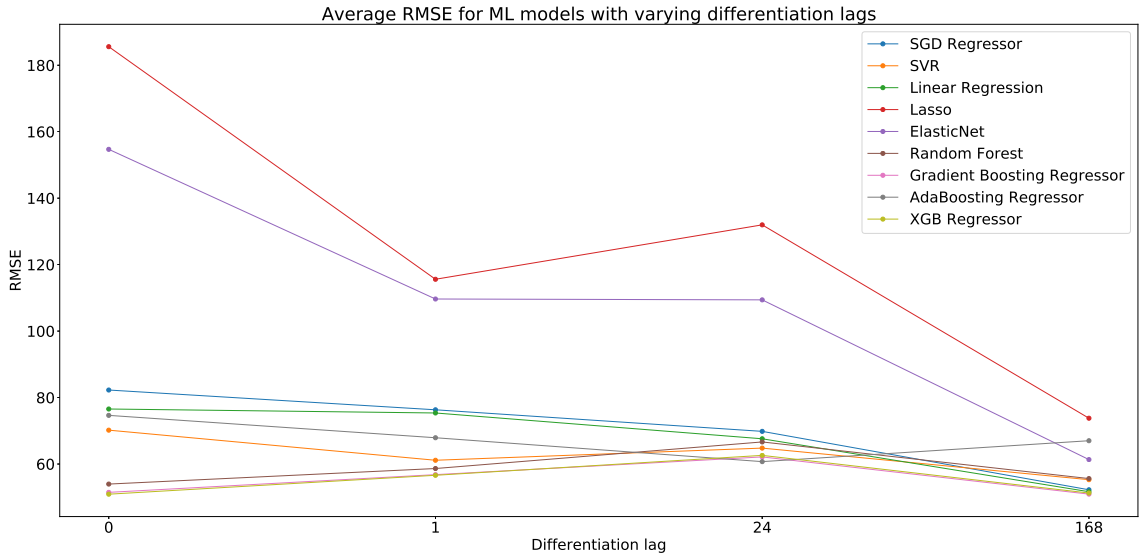

Figure 4.24: Average RMSE for classical machine learning algorithms varying the differentiation lag.

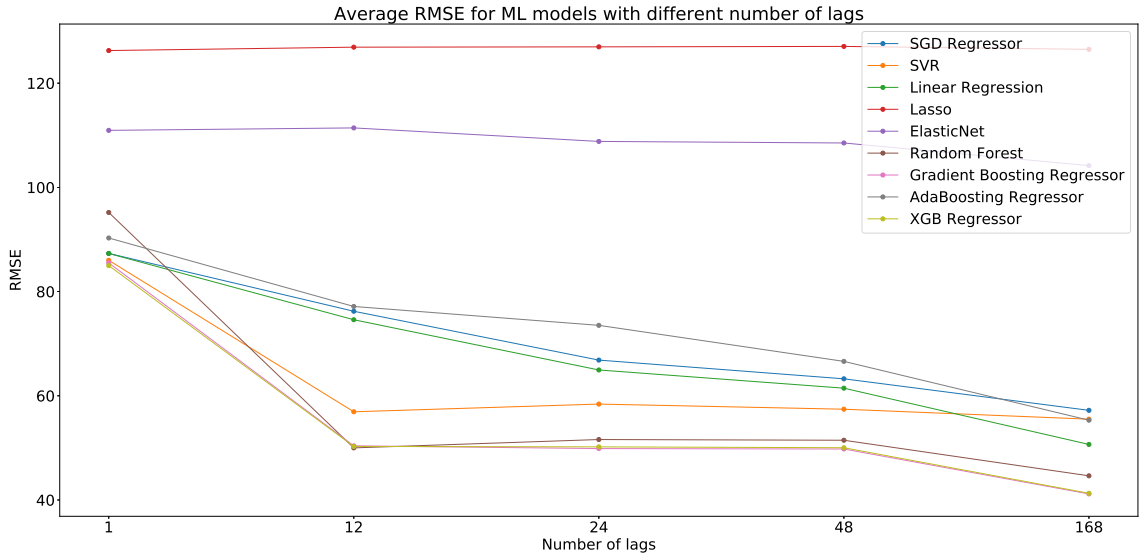

Figure 4.25: Average RMSE for classical machine learning algorithms varying the number of lags as input.

only able to capture linear relationships, where the data stationarity is needed, such as Linear Regression. Those results indicate that a neural network may achieve good results with no differentiation, due to its capacity to model the non-linearities of data.

Figure 4.24 also shows that the ElasticNet and the Lasso, two linear approaches for regression, only have comparable results to the other machine learning algorithms with a differentiation of 168 lags.

Figure 4.25 shows the average RMSE for the different algorithms with variation in the number of input lags. For ElasticNet and Lasso, the increase in the number of lags in the input has little effect. For the other algorithms, on average, the higher the number of lags, the better the performance, as expected. There is only a slight improvement in going from 12 to 24, and from 24 to 48 lags. There is a big reduction in the average RMSE going from 1 to 12 lags, and from 48 to 168 lags.

The best result (RMSE of 36.43 sessions per hour, MAPE of 27.31%) was achieved using the

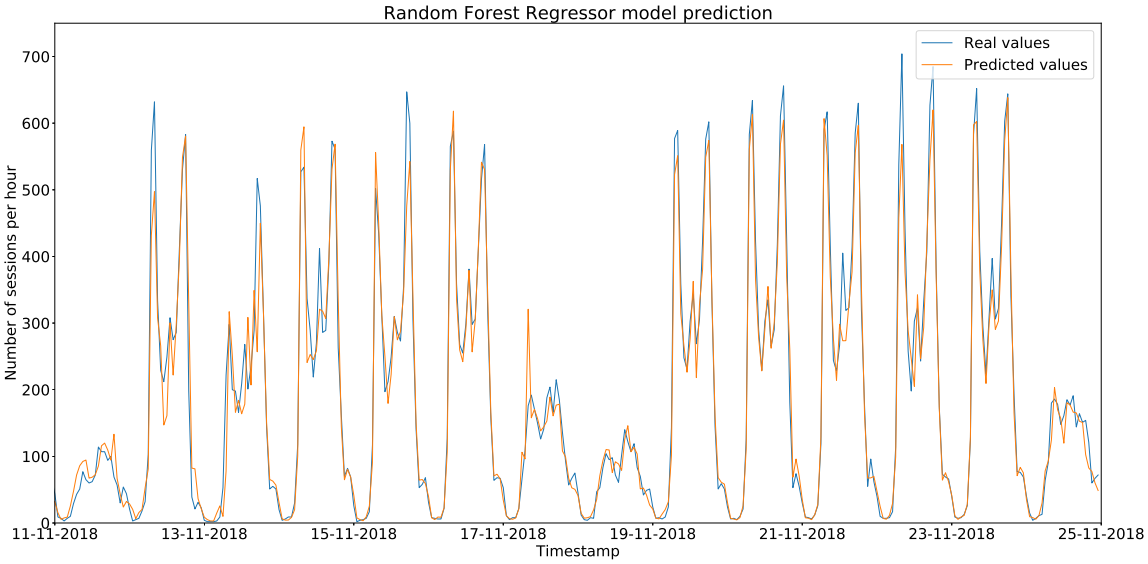

Figure 4.26: Prediction made by the Random Forest model in the cross-validation setof the vehicular network, with 168-lag differentiation and 12 input lags, compared with the real values.

Random Forest algorithm, with no differentiation, and an input lag of 12 (the predictions are in Figure 4.26).

In the machine learning algorithms, additional features can be added to the input besides the previous values. Those additional features must have a relationship with the data to be relevant and improve the forecasts. Three features of the data were considered useful to forecast the number of sessions in the next hour, and their effect on the results was tested. The additional features were:

- **Day of the week** - this feature is a one-hot encoded array with size 7, with '1' on the day of the week of the forecasted value. This feature was chosen because the number of sessions in different days depends heavily on the day of the week;
- **Workday** - this feature is a flag that is '1' if the forecasted value is a weekday and not a holiday, and '0' otherwise. This feature helps to model the weekends, where the number of sessions is much lower than on a regular day of the week, and also the holidays because, as seen in the time-series analysis (Section 4.2), they have different behavior from the weekdays;
- **Hour of the day** - this feature is a one-hot encoded array with size 24, with '1' on the hour of the day of the forecasted value. This feature was chosen because the hour of the day has a major influence on the number of sessions.

All the combinations of additional features were tested using all the combinations of hyper-parameters tested before. Table 4.6 shows the eight different combinations of additional features to be tested, and the average RMSE for each combination of features is in Table 4.7. On average, the results without any additional features were the worst, and the tests with all the additional features provided the best results. The feature of the hour of the day seems to have a higher impact on the results than both other features. However, on average, the impact of the additional features was minimum, improving the average RMSE on less than 3 sessions per hour.

Figure 4.27 shows the average RMSE varying the number of input lags, with all additional features used. In comparison with the result of the same tests but without any additional features (Figure 4.25), the RMSE for one lag is lower, due to the additional information. With only one lag as input and without additional features, there was not much that could be learned from the data. With the

Set of tests	Day of the Week	Work day	Hour of the day
1	False	False	False
2	True	False	False
3	False	True	False
4	False	False	True
5	True	True	False
6	True	False	True
7	False	True	True
8	True	True	True

Table 4.6: The eight different set of combinations for testing additional features. For each set of tests, for each feature, if the entry is True, the feature was used. Otherwise, the feature was not used.

Set of tests	RMSE
1	75.35
2	73.99
3	73.97
4	72.99
5	74.41
6	72.75
7	74.07
8	72.48

Table 4.7: Average RMSE for the forecasts for different combinations of additional features.

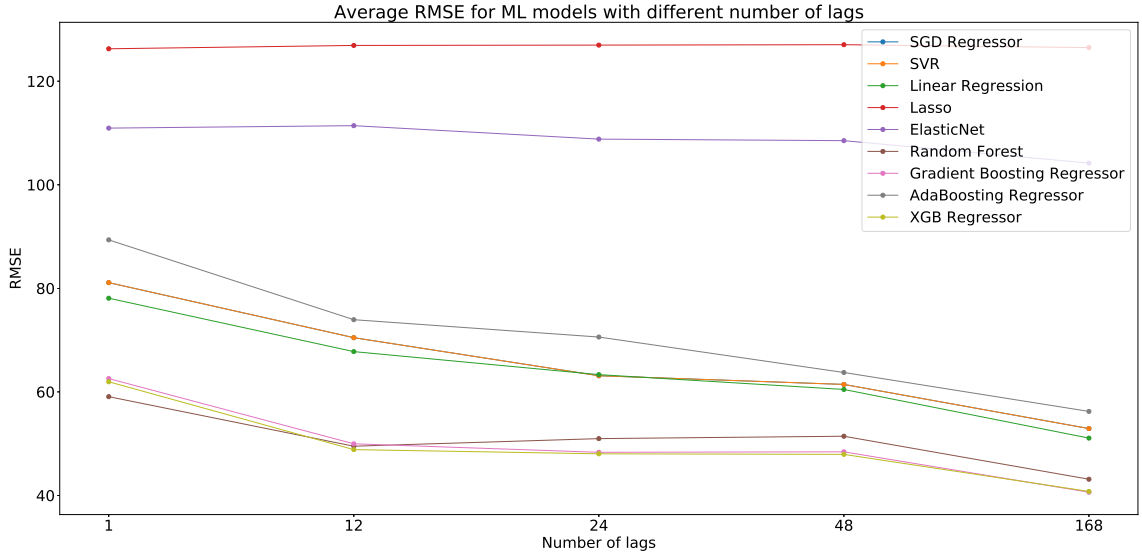

Figure 4.27: Average RMSE for classical machine learning algorithms varying the number of lags as input with all the additional features added in the input.

additional features, it is easier to make accurate forecasts. As the number of lags increases, the results of the tests with additional features become closer to the results of the tests without additional features, because the additional features can be derived from the data. For example, with 24 lags it is easier to know what is the hour of the day, and with 168 lags it is easier to know what is the day of the week, even without those features being explicitly fed to the model.

The small variations of the average RMSE for different additional features seem to have little impact on the best result for each combination of additional features, with the eight sets of tests

Set of Tests	Algorithm	Differentiation	Lag number	RMSE	MAPE (%)
1	Random Forest	0	12	36.43	27.31
2	Random Forest	0	48	35.66	25.99
3	Random Forest	0	48	36.57	22.79
4	XGBoost Regressor	0	24	36.70	34.23
5	Gradient Boosting Regressor	0	24	36.19	32.70
6	Random Forest	0	168	36.42	22.25
7	Gradient Boosting Regressor	0	24	36.19	35.60
8	XGBoost Regressor	0	24	36.36	35.74

Table 4.8: Best results for each set of tests using different additional features, with the best algorithm, its hyper-parameters, RMSE and MAPE.

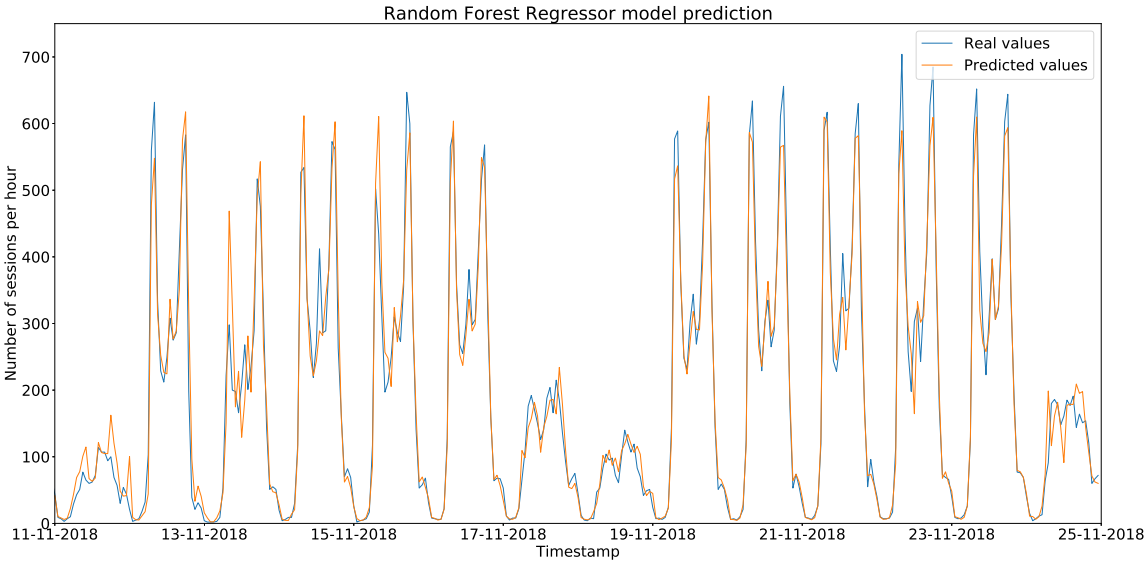

Figure 4.28: Prediction made by the Random Forest model in the cross-validation set of the vehicular network, compared with the real values, with a RMSE of 36.3 sessions per hour.

having the best RMSE between 35 and 37 sessions per hour (Table 4.8). All the eight best results are from an ensemble algorithm (four of the best results are achieved with Random Forests, two with XGB Regressor and two with Gradient Boosting Regressor). All the best results are achieved with no differentiation, and with a lag number between 12 and 168.

It is also notorious that the Random Forest algorithm produces a lower MAPE than the others best algorithms. That happens due to the better predictions at observations with a low value. Because MAPE calculates the prediction error relative to the real observation value (Equation 4.5), the same absolute error has higher weight when dealing with observations with a lower number of sessions per hour than dealing with observations with a higher number of sessions per hour. The Random Forest forecasts in the second set of tests, with no differentiation and 48 as lag, can be seen in Figure 4.28.

4.4.4 Feed-Forward Neural Networks

In this subsection, it will be used a *Time-Lagged Feed-forward Neural Network* to forecast the number of sessions in the vehicular network, using a sliding time window. The description of a Feed-Forward Neural Network and its hyper-parameters are in Subsection 2.4.1. The pre-processing is identical to the pre-processing done in the previous subsection (described in Subsection 4.4.2).

It will be adopted a batch size of one and the networks will be trained for 500 epochs. To control the overfitting, the model with lower RMSE in the cross-validation set along the number of epochs

Model Number	Number of hidden feed-forward layers	Number of neurons per layer	Dropout value
0	1	20	1
1	2	20	1
2	3	20	1
3	4	20	1
4	1	200	1
5	2	200	1
6	3	200	1
7	4	200	1
8	1	20	0.5
9	2	20	0.5
10	3	20	0.5
11	4	20	0.5
12	1	20	0.2
13	2	20	0.2
14	3	20	0.2
15	4	20	0.2
16	1	200	0.5
17	2	200	0.5
18	3	200	0.5
19	4	200	0.5
20	1	200	0.2
21	2	200	0.2
22	3	200	0.2
23	4	200	0.2

Table 4.9: Configurations tested for the feed-forward neural network.

will be chosen as the best model for the network with the hyper-parameters chosen. Throughout the number of epochs, the training RMSE and the cross-validation RMSE will also be monitored, for a better understanding of the internal behavior of the model. The loss function to optimize will be the RMSE (Equation 4.4), and the optimizer will be the Adaptative Moment Estimation (Adam) [104], because it is a well-tested optimizer known for its speed, little memory requirements and good results in comparison with other optimization techniques. It is also adequate to non-stationary objectives.

Different neural network configurations will be tested. The number of hidden feed-forward layers will vary between one and four and all the layers will have the same number of neurons, tested with 20 and 200 neurons for all configurations. The activation function used for all the hidden feed-forward layers will be the Rectified Linear Unit (ReLU) function, due to its known good results when dealing with sparsity and to the reduced likelihood of the gradient to vanish [105]. The output layer will contain one neuron with no activation function. It will also be tested Dropout, a neural network mechanism to regularize the network and prevent overfitting [106]. Dropout will be applied to all dense layers, with exception to the last (the output layer), and it will be tested with the values of 0.2, 0.5 and 1 (no Dropout). A summary of the 24 tested network configurations is in Table 4.9.

For each of these network configurations, other hyper-parameters will also be tested, namely:

- differentiation lag (d): 0 (no differentiation), 1, 24, 168;
- lag number (number of previous values in the input, l): 4, 12, 24, 48, 168.

These tests will be run on Python, using the libraries *Keras* [51] and *Tensorflow* [107] for the construction and testing of the neural network models.

The average RMSE for the different number of input lags can be found in Table 4.10. If the number of lags is equal or higher than 12, the average RMSE does not vary much. However, if the number of lags is less than 12, the forecast performance is much worse. This result indicates that, with the previous 12 hours, it is possible to make an accurate forecast about the number of sessions in the next hour, and that more than 12 hours does not add much relevant information.

When the differentiation is changed (Table 4.11), no differentiation or just one lag of differentiation produces better results than higher lag differentiations (24 lags or 168 lags). This result is similar to

l	**RMSE**
4	55.73
12	49.83
24	50.18
48	51.04
168	49.26

Table 4.10: Average RMSE for the forecast with the feed-forward neural networks for different number of input lags.

d	**RMSE**
0	45.60
q	46.27
24	60.07
168	52.48

Table 4.11: Average RMSE for the forecast with the feed-forward neural networks for varying differentiation lags.

Model Number	**Average RMSE**
0	54.98
1	48.75
2	46.54
3	46.55
4	51.49
5	45.11
6	44.10
7	44.80
8	56.16
9	54.10
10	57.01
11	61.46
12	53.11
13	45.50
14	46.90
15	47.88
16	52.93
17	50.06
18	53.70
19	58.47
20	51.09
21	45.81
22	44.40
23	45.43

Table 4.12: Average RMSE results for each network configuration

the ensemble algorithms in the previous subsection (Subsection 4.4.3), where the best differentiation lags were none or the lowest possible.

The average RMSE results for the different network configurations can be seen in Table 4.12. The best average results are from the models number 5, 6, 7, 21, 22 and 23, that represent the network configurations that have 200 neurons per layer, from 2 to 4 hidden layers and no dropout or only 0.2 of dropout.

The ten best results for all tests are presented in Table 4.13. All the ten best results have either one lag differentiation or no differentiation at all. Moreover, all the ten best results have as number of lags

Best results	Model Number	Differentiation	Lag number	RMSE
1	23	1	12	31.44
2	5	0	24	31.58
3	22	1	12	32.30
4	23	1	24	32.34
5	5	0	12	32.74
6	6	0	24	32.74
7	2	0	12	32.77
8	1	0	12	33.00
9	13	1	24	33.10
10	3	0	12	33.49

Table 4.13: Ten best results on the tests using a feed-forward neural network.

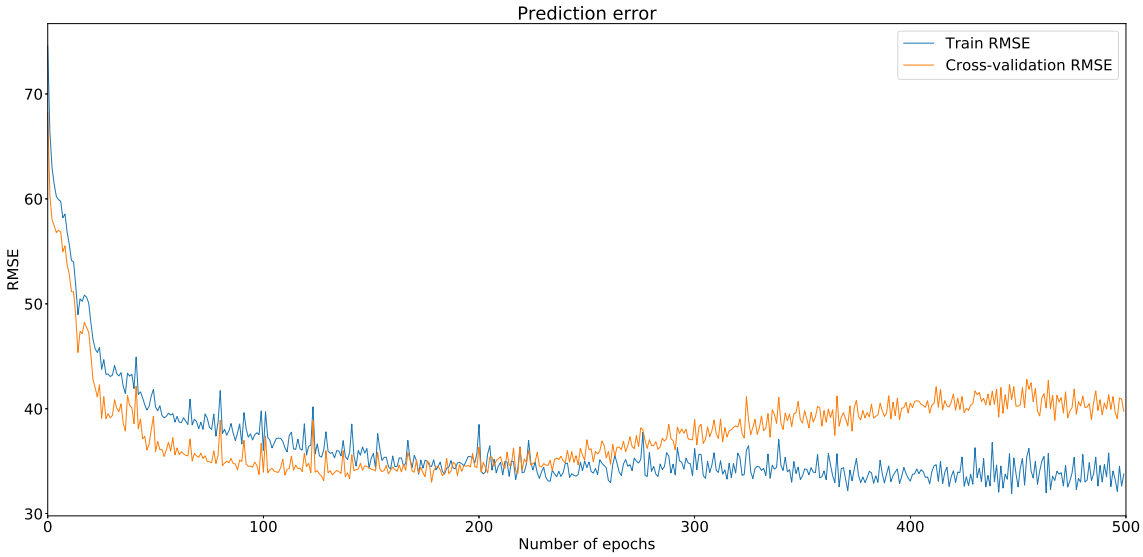

Figure 4.29: Training and cross-validation RMSE variation with the number of trained epochs for the model number 1 with no differentiation and lag number of 12.

12 or 24. The models with the best results vary, with the six best models having 200 neurons per layer and the next four best models having 20 neurons per layer. The best models with differentiation have a network configuration where dropout layers are present with a value of 0.2, while the best models without differentiation have a network configuration where there is no dropout.

The learning process can also be analyzed. Figure 4.29 shows the RMSE for the train and cross-validation set for the model number 1 with no differentiation and a lag number of 12. The spikes are caused due to the low batch size (one). Because the network has two hidden layers, 20 neurons per layer and no dropout, the networks overfits after around 200 epochs, with the lower cross-validation RMSE being at the epoch 178 (RMSE of 33.00).

Figure 4.30 shows the RMSE for the train and cross-validation set for the model number 5 with no differentiation and a lag number of 24. Because the network configuration is identical to the previously analyzed network configuration, but with 200 neurons per layer instead of 20, the model overfits much quicker, as expected, reaching the lowest cross-validation RMSE at epoch 69 (RMSE of 31.58). Due to the higher modeling capacity of the network, the training RMSE is lower for the same number of trained epochs.

Figure 4.31 shows the RMSE for the train and cross-validation set for the model number 23 with a

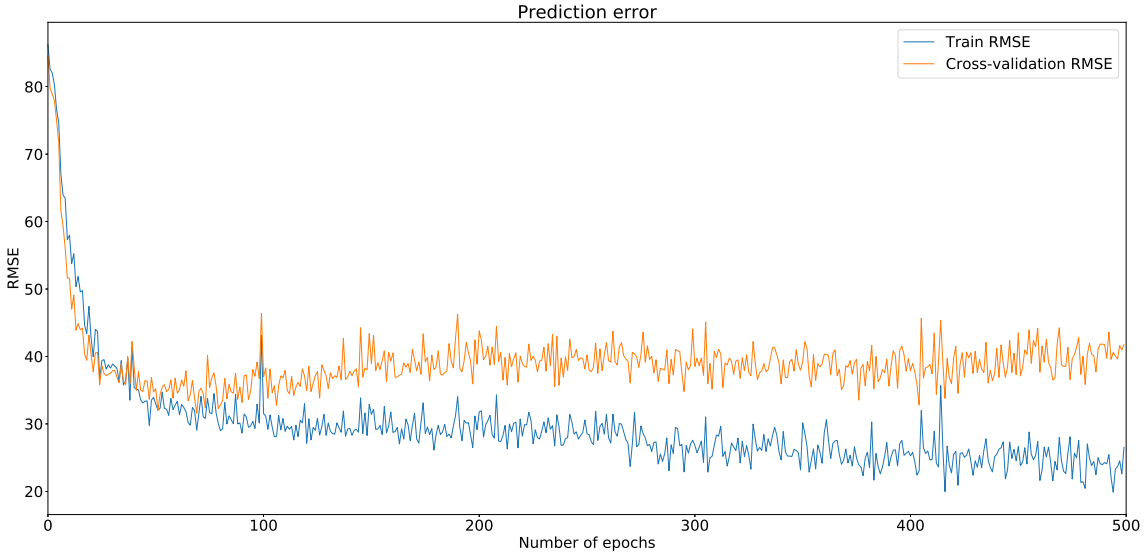

Figure 4.30: Training and cross-validation RMSE variation with the number of trained epochs for the model number 5 with no differentiation and lag number of 24.

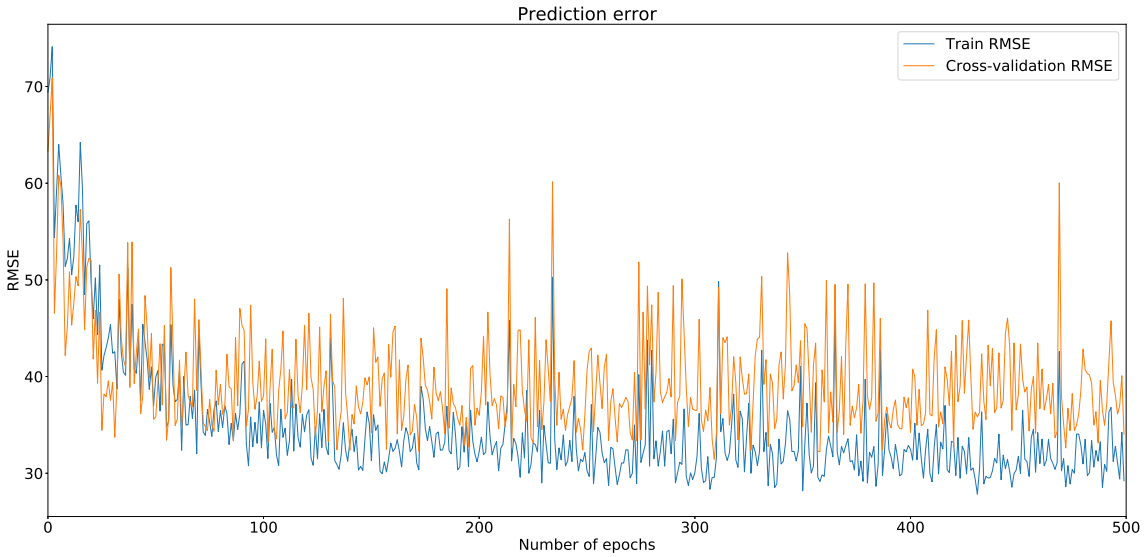

Figure 4.31: Training and cross-validation RMSE variation with the number of trained epochs for the model number 23 with one-lag differentiation and lag number of 12.

1-lag differentiation and a lag number of 12. Because the network has four hidden layers, each one with 200 neurons each, and the dropout value for each layer is 0.2, the best RMSE value is only at epoch 309 (RMSE of 31.44), showing no signs of overfitting because of the dropout value. The abrupt RMSE variations are not only due to the batch size of one, but also due to the dropout and the higher number of layers.

In an attempt to improve the results, additional features that were considered relevant to improve the forecasting were added. The same additional features used in the previous subsection (Subsection 4.4.3) were added: day of the week, workday and hour of the day.

The combinations of additional features are described in Table 4.6. Table 4.14 shows the average RMSE for the eight different combinations of additional features, where the previous ten best network

Best Results\Set of tests	1	2	3	4	5	6	7	8
1	**31.44**	30.43	33.48	**30.68**	31.29	33.29	33.39	32.84
2	31.58	32.79	34.39	36.55	33.56	34.93	34.99	35.84
3	32.3	**29.99**	32.74	31.81	**30.69**	**32.32**	**30.84**	**30.74**
4	32.34	32.13	31.81	33.06	32.15	34.80	33.35	33.61
5	32.74	35.14	**30.88**	32.66	34.16	36.43	32.30	31.74
6	32.74	36.28	33.05	31.79	35.07	36.12	32.47	34.63
7	32.77	37.44	32.06	32.7	33.22	34.78	32.40	37.00
8	33	35.19	36.4	35.79	39.74	35.54	33.65	36.10
9	33.1	33.93	33.79	33.65	36.42	34.16	34.69	36.85
10	33.49	34	31.67	34.34	34.16	33.56	34.19	36.82
Average	**32.55**	**33.639**	**33.027**	**33.303**	**34.05**	**34.59**	**33.23**	**34.62**
Best Result	**31.44**	**29.99**	**30.88**	**30.68**	**30.69**	**32.32**	**30.84**	**30.74**

Table 4.14: RMSE result for the different set of tests with additional features, made only to the previous ten best results.

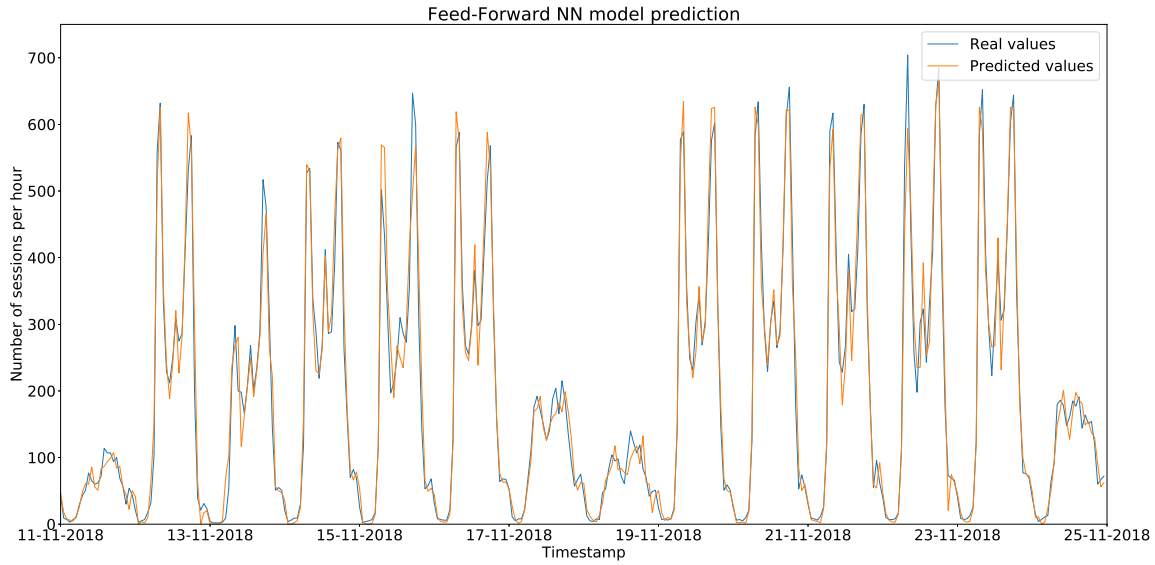

Figure 4.32: Forecasts made by the best feed-forward neural network model in the cross-validation set of the vehicular network, compared with the real values, with a RMSE of 29.99 sessions per hour and a MAPE of 23.26%.

configurations (Table 4.13) are tested with the new features.

On average, the RMSE of the tests has not improved in comparison with the tests without any additional features (set of tests 1). However, the best result for each set of tests shows that almost all configurations with additional features have at least one RMSE that outperforms the best result for the tests with no additional features. Similar to the tests in Subsection 4.4.3, the best result is achieved with only the weekday as feature, achieving a new best RMSE of only 29.99 sessions per hour and a MAPE of 23.26%, with the configuration of 1-lag differentiation, input lag of 12 and the model number 22, which has four hidden layers with 200 neurons and a dropout value of 0.2. The forecasts can be seen in Figure 4.32.

4.4.5 Recurrent Neural Networks

In this subsection, recurrent neural networks will be used to try to improve the forecasts of the vehicular network. The description of the recurrent neural networks is in Subsection 2.4.2. The approach used and the description of the data transformations, the input and the output are described

Model number	LSTM hidden layers	Neurons per Layer	Dropout+Recurrent Dropout
1	1	20	1
2	2	20	1
3	3	20	1
4	1	200	1
5	2	200	1
6	3	200	1
7	1	20	0.2
8	2	20	0.2
9	3	20	0.2
10	1	200	0.2
11	2	200	0.2
12	3	200	0.2

Table 4.15: Configurations tested for the recurrent neural network.

in the previous subsection (Subsection 4.4.4), with the differences described below.

The recurrent neural network algorithm used will be the LSTM, with the activation function being the *tanh* function. No differentiation will be applied to the data. Due to the best results in the feed-forward neural networks having no differentiation or just one-lag differentiation, and also because the RMSE difference between the tests with no differentiation and one-lag differentiation is minimum, it was decided to perform the tests with no differentiation. Because the time it takes to train an LSTM is too long, reducing the number of tests is essential to obtain results in a timely manner. To do that, the hyper-parameters combinations will be reduced based on the previous best results.

The models tested will vary between one and three LSTM hidden layers, with 20 or 200 neurons each. A neural network with more than one LSTM layer is called a stacked LSTM network. It will also be tested the dropout layers with the recurrent dropout mechanism [108]. Recurrent dropout is demonstrated to achieve better results when used coupled with standard forward dropout in LSTM layers, regularizing the model weights responsible for learning short and long-term dependencies without affecting the ability to capture long-term relationships.

It will be tested stateful and non-stateful LSTM networks. A stateful LSTM network stores not only the state of the current input batch, but also the state of the previous input batches to predict the output, while a non-stateful LSTM network resets the internal states at every batch. Theoretically, a stateful LSTM is able to learn long-term dependencies, even if they are not fed directly in the same batch, while a non-stateful LSTM only learns the dependencies for each input batch. The number of lags (l) on the input will vary between 12 and 24.

The output layer of all models will be a feed-forward layer with one neuron and no activation function. A complete list of the network configurations can be found in Table 4.15.

The ten best results are in Table 4.16. The tests with 12 lags had much better results than the tests with 24 lags. The stateful forecasting results are identical to the non-stateful forecasting results, which indicates that all the information needed for the forecast for the next hour are in the previous 12 values. Another indication that no more previous information is needed is the fact that the results are very similar to the results with the feed-forward neural networks (Subsection 4.4.4).

The configuration of the best model was tested with the additional features described in Subsection 4.4.3 and the set of tests enumerated in Table 4.6. Only the best model was used for the tests, due to the LSTM training process being very time-expensive. The results can be seen in Table 4.17. Only the feature of the workday alone improved the results, in contrast with previous Subsections 4.4.3 and 4.4.4, where the best results are from the tests with only the weekday feature. All other tests showed degradation in the RMSE.

Best results	Model Number	Lag Number	Stateful	RMSE
1	9	12	False	32.61
2	4	12	False	32.65
3	10	12	True	33.10
4	3	12	True	34.39
5	11	12	True	34.43
6	10	24	True	34.60
7	6	12	True	34.62
8	9	12	True	34.72
9	2	12	False	34.79
10	10	12	False	34.88

Table 4.16: Ten best results obtained using a recurrent neural network.

Set of tests	1	2	3	4	5	6	7	8
RMSE	32.61	35.48	31.36	37.32	33.40	36.76	34.20	34.67

Table 4.17: RMSE for the tests with different combinations of additional features, for the best model for the recurrent neurl networks.

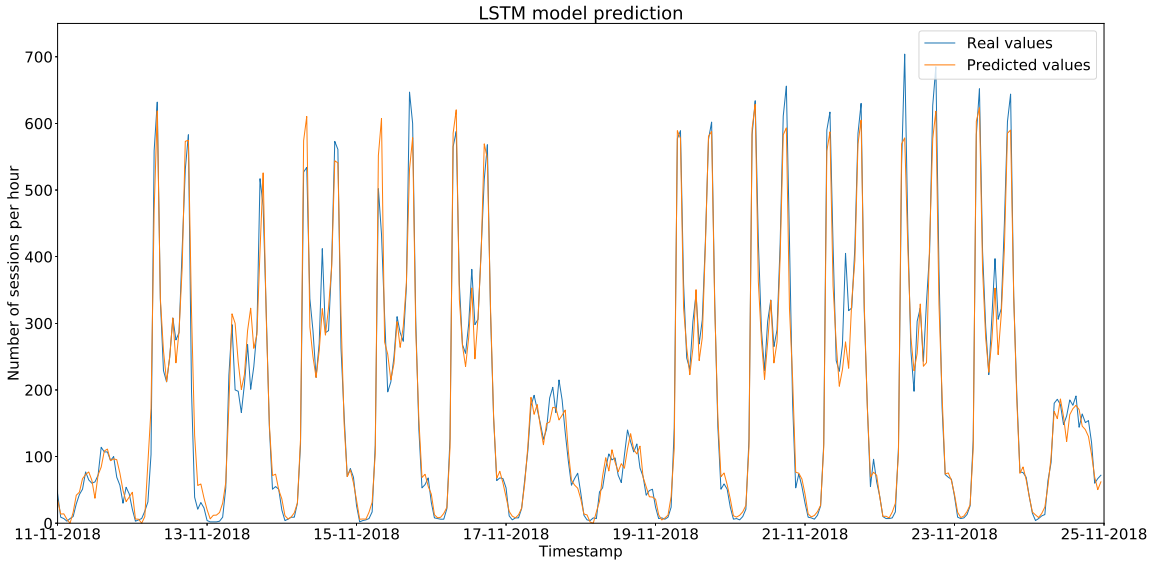

Figure 4.33: Prediction made by the best recurrent neural network model in the cross-validation set of the vehicular network, compared with the real values, with a RMSE of 31.36 sessions per hour and a MAPE of 31.72%.

The lower RMSE achieved was of 31.36 number of sessions per hour, with a MAPE of 31.72%. The predictions are in Figure 4.33. However, the best result achieved with the recurrent neural networks is still worse than the best result achieved with the feed-forward neural networks. A similar conclusion was taken in [109], where it was suggested that simple time-series prediction tasks with well-defined patterns in the data do not need to be modeled with an LSTM, because if the time-window used was large enough to capture the relevant past values of the data, a regular feed-forward neural network would be enough to predict the time-series values.

4.4.6 1-D Convolutional Neural Networks

In this subsection, 1-D convolutional layers will be tested to improve the forecast of the number of sessions in the vehicular network. A description of 1-D convolutional layers is in the Subsection 2.4.3.

Other two types of layers were used for the tests with the 1-D convolutional neural networks. A pooling layer was used to reduce the dimensionality of each feature map. Specifically, max pooling was used, which given a feature map, reduces its dimensionality by choosing only the maximum value from a dimension. For the regularization of the network, Dropout layers were not used, due to the known little effect that they have on convolutional layers. Instead, it will be used Batch Normalization [110]. Batch Normalization regularizes the network by reducing the internal covariance shift, normalizing the output of a layer by subtracting the batch mean and dividing by the batch standard deviation. During training, it is maintained an exponential moving average of the mean and variance of the data. The main goal of the Batch Normalization, however, is not regularization, but to accelerate the training of neural networks.

To test the forecasting performance of the neural networks with 1-D convolutional layers, the approach used and the description of the data transformations, the input and the output are described in Subsection 4.4.4.

The tests will be done using as input lag (l) the parameters 12 (half a day), 24 (the previous day) and 168 (the previous week). No differentiation will be applied. Several network configurations will also be tested. The convolutional layers are characterized, among other things, by the number of output filters, the kernel size, the length of the stride and the activation function.

In the tests done, the stride length of the convolutional layers used is of size one and the activation function used is ReLU. The number of output filters varies between 20 and 200 and the kernel size was set to four. The tests varied the number of convolutional hidden layers from one to four, with and without max pooling layers between convolutional layers. It was also tested networks with and without batch normalization between convolutional layers. Finally, before the output layer, it was tested two types of layers: Flatten, to reshape the output of a convolutional layer from a 3D into a 2D array by concatenating the result of multiple filters, and Global Average Pooling 2D, that does the same transformation from a 3D array into a 2D array, by selecting the maximum value for each array in the third dimension.

The models' high-level description is presented in Table 4.18. Each model described was tested with and without batch normalization, and with Flatten or Global Max Pooling layers. Due to the different input size and pooling transformation, not all network configurations could be tested with the different lag numbers.

The best five results are in Table 4.19. On average, the results with the Global Max Pooling layer instead of the Flatten layer and batch normalization have the worst results, due to their inability to learn patterns in the data. On the other hand, the tests with the Flatten layer and without batch normalization have the best results. Analyzing the lag number, the highest the lag number, the worse the overall results, with the best lag number being 12.

In Figure 4.34, it can be seen the train and cross-validation RMSE throughout the number of training epochs for the model number 4, with 12 lags as input, a Global Max Pooling layer and batch normalization. The batch normalization hurts the performance of the network in the training set, making it unable to learning usable features, maintaining a high training error. The same happens in the Figure 4.35, with the model number 11, lag number of 24, with a Flatten layer and batch normalization. The Figures 4.36 and 4.37 present the prediction error for two models without batch normalization (both models have the network number 2, with a lag number of 12; the first layer has a Global Max Pooling Layer, the second has a Flatten layer). The RMSE of the training set tends to zero, and both the train RMSE and cross-validation RMSE are much lower. Batch normalization causes the models to underfit the data.

Model number	Convolutional hidden layers	Output filters	Has Pooling Layers
1	1	20	No
2	2	20	No
3	3	20	No
4	4	20	No
5	1	200	No
6	2	200	No
7	3	200	No
8	4	200	No
9	2	20	Yes
10	3	20	Yes
11	4	20	Yes
12	2	200	Yes
13	3	200	Yes
14	4	200	Yes

Table 4.18: Different network configurations used for testing the 1-D convolutional networks. Each model described was tested with and without batch normalization and with Flatten and Global Max Pooling.

Best Results	Flatten/Global Max Pooling	Batch Normalization	Model Number	Lag	RMSE
1	Flatten	FALSE	5	12	38.59
2	Flatten	FALSE	2	12	38.87
3	Flatten	FALSE	6	12	39.6
4	Flatten	FALSE	1	12	41.41
5	Global Max Pooling	FALSE	2	12	41.71

Table 4.19: Best RMSE results in the 1-D convolutional networks tests and its hyperparameters, in descending order.

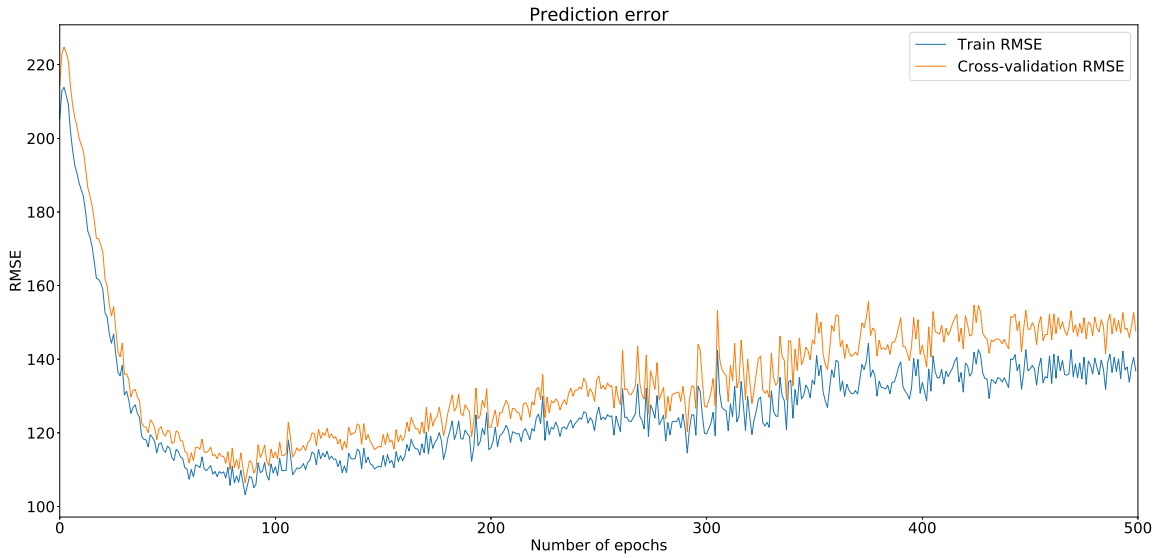

Figure 4.34: Training and cross-validation RMSE variation with the number of trained epochs for the model number 4, with a lag number of 12, with a Global Max Pooling layer and with batch normalization.

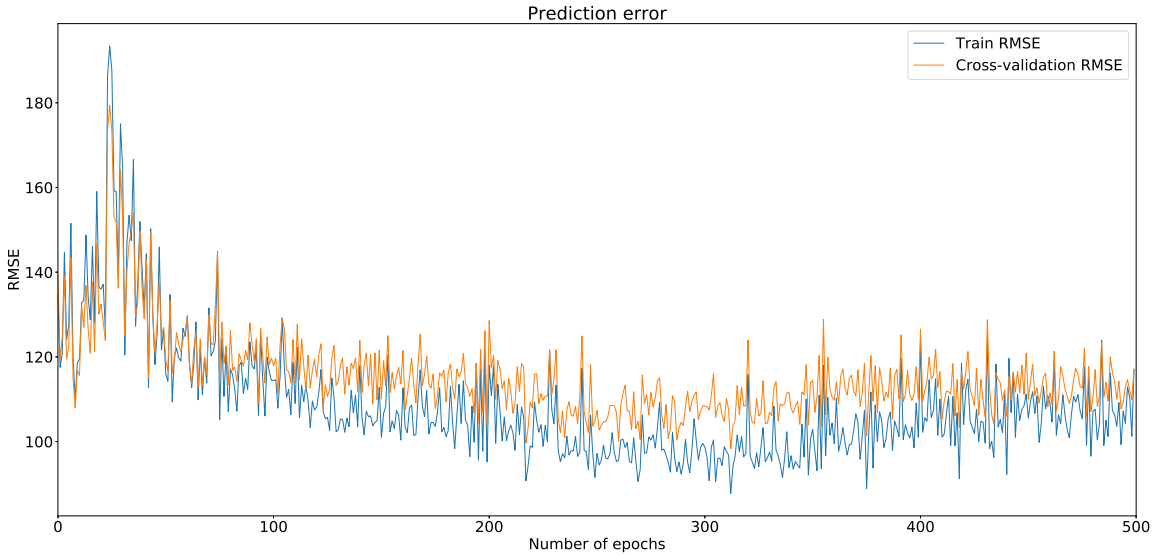

Figure 4.35: Training and cross-validation RMSE variation with the number of trained epochs for the model number 11, lag number of 24, with a Flatten layer and batch normalization.

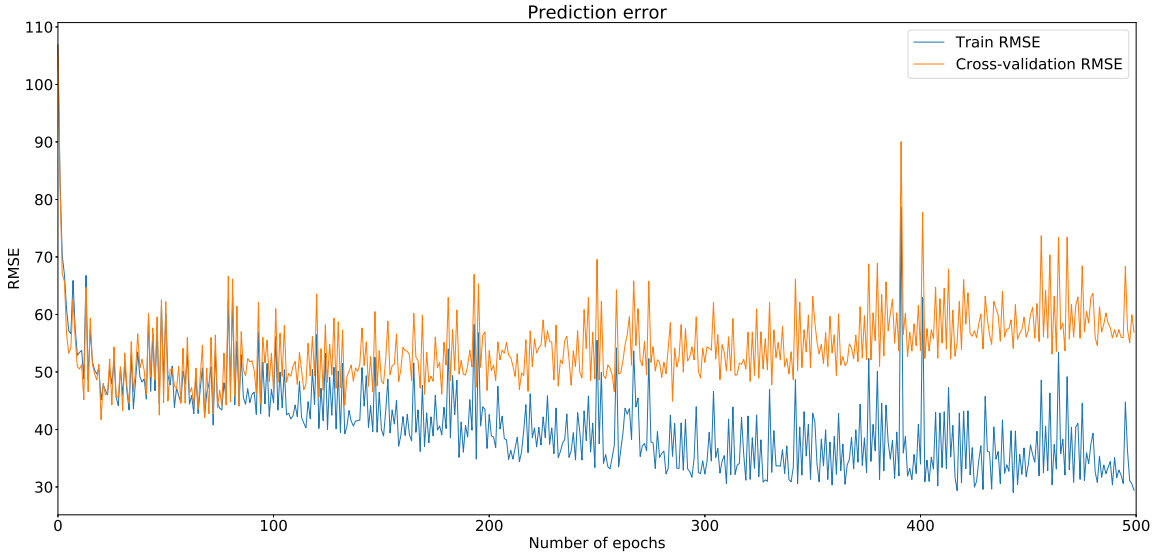

Figure 4.36: Training and cross-validation RMSE variation with the number of trained epochs for the model number 2, with a lag number of 12, with a Global Max Pooling Layer and no batch normalization.

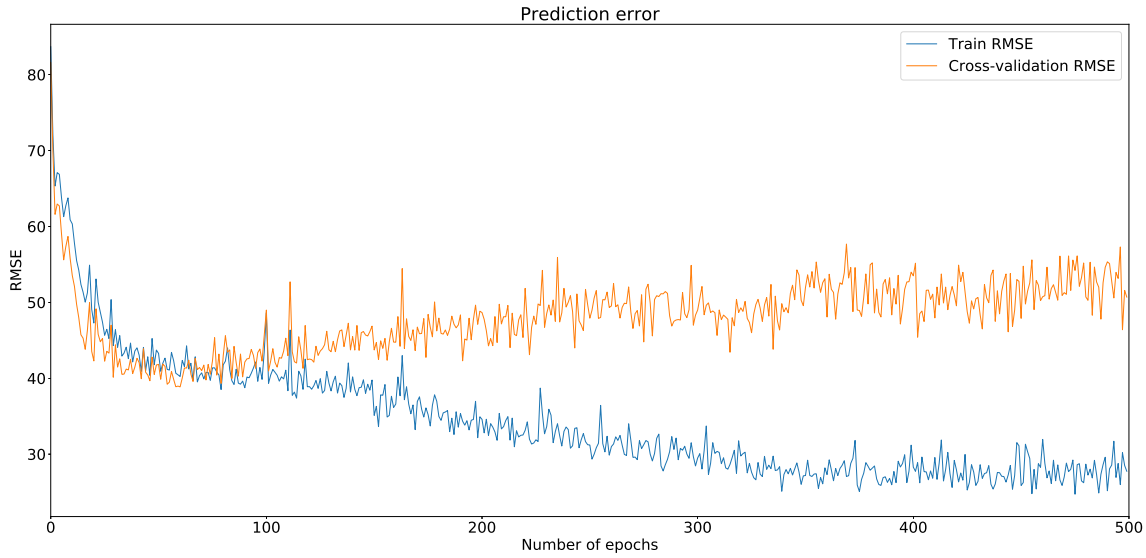

Figure 4.37: Training and cross-validation RMSE variation with the number of trained epochs for the model number 2, with a lag number of 12, with a Flatten and no batch normalization.

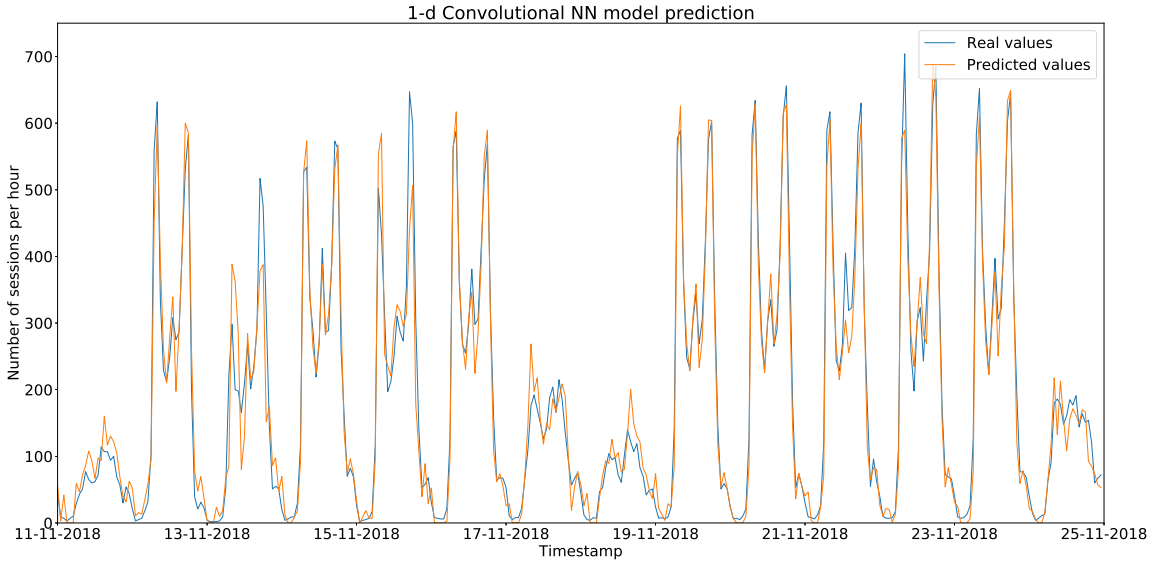

Figure 4.38: Forecasts made by the best 1-D convolutional neural network model in the cross-validation set of the vehicular network, compared with the real values, with a RMSE of 38.59 sessions per hour.

The best result was achieved on the test with the model number 5, 12 lag numbers as input, with a network with a flatten layer and without batch normalization, with an RMSE of 38.59 number of sessions per hour, corresponding to a MAPE of 42.5%, a worse result when compared to the RMSE obtained in the last Subsections 4.4.3, 4.4.4 and 4.4.5. The forecasts made by the best model are in Figure 4.38.

Given that this method is adequate only for sequence analysis, no additional features were tested.

4.4.7 Feature-based time-series forecasting

In the previous subsections, it was adopted a time-lagged approach to the construction of a forecasting model, where the previous values of the time-series were the input of the algorithms. The additional features added to the model were not time-series features, but features that described external factors that influenced the data, handmade by a domain expert. However, there is another approach to the construction of time-series predictors.

Instead of using directly the raw values from the time-series data, a set of features can be extracted from them. Those features can be chosen by a domain expert to provide better insights about the data. Using features extracted from the time-series data for the forecasting instead of using raw lagged values has some advantages:

- the features can be adjusted and optimized by a domain expert;
- the number of features is variable and sometimes it is possible to achieve similar or better results with a much lower number of features;
- usually, it has better results when there is not high autocorrelation between lags of values in the time-series data (when there is high variation in the data and it is hard to visually detect patterns).

However, this approach also has drawbacks, mainly:

- usually, it has worse results when there is high autocorrelation between lags of values in the time-series data (when it is possible to visually detect patterns in the data);
- depends heavily on the ability of the domain expert to extract the correct features for the prediction;
- the task of choosing the best features for the task can be time-consuming and non-optimal.

The time-series features extracted from the data to be the input of the learning algorithms are the following:

- maximum, minimum, average and standard deviation of the values in the previous time interval;
- values above and below the average in the previous time interval;
- median of the values in the previous time interval;
- sum of all values in the previous time interval;
- 0.25, 0.50 and 0.75 quantiles of the values in the previous time interval;
- number of values in the range $]-\infty,100[$, $[100,300[$, $[300,600[$, $[600,700[$, $[700,+\infty[$ in the previous time interval;
- position of the highest and lowest value in the previous time interval;
- raw value of the previous hour, day and week.

The time intervals chosen to extract the features are the previous day and the previous week.

To choose a set of features, it would be computationally expensive and time-consuming to test all possible combinations of features. A sub-optimal approach will be used to decide which features are the most important for the prediction task and which features only contribute to increase the RMSE.

The tests will be done using different combinations of features, to analyze the impact of each set of features in the algorithms tested. The combinations are the following:

1. all features;
2. all features with the exception of the additional features (hour of the day, day of the week and work day flag);
3. all features except the features with the time interval of a week;
4. all features except the features with the time interval of a day;
5. all features except the raw values;

Feature Set	FFNN	SVR	Linear Regression	Random Forest	Gradient Boosting	AdaBoost	XGB Regressor
1	54.17	60.77	72.56	**40.02**	46.15	85.36	44.36
2	60.86	66.45	77.66	45.78	**41.80**	75.97	42.97
3	52.31	61.26	78.25	**47.97**	48.91	88.70	50.59
4	63.95	79.21	71.27	**50.17**	53.97	90.30	54.48
5	**70.87**	99.77	109.51	74.61	87.04	117.33	87.08
6	59.99	64.13	69.94	**38.45**	46.30	81.20	45.04
7	63.21	61.95	71.14	**43.74**	44.48	77.78	45.00
8	55.95	59.29	70.75	**43.02**	46.44	83.63	46.18
9	47.45	56.27	68.28	**43.77**	46.08	79.93	44.74

Table 4.20: RMSE of the number of sessions for the different features sets tested with the best algorithms (the results for the SGD Regressor, Ridge, Lasso and ElasticNet are not represented).

6. all features except the quantiles features;
7. all features except the counts of the number of sessions in a range;
8. all features except the position of the highest and lowest number of sessions;
9. only basic features (only additional features, maximum, minimum, average, standard deviation, median, sum and raw values).

All features were normalized before being used by the algorithms. The forecasting algorithms chosen are the algorithms previously used in Subsection 4.4.3. A feed-forward neural network was also tested, due to the previously good results presented in Subsection 4.4.4. The ANN has four hidden dense layers with 200 neurons, each layer with a dropout value of 0.2 and with ReLU as the activation function. The tests follow the same approach presented in Subsection 4.4.3.

The results are presented in Table 4.20 for the best algorithms (the results for the SGD Regressor, Ridge, Lasso and ElasticNet are not represented). The forecasts heavily depend on the algorithm used. The feed-forward neural network, the SGD Regressor, the SVR, the Ridge and the Linear Regression algorithms have their best result with the minimal set of features.

The ensemble algorithms are known by their capacity of presenting good results when there is a high number of features. In this case, the ensemble algorithms tested (Random Forest, Gradient Boosting Regressor, AdaBoost Regressor and XGBoost) present better results than all other algorithms tested. On average, the Random Forest algorithm has the best average and overall result, beating all other algorithms in six of the nine tests done. The best result was achieved with the set of features without the quantile values (feature set 6), with an RMSE of 38.45 number of sessions per hour, corresponding to a MAPE of 25.32%. The forecasts made by this model are in Figure 4.39.

The features of the values of the number of sessions from the last hour, one day ago and one week ago are the most important features, because the results with the higher RMSE for all algorithms happen in the tests where those features are excluded.

The best RMSE of the feature-based forecasting tests is higher than the best result achieved up to this subsection. The most relevant features (the values from the last hour, one day ago and one week ago) show that the previous values are very important for the forecasting, mainly due to the high autocorrelation calculated between the time-series values (Section 4.2). However, it is possible to do an accurate forecasting based only on hand-made features crafted by a domain expert.

The failure of the feed-forward neural network to make accurate predictions with time-series features in comparison with the ensemble algorithms has some possible explanations. With the time-lagged approach, more information can be extracted indirectly from the input. Neural networks are more adequate for automatic feature extraction than ensemble algorithms, due to the multi-layer setup. However, when dealing with a large number of inputs that already have significant information, ensemble algorithms can converge faster and achieve lower error.

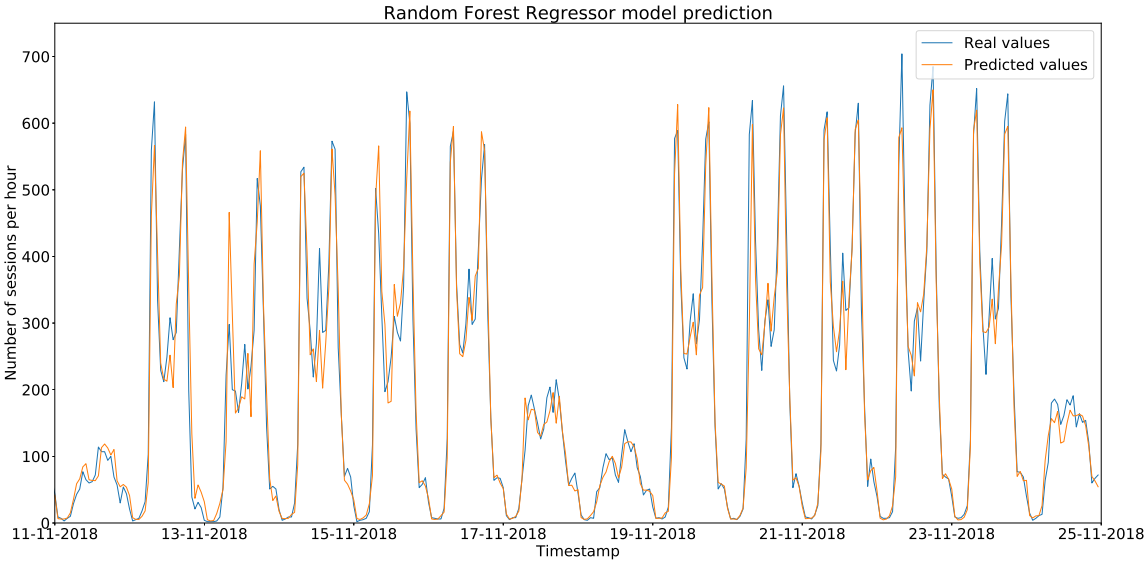

Figure 4.39: Prediction made by the Random Forest model in the cross-validation set of the vehicular network, with the set of features 6, with a RMSE of 38.45 sessions per hour.

4.4.8 Custom forecasting architectures

In this subsection, combinations of previous forecasting methods will be tested to check if the result improves. Three approaches will be taken:

- creation of neural networks with combinations of the layers previously tested;
- creation of multi-input neural networks with separate layers for each type of input (raw values and time-series features), followed by a merge operation into one global neural network;
- creation of ensemble methods by averaging the forecasts by more than one model.

The first four tests will be done using the same configurations as in Subsection 4.4.4. The input will be the number of sessions per hour in the previous 12 hours, with no differentiation applied. However, the network configuration will vary.

The first test will contain one 1-D convolutional layer with 200 output filters and a kernel size of 4, with the ReLU as activation function. It will be followed by a Flatten layer and three hidden Dense layers with 200 neurons, with the ReLU as the activation function and 0.2 of dropout.

The second test will maintain the first convolutional layer, but it will be followed by an LSTM layer with 200 neurons, *tanh* as the activation function, not stateful, with recurrent dropout and regular dropout of 0.2.

The third neural network consists of an LSTM layer with 200 neurons, *tanh* as the activation function, not stateful, with recurrent dropout and regular dropout of 0.2, followed by two hidden dense layers with 200 neurons, with the ReLU as activation function and 0.2 of dropout.

The fourth test has a neural network with one 1-D convolutional layer with 200 output filters and a kernel size of 4, with the ReLU as activation function, one LSTM layer with 200 neurons, *tanh* as the activation function, not stateful, with recurrent dropout and regular dropout of 0.2 and two hidden dense layers with 200 neurons, with the ReLU as activation function and 0.2 of dropout.

The last layer of all the networks has only one neuron with no activation function.

The results are in Table 4.21. The results show similar RMSE between them, but still worse than the best results obtained with the feed-forward neural networks (Subsection 4.4.4) or with the recurrent neural networks (Subsection 4.4.5).

Neural network hidden layers	RMSE
1 - Convnet and Dense layers	35.57
2 - Convnet and LSTM layers	35.41
3 - LSTM and Dense layers	36.63
4 - ConvNet, LSTM and Dense layers	36.66

Table 4.21: RMSE results of the first four tests in the custom forecasting architectures.

The next tests introduce additional features to try to improve the results. The first test will have as input the number of sessions per hour in the previous 12 hours and the basic features in Subsection 4.4.7, because on the tests done with additional features, the feed-forward neural network performed better with only the basic features. The network configuration is composed of four hidden dense layers with 200 neurons, with the ReLU as activation function and 0.2 of dropout. The last layer of the network has only one neuron with no activation function. The RMSE of this model was 41.45 sessions per hour, which is better than the result of the feed-forward neural network only with basic features, but worse than the result of the feed-forward neural network with only the previous values of the number of sessions per hour.

The following test splits the input features into two classes. A two-input neural network is trained: both sides of the network are composed of two hidden dense layers with 200 neurons, with the ReLU as activation function and 0.2 of dropout, and are merged into the same network with another dense layer with the same configuration. The last layer of the network has only one neuron with no activation function. While one side of the network will receive as input the number of sessions per hour in the previous 12 hours, the other input will be all features used in Subsection 4.4.7. The RMSE of this test was 36.09 sessions per hour, which improved an average of 5 sessions per hour in comparison with the previous test. However, there are still feed-forward neural network configurations (Subsection 4.4.4) with better results than the one tested.

The last test of this subcategory is similar to the previous one, but the input of the network that receives the values from the previous number of sessions does not have two hidden Dense layers. Instead, both Dense layers are replaced by an LSTM layer with 200 neurons, *tanh* as the activation function, not stateful, with recurrent dropout and regular dropout of 0.2. The RMSE of this test was 36.6 sessions per hour, which slightly increased the error in comparison with the previous result.

The last set of tests tries to improve the forecasting accuracy using model ensembles. An ensemble combines the result of several models to achieve more accurate forecasts. The best feed-forward network model with only the previous number of sessions per hour as input and external features (feed-forward neural network model number 22, with one-lag differentiation, a lag number of 12 and the weekday as additional feature, as seen in Subsection 4.4.4) was combined with the best algorithm that has as input only time-series features and external features (Random Forest with all features except quantile features). The combination of the two models was done with an average of the forecasts. The RMSE was 28.44 sessions per hour, a new best record, and the MAPE was also the best MAPE in the tests, 19.47%.

A way to improve the results is to try to add another predictor. Another test was made with both predictors used before, and with the best model using the recurrent neural networks (recurrent neural network model number 9, not stateful, with a lag number of 12 and with the workday as an additional feature, as seen in Subsection 4.4.5). The output is now the average of the forecasts of the three models. The RMSE was 26.22 sessions per hour, which overtakes the previous best result, and the MAPE was 21.47%, slightly higher than the best result. The forecasts made by this model are in Figure 4.40.

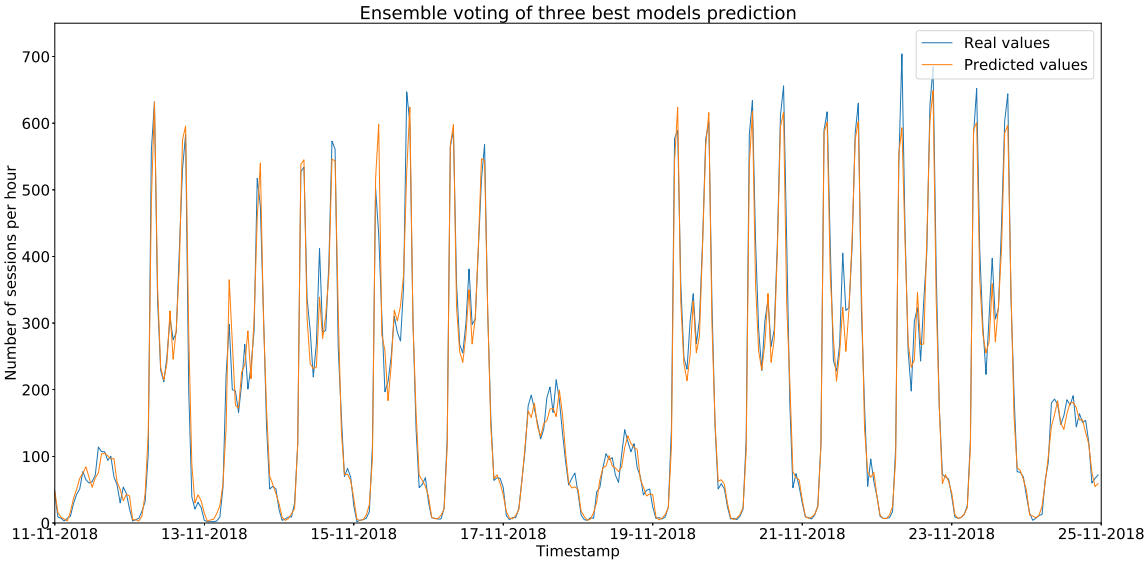

Figure 4.40: Forecasts made by the ensemble model with three predictors in the cross-validation set of the vehicular network, compared with the real values, with a RMSE of 26.22 sessions per hour.

4.4.9 Final test prediction

The final tests will be done by training the algorithms with best results with the training and cross-validation sets, and testing with the test set. The train, cross-validation and test sets can be seen in the Figure 4.21. Five different models will be used for the tests:

1. Feed-forward neural network with the network model number 22, one-lag differentiation, with 12 lags as input and the weekday as additional feature (the network configuration with best result in the Subsection 4.4.4);
2. Recurrent neural network model number 9, not stateful, with 12 lags as input and with the workday as additional feature (the network configuration with the best result in the Subsection 4.4.5);
3. Random Forest with all features except quantiles features (best model where the input is feature-based and not time-lagged based, Subsection 4.4.7);
4. Ensemble by average of the forecasts of the models 1 and 3;
5. Ensemble by average of the forecasts of the models 1, 2 and 3.

The results of the tests can be seen in the Table 4.22, and they show that the RMSE improved from the previous best results (previously RMSE of 29.99, 31.36, 38.45, 28.44 and 26.82 sessions per hour, respectively), due to the larger set of training data. For a small margin, the recurrent neural network performs better than the feed-forward neural network in both the RMSE and MAPE, as opposed to the tests in the cross-validation set.

Just like in the previous tests with the cross-validation set, the Random Forest with hand-made features performs worse than both the neural networks tested, but it has the best MAPE of the test. The lowest MAPE happens due to the higher accuracy when the number of sessions is low (the MAPE weights higher the observations with lower absolute value, Equation 4.5).

The model ensembles, just like in the previous tests (Subsection 4.4.8), outperform the result of the models separately. The best result was achieved with an ensemble of the feed-forward neural network, the recurrent neural network and the random forest algorithm, with an RMSE of 22.11. However,

Test number	RMSE (number of sessions per hour)	MAPE (%)
1	26.59	28.09
2	23.40	17.06
3	29.10	16.28
4	24.10	19.60
5	22.11	18.94

Table 4.22: RMSE and MAPE results with the test set of the best five tests.

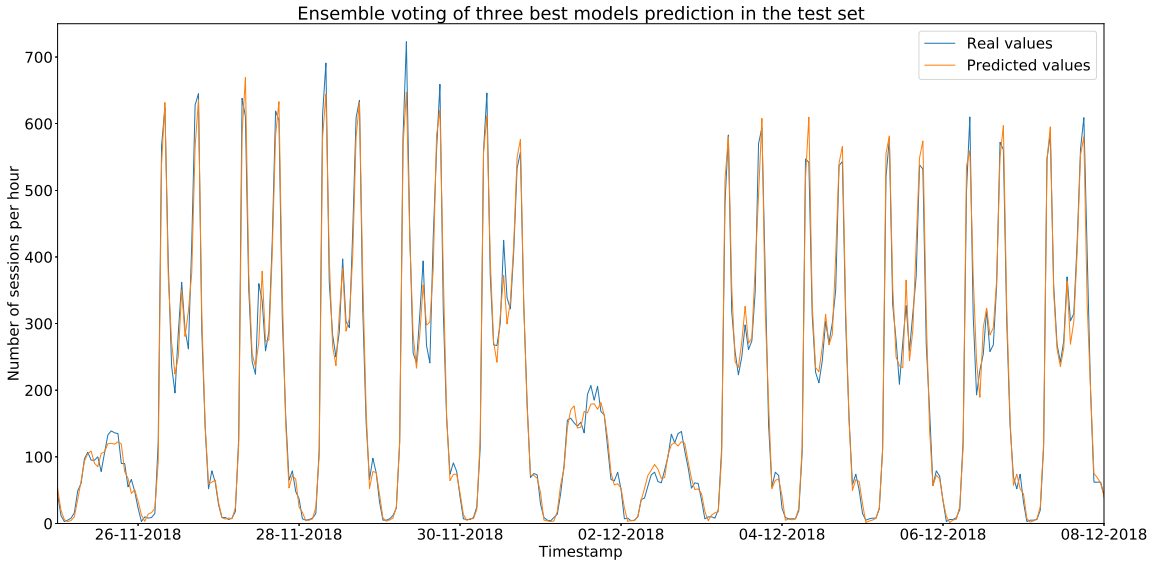

Figure 4.41: Forecasts made by the ensemble model with three predictors in the test set of the vehicular network, compared with the real values, with a RMSE of 22.11 sessions per hour.

lowest MAPE is achieved by the random forest algorithm (MAPE of 16.28%). The forecasts of the best ensemble model are in Figure 4.41.

4.5 Discussion

This chapter started with an exploratory data analysis before the tests' implementation and results' analysis. The exploratory data analysis encompasses Sections 4.1, 4.2 and 4.3. In these sections, the data distribution was discussed, the outliers were removed, data transformations were made and statistical tests were conducted to better understand the data and to adequate the learning algorithms and the hyper-parameters to the data. The data exploration step is essential for the resolution of a task using machine learning methods, for testing statistical hypotheses and to assure the quality of the data. This step cannot be automated, because it is highly subjective and it needs a human to critically analyze the data, usually using graphic visualizations, such as plots, charts, histograms and other techniques.

In the Section 4.4, it was tested various approaches to the forecast of the number of sessions in a network. The persistence model, the ARIMA model, classical machine learning models, feed-forward neural networks, recurrent neural networks and 1-D convolutional neural networks were tested with the previous values of the number of sessions as input to forecast the number of sessions in the next hour. Other machine learning algorithms were also tested, but with features extracted from the time-series. It was concluded that the approaches with the best results were the feed-forward neural networks and the recurrent neural networks. Specifically, an ensemble algorithm with both neural networks and with

a Random Forest with features extracted from the time-series data was the model with the best result. The best RMSE achieved was 22.11 sessions per hour, which demonstrates the effectiveness of the current state of the art machine learning approaches to accurately forecast a metric.

This chapter also adressed different data transformations to the data to make it as close as possible to a stationary distribution. Various types of differentiation were done, and it was concluded that the best differentiation value depends on the algorithm: for the persistence and ARIMA algorithms, the best differentiation value was 168; for the classical machine learning algorithms and neural networks, the best differentiation value was 0 (no differentiation).

Other features were added to the models, such as the hour of the day, the day of the week and the workday flag to check if it was a workday. The results showed that, even though the RMSE improved with some additional features, their impact was not big in the final results. However, in the dataset there were only two holidays in workdays (the 5^{th} of October and the 1^{st} of November), both in the train set. Due to that, the effect of the additional features in the forecasting in a workday with a holiday was not tested, because there were not any holidays in a workday in the cross-validation set neither on the test set. It is expected that, when tested on a workday with a holiday, the additional feature of workday has a higher impact than what was detected on the tests.

The best RMSEs of the Feed-forward Neural Networks and LSTMs were very close in the cross-validation set and in the test set. While in the cross-validation tests the Feed-forward Neural Networks achieved a better result, in the final tests the LSTMs had a lower RMSE. Because more tests were done with the Feed-forward Neural Networks than with the LSTM on the cross-validation set, the result is bigger, and the probability of achieving a lower RMSE is higher. As discussed at the end of the Subsection 4.4.5, because the results achieved with both methods are similar, it indicates that the data can be predicted taking into account only the previous input values, and the use of an LSTM for this dataset is not required, since a Feed-forward Neural Network is faster to train, faster to forecasat and less computationally expensive.

This analysis was made for the number of sessions in a network, but it can be extended for any network metric that has strong seasonality patterns in the data. For the network operator, it is crucial to forecast the network metrics, to be able to pro-actively manage the network. In a 5G network, where network slicing and resource allocation will be important characteristics of the network, forecasting becomes essential to the 5G management frameworks.

In the next chapter, the same dataset of the vehicular network will be used to forecast the number of sessions. However, instead of forecasting accurately the number of sessions, two different goals will be proposed for specific scenarios. In the first scenario, for a network operator, it can be more important to forecast the peaks in the network metrics than to forecast lower observations, because of resource management. The other scenario is to forecast accurately in anomalous time intervals, such as anomalous weeks (for example, holidays, football games, music festivals or special events), where the network usage varies from the regular usage.

5

Forecasting of Specific Characteristics in Network Metrics

In a real network, the goal of the forecast may be concerned with specific characteristics in network metrics: for example, in the case of resource management, the network operator is more interested in forecasting sudden increases in the network metrics, to be able to adapt the network resources to those unusual events before they happen. Another example is when there is an anomalous time-interval where the network metrics behave different from normal, such as when there is a music festival, a football game or the holidays begun, and the network traffic is different from the usual. It can be created models specifically accurate for each scenario, and they can be used simultaneously with other models for the same network metric. While some models are more accurate in forecasting the exact number of sessions in the next hour, other models may not be so good at that task, but more reliable in identifying specific events that will change the normal behavior of the network metrics.

In this chapter, two different scenarios are proposed to adapt the forecasting objective for different use cases. The first scenario is to forecast accurately higher value observations, to be able to detect the peak values more reliably. The second scenario is to forecast accurately in anomalous time intervals. Both scenarios are commonly faced when the forecasting models are deployed into production. The best model configurations from the previous chapter will be used in this chapter to avoid running time-consuming tests for all hyper-parameters.

5.1 Forecasting accurately higher value observations

Commonly, it is more important to forecast accurately higher value observations than lower ones. In the use case of forecasting the number of sessions in a 5G network, it can be more important to predict higher value observations due to the need of resource allocation to handle more sessions in the network or due to the need of resource deallocation, to allow the network management to take resources from one slice and give them to another. For those tasks, forecasting accurately the number of sessions is not so important if the number of sessions is approximately constant, because the resource allocation or deallocation will only happen for big variations in the network metrics.

To do that, the performance metrics and the objective function both need to be optimized to measure accurately the errors on the higher observation values and to adjust the weight of the errors according to the observation values. In this section, two new performance metrics will be proposed to measure the error with higher weight in higher observations and in higher variations in the data. With the best models tested in the previous chapter (Subsection 4.4.9), the new performance metrics will be

calculated. Then, the learning process is adapted to minimize the new performance metrics. Finally, a custom objective function that has higher error for higher observations is proposed and tested in the neural network algorithms.

5.1.1 Performance Metrics

The performance metrics used previously for forecasting were the RMSE (Equation 4.4) and MAPE (Equation 4.5), explained in Section 4.4. However, none of these metrics captures the error with a higher weight on the higher observation values. The RMSE measures the absolute error in the same scale independently of the observation value. The MAPE, for the reasons explained in Section 4.4, gives higher weight to the low observation values, having the opposite effect that is desired to measure the errors on the high observation values.

To measure the errors on the higher observation values, two novel performance metrics will be proposed. The Root Mean Squared Error above Threshold (RMSET) calculates the RMSE only above a defined threshold. In that way, it is possible to measure accurately the RMSE only for higher observation values. Equation 5.1 uses the Iverson bracket notation [111] to describe the mathematical formula of the metric, where y is the vector of observed values, $\hat{y}$ is the vector with the forecasts and t is a vector with the threshold value for each observation. The RMSE is only taken into account when $y_j > t_j$. The sizes of the three aforementioned vectors must be the same (n).

$$RMSET(y, \hat{y}, t) = \sqrt{\frac{1}{\sum_{j=1}^{n} 1[y_j > t_j]} \sum_{j=1}^{n} (y_j - \hat{y}_j)^2 [y_j > t_j]} \tag{5.1}$$

While the RMSET measures the errors of the observations above a certain threshold, it is also important to measure the differences between two consecutive observations, because it is important to understand when there are big variations in the time-series data to react pro-actively to those expected variations.

The Differentiated Root Mean Squared Error above Threshold (DRMSET) measures the RMSE taking into account two consecutive time-series values with a difference higher than a threshold. The formula for this performance metric is in Equation 5.2, where y is the vector of observed values, $\hat{y}$ is the vector with the forecasts, t is a vector with the threshold value for each observation and n is the size of the three vectors (y, $\hat{y}$ and n).

$$DRMSET(y, \hat{y}, t) = \sqrt{\frac{1}{\sum_{j=2}^{n} 1[\|y_j - y_{j-1}\| > t_{j-1}]} \sum_{j=2}^{n} (y_j - \hat{y}_j)^2 [\|y_j - y_{j-1}\| > t_{j-1}]} \tag{5.2}$$

The DRMSET vector will have size $n - 1$. Because of the difference between two points, the first point of the dataset must be discarded, because there is no previous point to compute the difference.

In the tests done, it is needed to define the threshold value for both equations. The threshold needs to be adaptive throughout time, because in the analyzed data there are days with a higher number of sessions than others, and a fixed threshold could cause the error of some days to have more weight to the final metric than the error of other days. It can be taken advantage of the fact that, in these tests, the performance metric can be calculated *a-posteriori*, which means that future observations can be used to calculate the threshold of previous observations.

To give similar weight to the errors among different days, the threshold used for both performance metrics will be based on the maximum number of sessions per day. For the number of sessions of each day, the threshold will be the maximum value of that day multiplied by a factor ranging from between

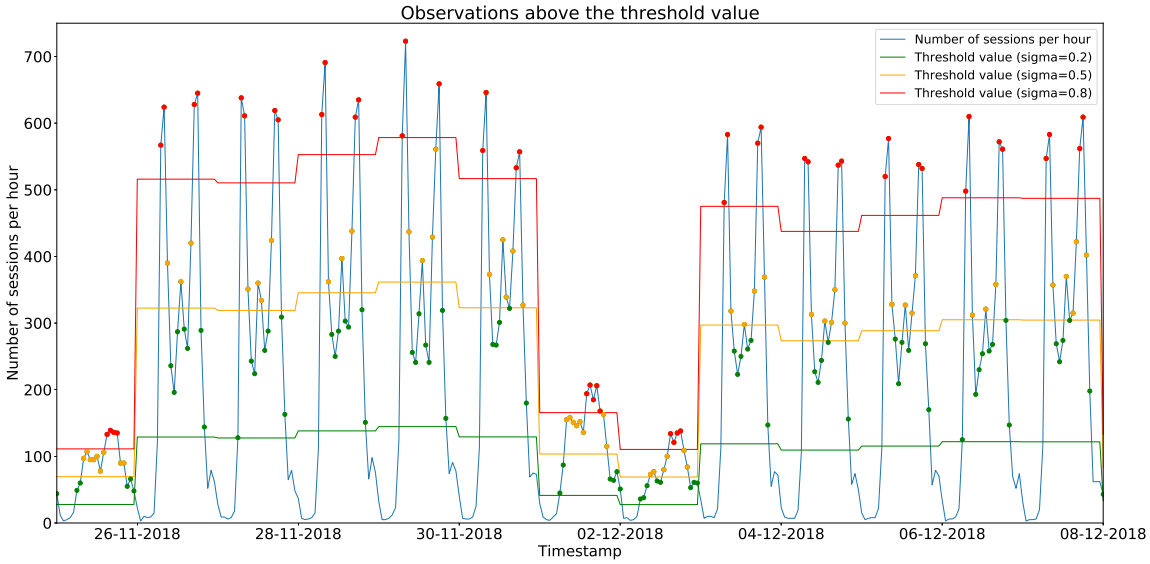

Figure 5.1: Plot with the observations of the test set, showing the thresholds with $\sigma = 0.2, \sigma = 0.5$ and $\sigma = 0.8$. The dots are the observations used to calculate the RMSET above each threshold.

0 and 1. Equation 5.3 shows the formula for the threshold calculations, where t_{day} represents the threshold for all observations in that day, x_{day} represents the observations on that day and σ is the multiplicative factor.

$$t_{day}(x_{day}, \sigma) = max(x_{day}) * \sigma, \sigma \in \mathbb{R} \wedge 0 < \sigma < 1 \tag{5.3}$$

The σ values chosen were 0.2, 0.5 and 0.8. Those values were chosen arbitrarily between 0 and 1, to understand the variation of the performance metrics for different values of the threshold.

Figure 5.1 shows the threshold values for each day of the test set. The dots are the observations used to calculate the RMSET. All dots above a threshold are part of the calculations for that threshold (for the threshold $\sigma = 0.2$, the dots of the thresholds $\sigma = 0.5$ and $\sigma = 0.8$ are also part of the calculations for the threshold $\sigma = 0.2$).

Figure 5.2 also shows the threshold values for each day of the test set, but the dots are the observations used to calculate the DRMSET (the observations where the absolute difference with the previous point is above the threshold value). Because there is only one observation where the difference of two consecutive values is higher than 80% of the maximum value of the day, for the DRMSET only the σ values of 0.2 and 0.5 will be used in the tests.

The models obtained in the tests done in Subsection 4.4.9 were used to calculate the new performance metrics. A summary of the five tests configuration is in Subsection 4.4.9. The results are in Table 5.1. As expected, there is an increase in the RMSET and in the DRMSET when the threshold is higher, mainly due to most number of sessions having low value, and high values are hard to forecast because they are less common. For the same value of σ, the DRMSET is higher than the RMSET, showing that it is harder to forecast observations with big differences than observations with higher values. The test number 5 has the lowest RMSET values, being the best model for almost all performance metrics, with the exception of the MAPE and the DRMSET with σ=0.2.

For the neural networks (tests 1, 2, 4 and 5, as described in Subsection 4.4.9), the results were optimized for the lowest RMSE value, with an early stopping approach. However, with the new performance metrics, it is possible to store the model that minimizes each new performance metric,

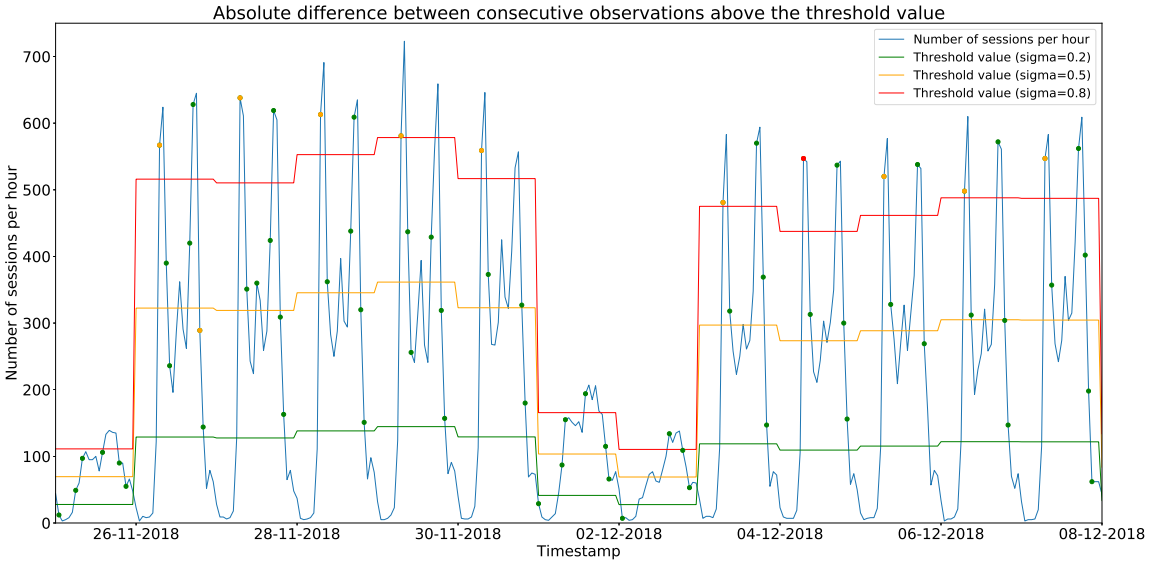

Figure 5.2: Plot with the observations of the test set, showing the thresholds with $\sigma = 0.2, \sigma = 0.5$ and $\sigma = 0.8$. The dots are the observations used to calculate the DRMSET above each threshold.

Test Number	RMSE	MAPE	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2	DRMSET σ=0.5
1	26.59	28.09	34.85	39.54	41.62	38.24	50.90
2	23.40	17.06	36.20	37.63	42.65	41.35	57.28
3	29.10	16.28	27.96	32.83	33.45	28.86	40.92
4	24.10	19.60	30.34	33.08	36.25	36.08	43.76
5	22.11	18.94	27.29	30.38	32.53	31.4	39.66

Table 5.1: New performance metric results with the test set of the best five tests.

Test Number	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2	DRMSET σ=0.5
1	31.84	34.24	32.24	33.96	28.92
2	29.26	33.85	32.67	29.59	37.67

Table 5.2: Performance metric results for the first two tests optimized for each performance metric.

instead of choosing the model that minimizes the overall RMSE. Table 5.2 shows the lowest value for each new performance metric during the epochs of the training of the models for tests 1 and 2. As can be observed, it can be created models with a more accurate result for a higher number of sessions.

Tests 4 and 5 are ensemble forecasts of the previous models. Because there are different models according to different performance metrics that are chosen to optimize, the performance metric RMSET (σ=0.8) will be used for the optimization of the models for building the ensemble. The results are in the Table 5.3. The RMSET (σ=0.8) had a result of 27.75 number of sessions in the test number 5, the lowest on the tests made in the test set. The forecasts made can be seen in Figure 5.3.

Test Number	RMSE	MAPE	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2	DRMSET σ=0.5
4	23.50	17.56	28.50	31.74	28.72	33.25	32.39
5	23.04	18.87	34.14	32.88	27.79	39.90	41.67

Table 5.3: Performance metric results for the last two tests optimized for the metric RMSET (σ=0.8).

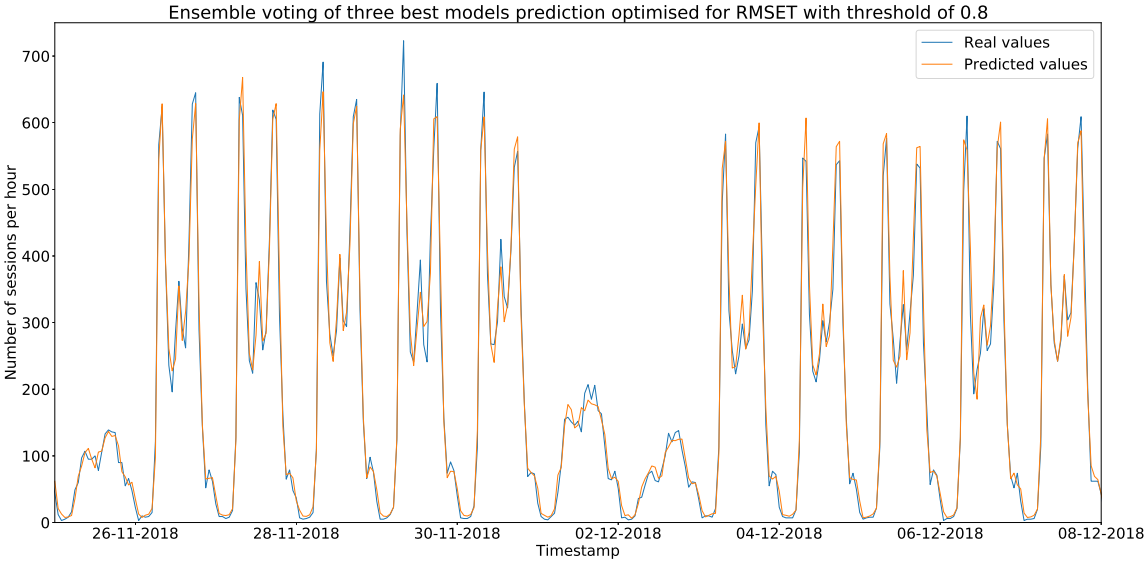

Figure 5.3: Forecasts made by an ensemble of the three models optimized for the lowest RMSET (σ=0.8) in the test set compared with the real values.

5.1.2 Custom Objective Function

To further improve the performance metric RMSET (σ=0.8), a new approach is tested in this subsection. The objective function will be modified to try to reduce the error in the new performance metrics.

The previous objective function was RMSE. The custom objective function must forecast better the observations with a higher value than the observations with a lower value. It is still desirable to penalize the absolute error between the observations and the forecasts, including the observations with lower values, only with a smaller weight than the observations with high values.

The objective function is described in Equation 5.4. The W is the new weighting function that will be used for weighting every absolute difference in the observations. The weighting function will output a real number between zero and one. The w_{cost} is a constant value that multiplies by the W output, to increase or decrease the importance of the weighting function. y is the vector of observed values, $\hat{y}$ is the vector with the forecasts and n is the size of both vectors (y and $\hat{y}$).

$$L(y, \hat{y}, W, w_{cost}) = \sqrt{\frac{1}{n}\sum_{j=1}^{n} W(y_j) * w_{cost} * (y_j - \hat{y}_j)^2} \tag{5.4}$$

Because the goal is to increase the penalization for observations with a higher value, the weighting function tested will be a linear function, a ratio between the value of the observation and the maximum value of the observations for that day (Equation 5.5). The weighting function will provide higher weight to the forecasting error when the observation is closer to the maximum value of that day. The

Test Number	RMSE	MAPE	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2	DRMSET σ=0.5
1	25.90	18.75	30.90	33.06	30.53	35.17	30.98
2	23.61	16.98	28.50	33.13	31.29	29.63	38.79

Table 5.4: Performance metric results for the feed-forward neural network and recurrent neural network tests optimized for each metric with the custom objective function.

Test Number	RMSE	MAPE	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2	DRMSET σ=0.5
4	26.19	21.03	31.80	34.57	31.52	41.61	41.26
5	23.87	25.52	28.78	32.12	29.63	34.74	39.98

Table 5.5: Performance metric results for the ensemble tests optimized for each metric with the custom objective function.

w_{cost} will be set to 2.

$$W(x_j) = \frac{x_j}{max(x_{day})} \tag{5.5}$$

There are two major disadvantages of this weighting function. It is needed to know *a priori* the maximum observation for that day, making the weighting function unreliable to use in real-time. A possible solution for this is to, instead of calculating the maximum value of the observations on the current day, calculate the maximum value of the observations in the last 24 hours, with a rolling window.

The second disadvantage in the use of this weighting function is the division by zero error. If the current day observations are all zero-valued, the maximum value of the observations of the day is zero, and the result of the weighting function is indeterminate. However, it is admitted that all days in the dataset have at least one non-zero observation per day.

The feed-forward neural network and the recurrent neural network with the configurations used in the Subsection 4.4.9 will be used to test the custom objective function. The Random Forest implementation does not allow to implement custom objective functions. Table 5.4 shows the lowest value for each performance metric throughout the epochs of the training of the models with the custom objective function.

The results are similar to the previous results with the RMSE objective function (Table 5.2). The performance metric RMSET marginally improved in comparison with the previous results around 1.5 sessions per hour. However, the tests with the ensemble approaches, in Table 5.5, show that the ensemble results perform marginally worse than the ensemble results in Table 5.3.

While the custom objective function seems to improve independently the models' accuracy on observations with high values, the ensemble of those models still performs better with the standard RMSE objective function than with the custom objective function for observations with high values. It can be concluded that the custom objective function did not improve significantly the performance of the observations with high values. One possible reason for this behavior is that the custom objective function applies a scaling transformation to the previous objective function, not changing the *minima* of the function relative to the other points, but scaling the other points in a way that the derivatives of the function, used by gradient descent for learning, have higher absolute value for observations with higher values than for observations with lower value. This causes the loss function to learn faster (in fewer epochs) to adapt to observations with higher values.

Feed-forward Network training epoch	RMSE	MAPE	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2	DRMSET σ=0.5
RMSE Objective function	**58**	**336**	**58**	**58**	121	**87**	361
Custom Objective function	211	375	78	203	**90**	456	**97**

Table 5.6: Number of training iterations to reach the lowest error value on the test set with the feed-forward neural network, comparing both objective functions.

Recurrent Network training epoch	RMSE	MAPE	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2	DRMSET σ=0.5
RMSE Objective function	95	499	95	95	110	110	264
Custom Objective function	**44**	**388**	**44**	**44**	**22**	**44**	**22**

Table 5.7: Number of training iterations to reach the lowest error value on the test set with the recurrent neural network, comparing both objective functions.

Table 5.6 shows the epoch number of the feed-forward neural network model with the best result for that performance metric. The bold values are the lower number of epochs for a performance metric in both trainings with different objective functions, which means that the algorithm reached to a better model earlier. As it can be seen, while the model with a standard RMSE cost function reached a lower value earlier, the custom objective function has a lower number of iterations for the performance metrics that measure the higher values.

Table 5.7 shows the same tests for the recurrent neural network. The results show that the custom objective function needs a lower number of iterations to reach a better model for all the performance metrics.

These results show that the custom objective function, regardless of not improving significantly the forecasts on the observations with higher values, speeds up the process of learning on these observations. Because in a 5G management system there could be many predictors training simultaneously, speeding up the learning process can be valuable to consume less computational resources and to save costs in operational expenditures.

5.2 Forecasting accurately on anomalous time periods

The tested models can accurately forecast the number of sessions per hour in a regular time period. However, sometimes there are anomalous time periods, where the number of sessions varies in an unexpected way, or the number of sessions follows the same relative pattern between days, but with a trend not present in the training data. It is important to understand the best way to build a model the more robust possible to anomalous time periods.

The task of forecasting accurately on anomalous time periods is hard by definition. It can be portrayed as similar to the *black swan problem*, an event that is never predicted because it never happened in the training set[1]. The best approaches for forecasting are based on the past history of the time-series data. If the train and test set come from mismatched data distributions, the optimization function on the train set is optimizing on a different data distribution, causing the forecasts on the test set to be inaccurate. To solve that problem, the train and the test set must be normalized to be as similar as possible, for the training step optimization to provide accurate forecasts on the test set.

[1] `https://en.wikipedia.org/wiki/Black_swan_theory`

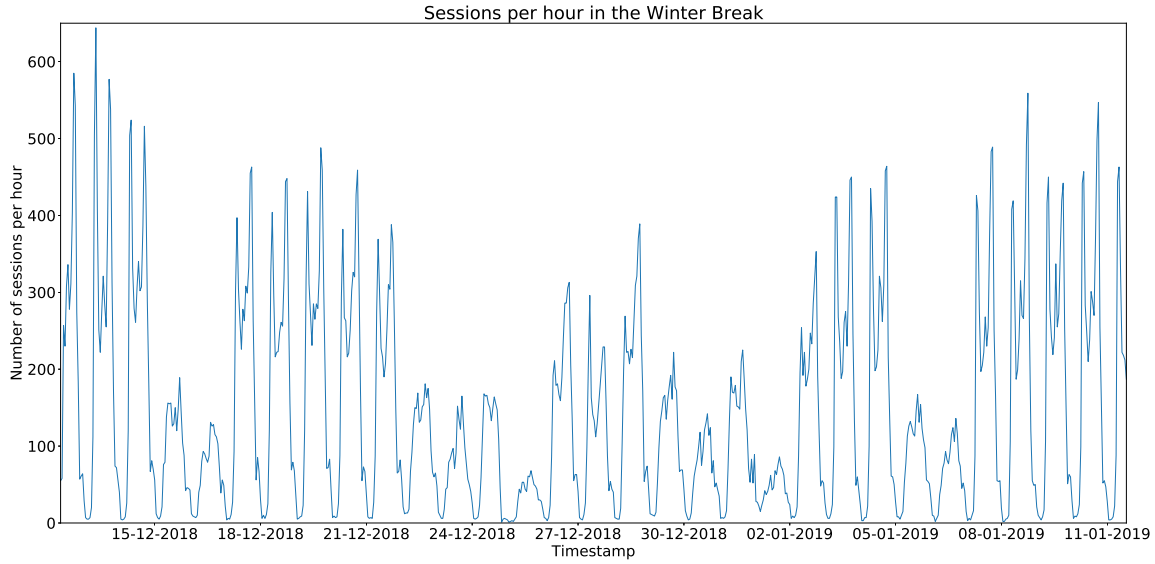

Figure 5.4: Number of sessions per hour in the Porto public buses from the 12^{th} of December of 2018, at 08:00:00, to the 11^{th} of January of 2019, at 13:00:00.

To test the previous models in an anomalous time period, data from the vehicular dataset described in Section 4.1 was used (Figure 4.3). The test set will be from the period of the 12^{th} of December of 2018, at 08:00:00, to the 11^{th} of January of 2019, at 13:00:00. In this time period, the number of sessions has a consistent decrease followed by an increase due to the Winter break in Porto (Figure 5.4).

A seasonal decomposition using moving averages was done using an additive model with a frequency of one week (Figure 5.5). The trend component shows the general decrease and increase in the number of sessions per hour throughout the days, while the seasonality component is similar for different weeks. This observation confirms that the data patterns are the same across different weeks, with the trend causing the main variations in the data. The best way to transform the train and test set for them to become similar is to apply a transformation that eliminates most of the trend.

The autocorrelation function of the test data (Figure 5.6) shows that there is a high autocorrelation for the lag 1 and for every lag multiple of 24. When compared with the autocorrelation function of the training data (Figure 4.8), the results are similar. However, there is a higher correlation for lag 1 and lower correlation for lag of 168 (one week). This indicates that a lag of 1 may be adequate for differentiation.

The tests follow the same configurations as described in the Subsection 4.4.4. The same models and hyperparameters tested in the Subsection 4.4.9 will be used to forecast in the anomalous time period. However, there are more differences besides the different train and test sets described before. Prior to data normalization, two ways of transforming the input values will be tested, to try to reduce the mismatch between the train and test set. The first set of tests will be done with the original values.

The second set of tests will be done with one-lag differentiation, as described in Equation 4.3 with $lag = 1$. One-lag differentiation tries to reduce the mismatch between the training and test sets by applying a differentiation operation to the data. The differentiation removes the trend from the data, changing the forecasting task to, instead of forecasting the absolute value in the next hour, it forecasts the difference between the previous and the next time-series points. In this way, even if the train and test data have different absolute values, the datasets with differentiation can be similar if the absolute

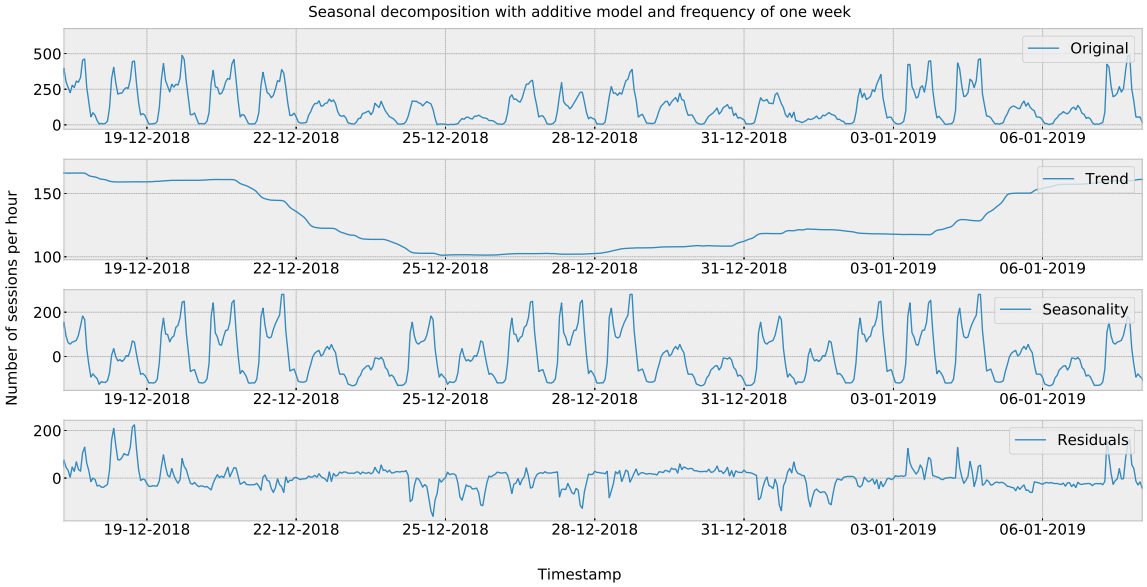

Figure 5.5: Seasonal decomposition using moving averages with additive model and frequency of one week. The first plot shows the original data; the second plot shows the captured trend; the third plot shows the seasonality; the last plot shows the residuals.

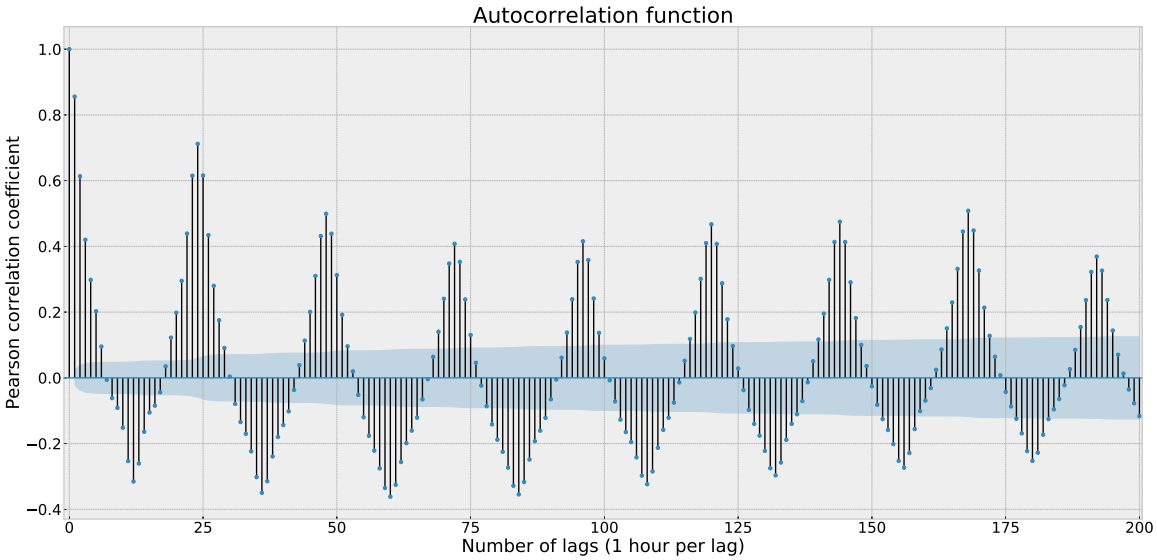

Figure 5.6: Autocorrelation plot of the time series data.

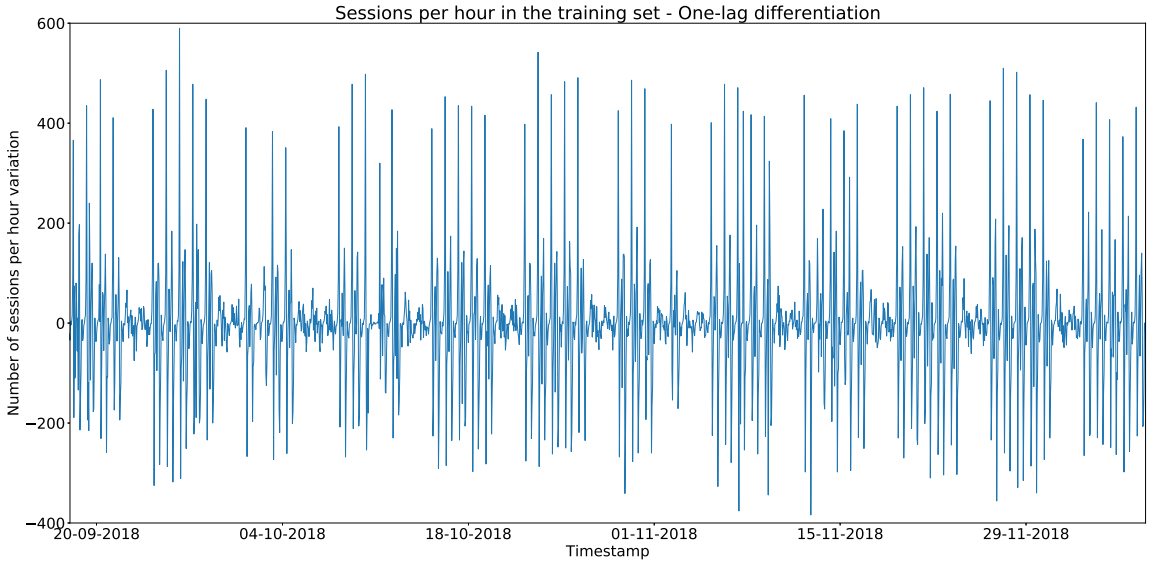

Figure 5.7: Train set with one-lag differentiation applied.

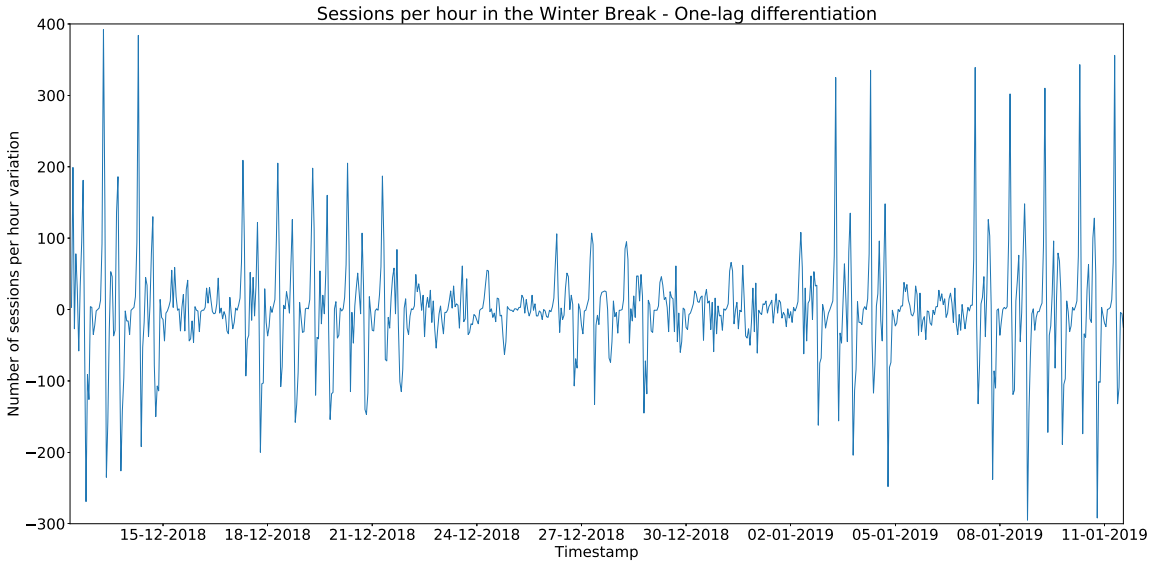

Figure 5.8: Test set with one-lag differentiation applied.

differences between the time-series points is close.

The train and test datasets with one-lag differentiation can be seen in Figures 5.7 and 5.8, respectively. While the differences between the two sets seem to be lower than with the original values, there is still a data mismatch, mainly in the middle of the test set. Overall, the training set has higher differences between consecutive time-series points than the differences in the test set.

The third set of tests will be done with a multiplicative transformation instead of an additive one. With the one-lag differentiation, if the absolute differences between the time-series points are close, the differentiation will perform well. However, if all time-series points are scaled to a factor, the absolute difference between consecutive time-series points will not be the same in the train and test sets, but the ratio between them will. Both transformations applied in this section depend only on the datasets and the type of data mismatch between the train and test data to provide good results. The multiplicative

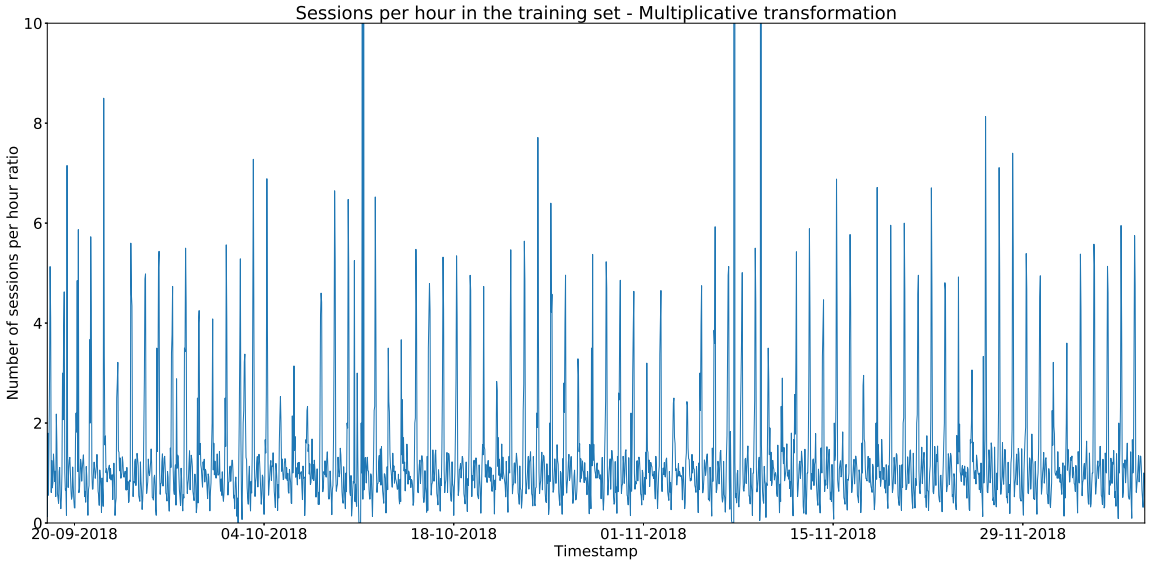

Figure 5.9: Training set with one-lag multiplicative transformation applied.

transformation is described in Equation 5.6, with $lag = 1$.

$$transformed(t) = \begin{cases} \frac{observation(t)}{observation(t-lag)}, & \text{if } observation(t-lag) > 0 \\ observation(t) & \text{if } observation(t-lag) = 0 \end{cases} \tag{5.6}$$

The equation takes into account a special case when the previous observation is 0 and the ratio cannot be made. In that case, the algorithm must predict the absolute value of the next observation. This special case does not have a very negative impact on the forecasting algorithms because the number of zero-valued observations in the training set and in test set are very reduced (in the training set, 14 points of 1992 are zero; in the test set, there are no points with the value zero), and the algorithms used are robust to random noise in the input [112]. Besides, the algorithms used are able to model non-linearities in the input, and thus can learn to model this special case.

Figures 5.9 and 5.10 show the values after the multiplicative transformation applied to the training and test set, respectively. It can be seen that, on average, the ratio of the number of sessions per hour between two consecutive points is higher in the training set than on the test set. The training set contains a few outliers, due to the previous observation having the value 0.

The tests will be made with the RMSE objective function because the goal is to achieve the lowest RMSE for all observations. However, for completeness sake, the performance metrics described in the Subsection 5.1.1 will also be measured in the tests, to provide more insights about the forecasts made. Because there are only 13 observations in the test set considered for the metric DRMSET with σ=0.5, only σ=0.2 will be considered for the DRMSET, to be statistically relevant. The test numbers and model descriptions are the same as in Subsection 4.4.9.

Tables 5.8, 5.9 and 5.10 show the performance metric results for the tests with no transformation applied, with one-lag differentiation and with one-lag multiplicative transformation, respectively. The tests with one-lag differentiation have better performance than the other tests. The test number 5 has better performance, measured with the RMSE performance metric, than the other tests, for all three differentiations. For most of the other performance metrics, the same can be confirmed.

The RMSE along the number of epochs trained in the test number 2 with one-lag differentiation can be seen in Figure 5.11. As the number of training epochs increases, the test error does not reduce.

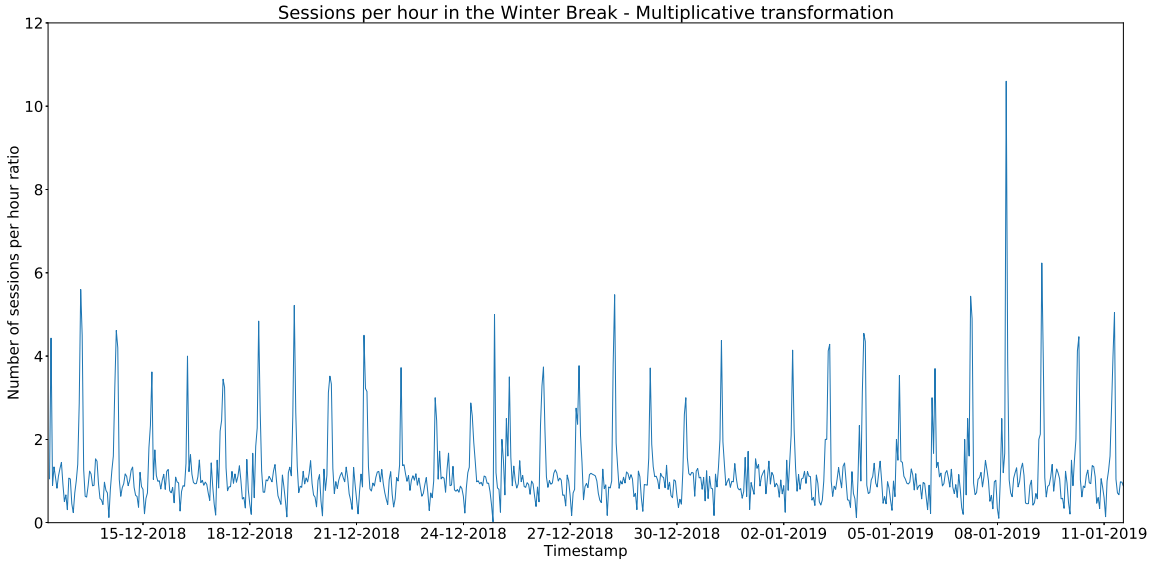

Figure 5.10: Test set with one-lag multiplicative transformation applied.

Test Number	RMSE	MAPE	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2
1	65.94	68.34	79.08	92.25	101.82	99.28
2	43.96	48.62	52.53	60.76	71.87	75.93
3	54.45	59.06	64.79	73.60	102.60	95.24
4	41.76	55.24	49.79	56.89	66.41	66.34
5	35.74	53.51	42.53	48.77	57.66	57.95

Table 5.8: Performance metric results with the new test set in the five tests, with no transformation applied to the input data.

Test Number	RMSE	MAPE	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2
1	37.47	52.02	43.66	46.69	44.52	64.81
2	40.52	41.30	47.01	53.22	51.32	71.94
3	39.94	29.17	47.48	51.62	59.32	69.12
4	31.37	33.84	36.82	39.76	42.93	54.52
5	29.47	32.63	34.42	37.65	37.95	51.32

Table 5.9: Performance metric results with the new test set in the five tests, with one-lag differentiation applied to the input data.

Test Number	RMSE	MAPE	RMSET σ=0.2	RMSET σ=0.5	RMSET σ=0.8	DRMSET σ=0.2
1	38.80	38.68	45.11	51.19	43.31	68.99
2	56.95	51.53	67.16	72.39	79.90	92.33
3	56.43	37.40	67.44	69.64	76.03	94.25
4	38.97	33.79	46.05	49.62	47.23	66.84
5	37.93	36.87	44.52	47.16	48.18	64.88

Table 5.10: Performance metric results with the new test set in the five tests, with one-lag multiplicative transformation applied to the input data.

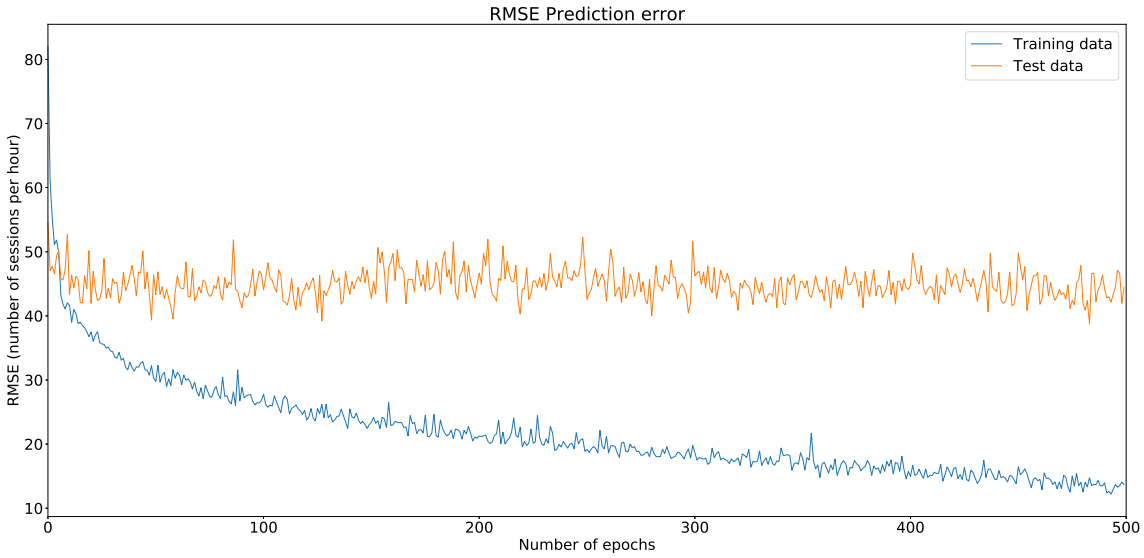

Figure 5.11: Value for the RMSE performance metric for the training set and test set along the number of trained epochs.

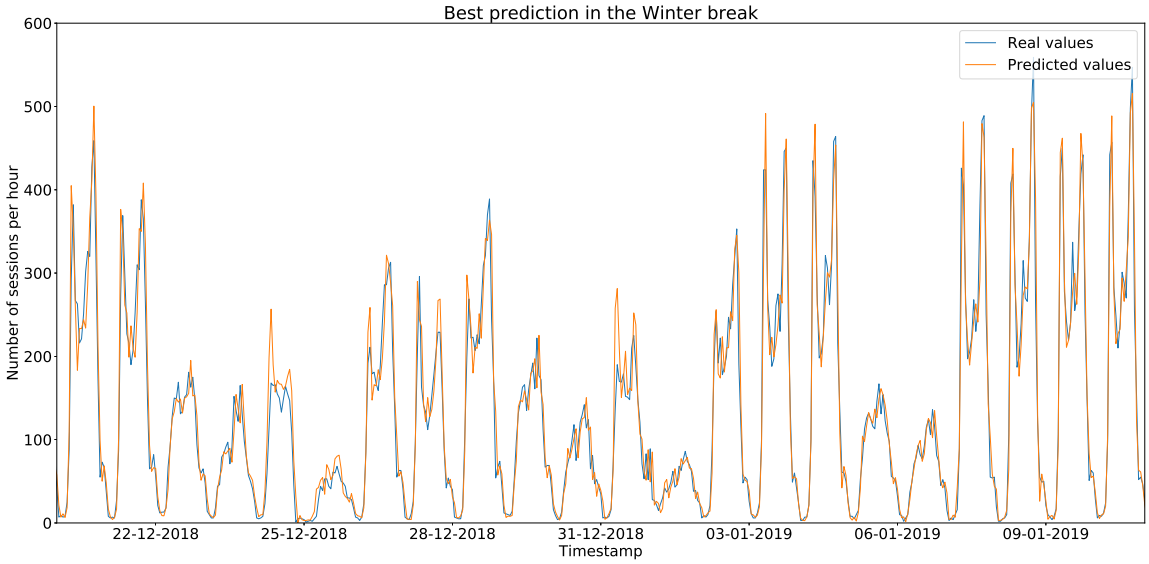

Figure 5.12: Forecasts made on the test set compared to the real observations. The model used was the test number five with one-lag differentiation.

While it seems that the model is overfitting, this proves the data mismatch between the train set and the test set, because after just a few epochs of training, the model starts to learn specific traits of the train set that do not generalize to the test set.

The predictions made by the test number five with one-lag differentiation in the test set can be seen in Figure 5.12. It shows that the predictions are surprisingly accurate, even for data that is different from the training set. Comparing with the persistence approach, described in Subsection 4.4.1, and using it as a naive baseline, the RMSE result is 71.46 sessions per hour, much worse than the best result obtained with an ensemble approach with the fine-tuned models.

5.3 Conclusion

In this chapter, two different scenarios were explored. In the first section, it were designed more accurate models for higher value observations. By creating new performance metrics, it was possible to measure and apply optimization of those metrics in the models, to achieve lower error. It was also tested to modify the loss function to improve the predictions for higher observation values. It was concluded that the new loss function does not improve significantly the prediction in those values, but it speeds up the training phase.

In the second section, it was shown that it is possible to build robust accurate prediction models that perform well for forecasting time-series data for anomalous time periods when the time-series data has the same patterns as the training data, but the absolute value of the data has a trend that it was not seen before in the training data. That can be done by pre-processing the data, performing normalization for the train and test datasets to become as similar as possible.

The models developed in this chapter are alternatives for the forecasting of the number of sessions, but with different goals. They can be implemented along with the models developed in the previous chapter (Chapter 4) in the same management system. If a network manager already knows that a certain time period will be anomalous, he can switch the predictor model and start using the model robust to anomalous time periods. The automatic switching of predictor models is essential for large networks with many network metric predictors. However, that approach is not addressed in this work.

In the Chapter 4, it was implemented and tested an accurate forecasting model for the number of sessions in a vehicular network, using different approaches. To demonstrate that the approaches taken are flexible for different types of metrics and networks, in the next chapter it will be used the same techniques to forecast the number of connected users in a 4G network. It will also be tested if the assumptions made in the vehicular network about the best hyper-parameters and the best models are maintained with a different dataset.

6

Forecasting the number of connected users in a 4G network

Mobile communication traffic has been increasing in the last few years, mainly due to video consumption and the real-time requirements by the users[1]. At the end of 2017, South Korea and Japan already had a 4G availability of 97.49% and 94.70% respectively, while Portugal had a 4G availability of 75.57%[2]. Because 5G networks are still far from wide availability, the process of migrating the current 4G network infrastructure to a 5G network infrastructure will take few years. Meanwhile, it is needed to monitor not only the 5G slices, but also 4G traffic, to be able to effectively provide quality of service to the users of both networks.

In this chapter, it will be described a forecasting model for the maximum number of connected users per hour in a 4G network, which is the metric most correlated with the network used bandwidth (and provides a similar and fair comparison to the work in the vehicular network). The first three sections of this chapter are dedicated to data exploration and data analysis. The last section is dedicated to the implementation, testing and tuning of some forecasting techniques (described in Section 2.4). In the Chapter 4, other forecasting techniques were used, such as 1-D convolutional networks, feature-based forecasts instead of time-lagged based forecasts and custom neural network architectures. Because those tests were time-consuming and the results were not satisfactory, those approaches will not be included in the analysis of this chapter.

Besides forecasting a 4G KPI, the goal of this section is to demonstrate that the forecasting approaches are flexible for different types of metrics and networks. It is also an objective to test if the assumptions made in the vehicular network about the best hyper-parameters and the best models are maintained with a different dataset.

6.1 Dataset exploration

The dataset used for the 4G forecasting has a set of KPIs (Appendix A - 4G Network KPIs) that describe the operation of 60 cells of an ISP in Portugal during March of 2019. The interval period between each measurement of a KPI is 1 hour, which means that there are 24 values for each KPI for each cell every day.

Because there are 60 KPIs in the dataset, it is needed to decide which KPI will be goal for forecasting. Just like in the vehicular network (Chapter 4), the goal is to forecast the number of

[1] https://www.stateofdigitalpublishing.com/insights/mobile-video-consumption/
[2] https://www.opensignal.com/reports/2018/02/state-of-lte

sessions per hour. Because in the 4G network each user is a session, there are four KPIs that qualify as candidates to forecast the number of sessions per hour in the network:

- Average number of connected users per hour - average number of User Equipments (UEs) connected to the E-UTRAN Node Bs (eNBs) per hour;
- Maximum number of connected users per hour - peak number of UEs connected to the eNBs per hour;
- Average number of active users per hour - average number of UEs considered active per hour (an active UE performs download or upload data transfers besides the control packets);
- Maximum number of active users per hour - peak number of UEs considered active per hour (an active UE performs download or upload data transfers besides the control packets).

The KPI values for each cell will be aggregated for the entire network. To choose one of the four KPIs to be the forecasting goal, it will be calculated the Pearson correlation coefficient (Equation 4.2) of each KPI with the data volume of upload and download in the network. The KPI with a higher Pearson correlation coefficient is the KPI that better represents the network traffic, and it will be chosen as the KPI to forecast.

The KPIs of Upload Packet Data Convergence Protocol (PDCP) Data Volume and Download PDCP Data Volume will be added to calculate the overall network traffic in megabytes (MB). The Pearson correlation coefficient of the four KPIs with this aggregated metric is the following:

- Average number of connected users per hour - 0.87;
- Maximum number of connected users per hour - 0.88;
- Average number of active users per hour - 0.78;
- Maximum number of active users per hour - 0.68.

The connected users have higher Pearson correlation coefficient with the overall network traffic than the active users. This indicates that, while one active user produces much more traffic than one connected user not active, most network traffic comes from the connected users because they are many more than the active users, they have background traffic, and the overhead of the control packets is high. Figures 6.1 and 6.2 show the maximum connected users and the maximum active users, respectively: the maximum number of connected users per hour is higher than the maximum number of active users per hour.

The chosen KPI to forecast for the 4G traffic is then the maximum number of connected users per hour (Figure 6.1). There are several patterns in the data that are important to mention. There is a daily pattern: the number of maximum connected users increases every day until around 12 a.m., and then decreases in the afternoon. There is also a weekly pattern, where the weekends typically have a lower maximum number of connected users than the same hour in the workweek (in Figure 6.1, the circles mark the peak values on the weekend days). Finally, there is a pattern in the data that can be explained by the monitoring of the KPI of the cells. Everyday, the maximum number of connected users at 00 a.m. is zero. This indicates that, at that time of the day, no monitoring measurement is being done of this KPI on the cells.

After an individual analysis of the maximum number of connected users per cell, it can be noticed that every cell has its specific patterns. Some cells have higher maximum number of connected users, while other cells have lower maximum number of connected users. Specifically, the high maximum number of connected users at 12 p.m. on the 28^{th} of March (Figure 6.1 is not the result of a singular event in a single cell, but rather the result of the high number of connected users in multiple cells. The maximum number of connected users in March in two of those cells can be seen in Figures 6.3 and 6.4. The cell number and location are anonymized due to privacy issues.

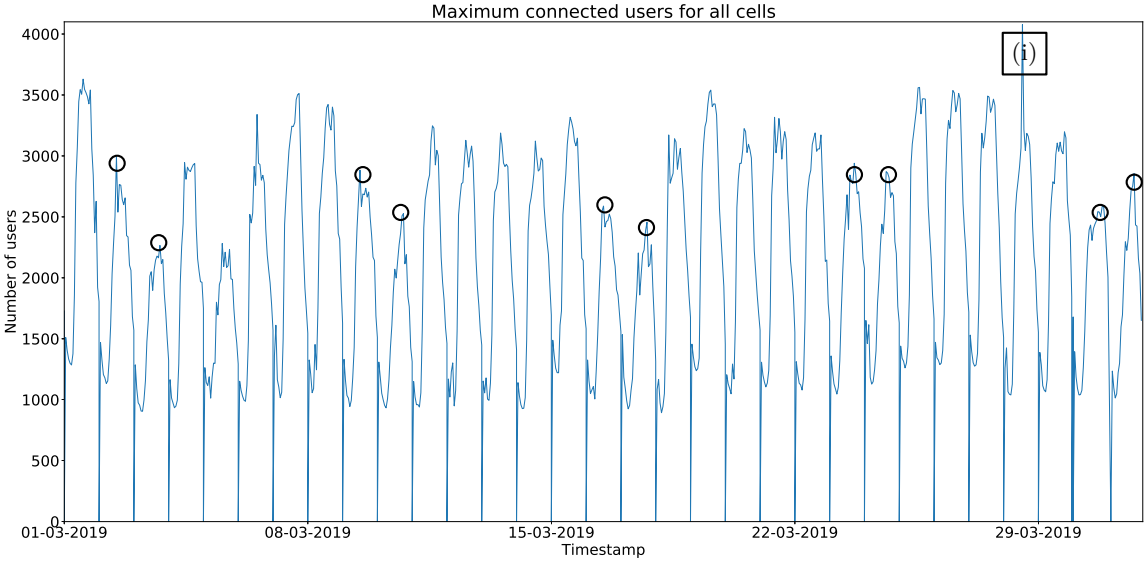

Figure 6.1: Maximum number of connected users per hour. The maximum values on the weekend days are marked with a circle.

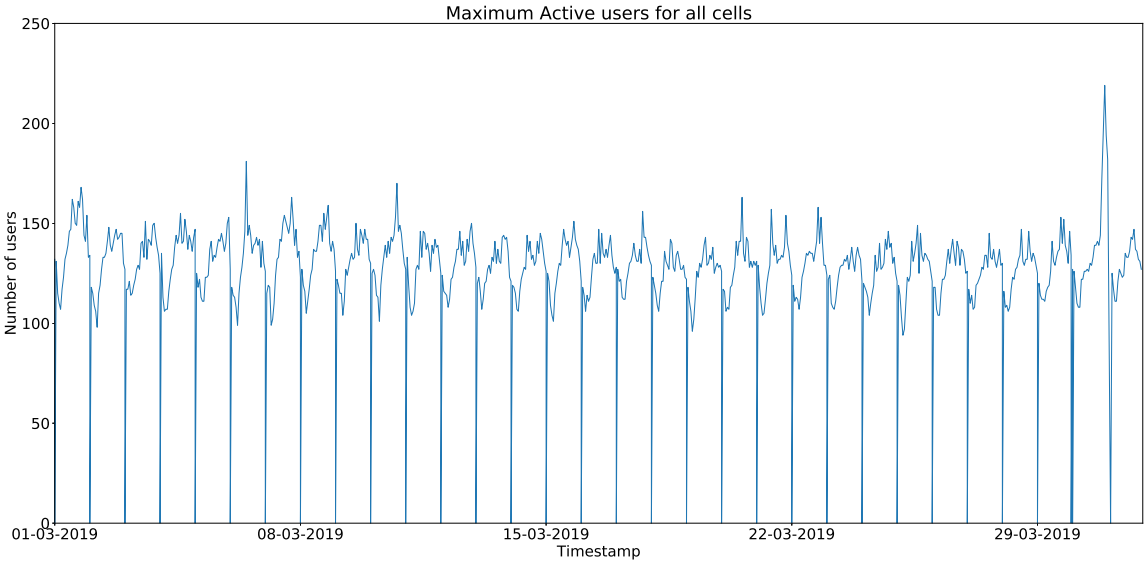

Figure 6.2: Maximum number of active users per hour.

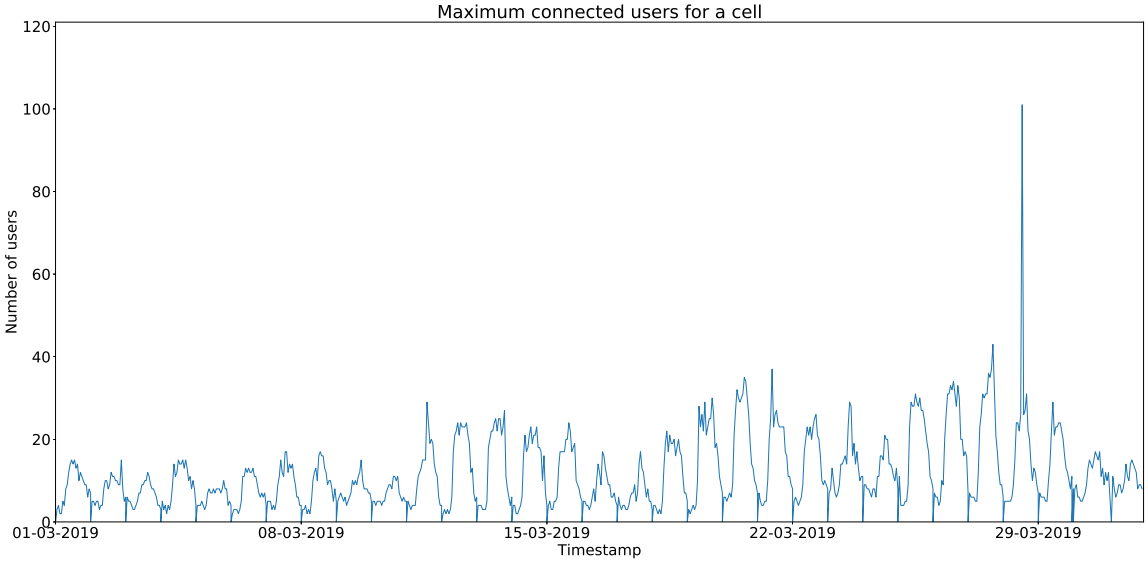

Figure 6.3: Maximum number of connected users per hour in a cell.

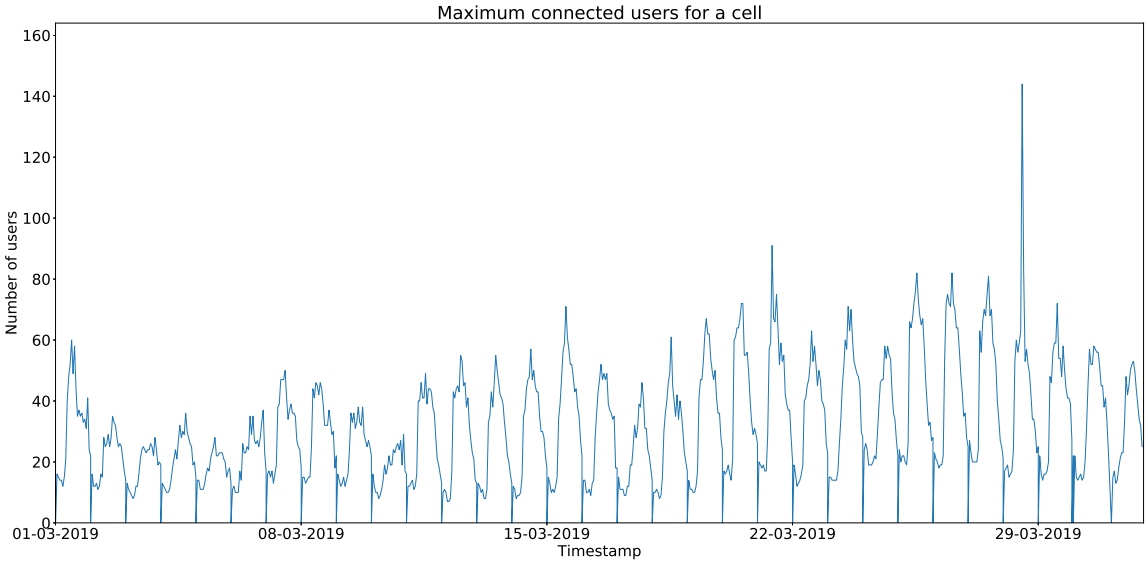

Figure 6.4: Maximum number of connected users per hour in a cell.

6.2 Time-series analysis

Just like in Section 4.2, an analysis of the time-series attributes of the dataset is needed to understand its stationarity. Figure 6.5 shows the rolling mean and standard deviation of the maximum number of connected users throughout the month, with a window of 24 hours. Both have variations depending on the day of the week, mainly on the weekends, where the mean is lower than on weekdays. However, in the long-term, no trend is detected. The overall mean of the maximum number of connected users is 2060.22 and the standard deviation is 876.91.

Figure 6.6 represents the autocorrelation function. It is possible to see the peak values around the lags multiple of 24, which indicates that there is a high correlation between days.

The histogram of the number of maximum connected users is in Figure 6.7. Most values are between the interval of 900 and 3600. There are 31 values of maximum connected users of 0, due to the maximum number of connected users everyday being 0 at 00 a.m. because of the monitoring done in the cells. It can also be seen that the most common maximum number of connected users per hour is around 1100 and around 3000.

A seasonal decomposition using moving averages was done using an additive model with a frequency of one week. Figure 6.8 shows the results, where the seasonality shows a trend of a lower maximum number of users on the weekends. It was also identified a smooth rising trend on the dataset, almost non-noticeable when looking at the previous data plots.

The Augmented Dickey-Fuller test, introduced and explained in Section 4.2, was also used to test if the time-series values had a unit root. The obtained p-value was 0.17%, with a significance level of less than 1%, which confirms that there is no major trend on the time-series data.

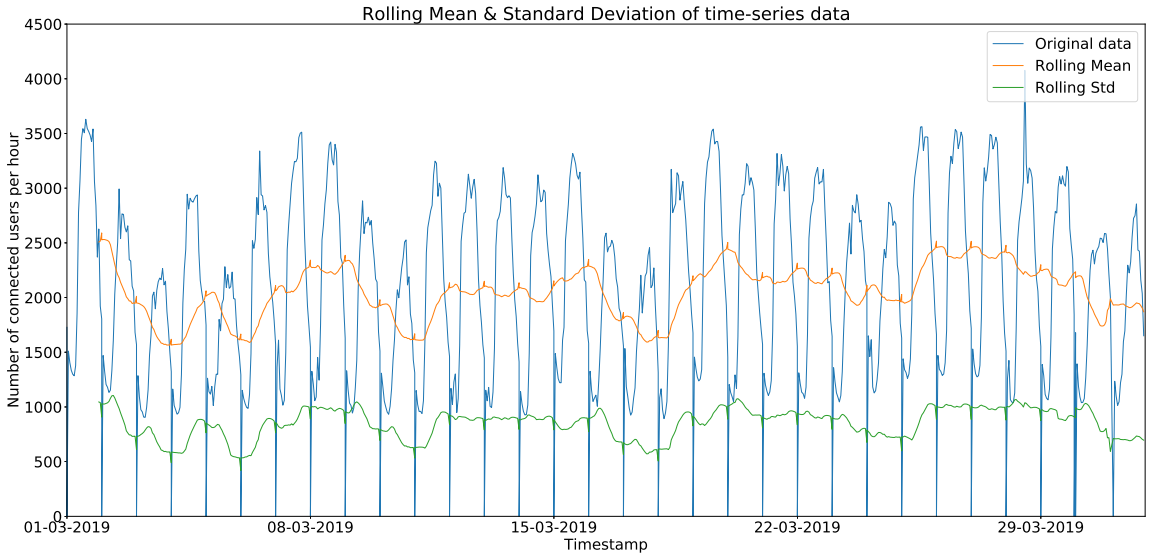

Figure 6.5: Rolling mean and standard deviation of the number of connected users along the month.

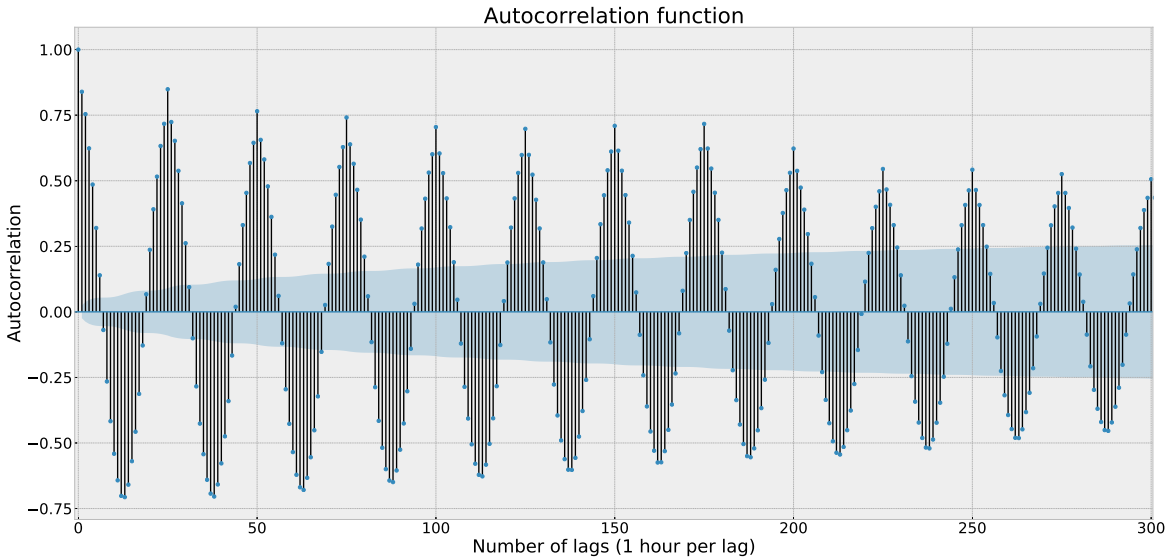

Figure 6.6: Autocorrelation plot for the maximum number of connected users.

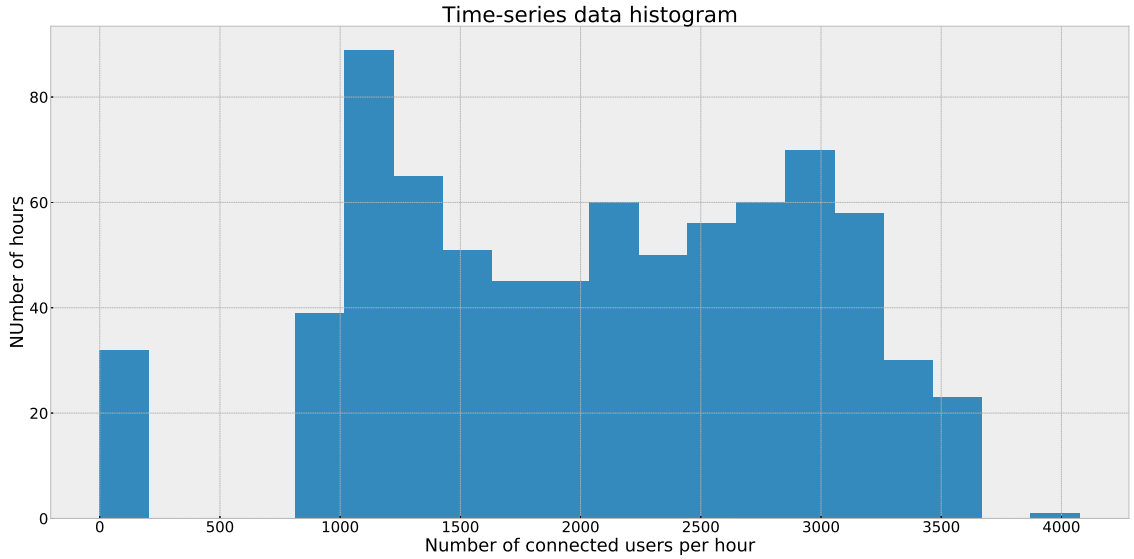

Figure 6.7: Histogram of the connected users in the network.

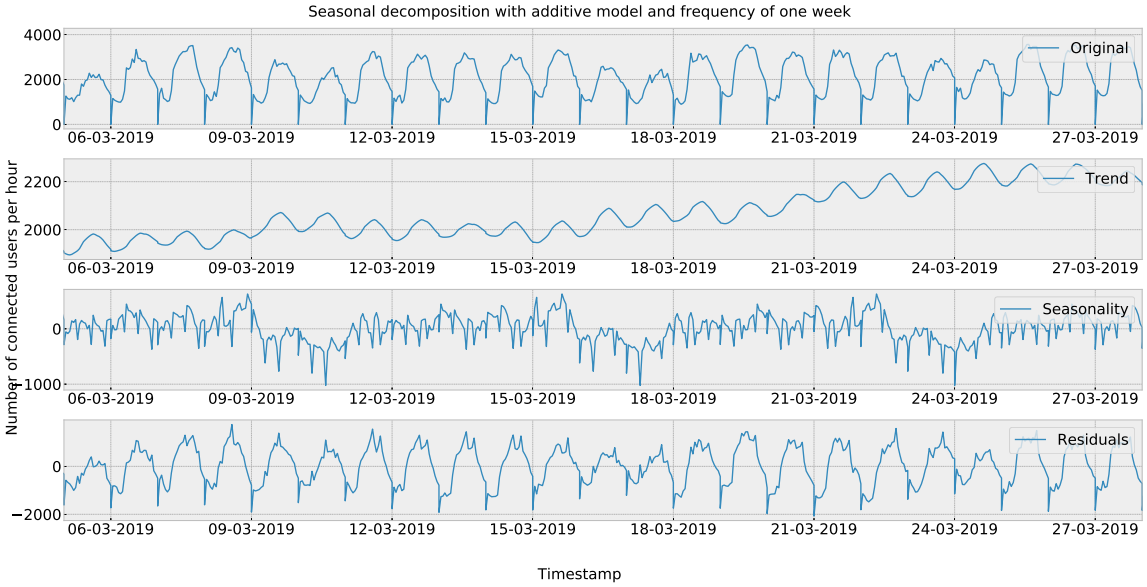

Figure 6.8: Seasonal decomposition using moving averages with an additive model and frequency of one week. The first plot shows the original data; the second plot shows the captured trend; the third plot shows the seasonality; the last plot shows the residuals.

6.3 Differentiation tests

The same three types of differentiation used in Section 4.3 will be used in this section: 1-lag differentiation, 24-lag differentiation and 168-lag differentiation. According to the dataset, the same differentiation seems appropriate due to the higher lags in the autocorrelation function (Figure 6.6) and due to the observed seasonal patterns. The differentiation step tries to make the data distribution closer to a stationary distribution by eliminating trend and seasonality, for the forecasting algorithms to make more accurate forecasts.

Figure 6.9 shows the values, the rolling mean and the standard deviation of the data when differenced by one lag. The distribution has periodic peaks due to the differentiation of the zero-values in the dataset. The rolling mean was reduced to near-zero values, while the rolling standard deviation remains at around 500. The autocorrelation function (Figure 6.10) shows that most of the autocorrelation was removed from the time-series. There is still autocorrelation in the lags multiple of 24, indicating a daily seasonality pattern. Figure 6.11 shows the seasonal decomposition. It is possible to see that the trend was removed from the time-series, now having a seasonal pattern with small influence in the final values (between -10 and 10 connected users per hour).

With the 24-lag differentiation applied to the dataset, the rolling mean varies much more than with the 1-lag differentiation (Figure 6.12). The rolling standard deviation also varies more, but the overall mean is similar. The autocorrelation function (Figure 6.13) has smaller peaks at lags multiple of 24 than the one-lag differentiation, and the highest peak is at 168 lags, indicating that a one-week lag may be a good differentiation. Just like in the 1-lag differentiation, the seasonal decomposition (Figure 6.14) shows that there is no clear trend in the data.

The dataset with 168-lag differentiation rolling mean and standard deviation can be seen in Figure 6.15, and the autocorrelation plot in Figure 6.16. The overall mean is 60.83 and the standard deviation mean is 297.09. The autocorrelation function is similar to the one with no differentiation, which can indicate that this differentiation may not be the most adequate. In the seasonal decomposition, a smooth rising trend was identified (Figure 6.17).

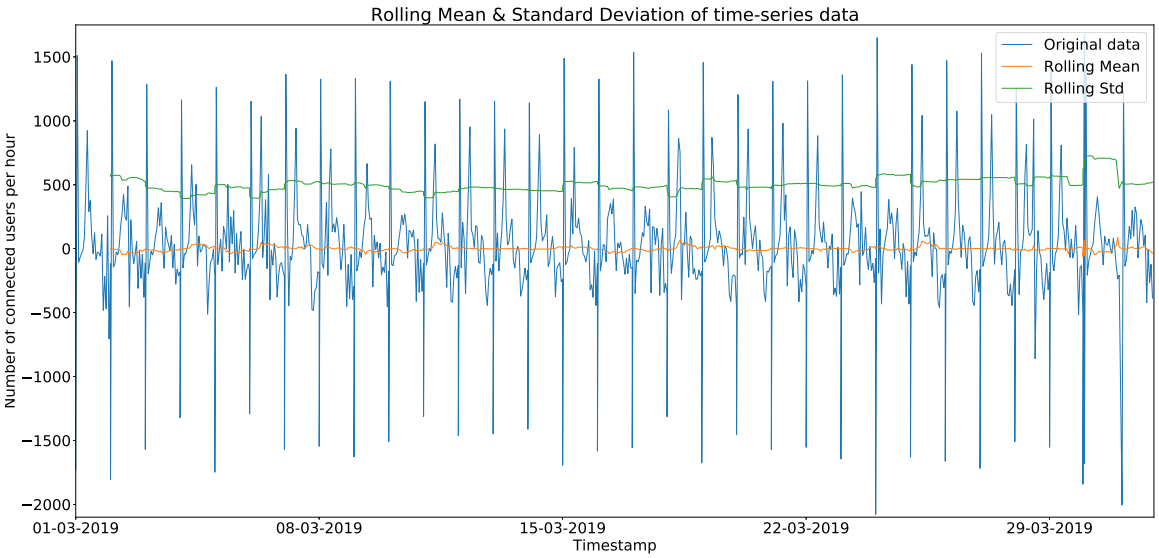

Figure 6.9: Rolling mean and standard deviation of the number of connected users along the month with one-lag differentiation.

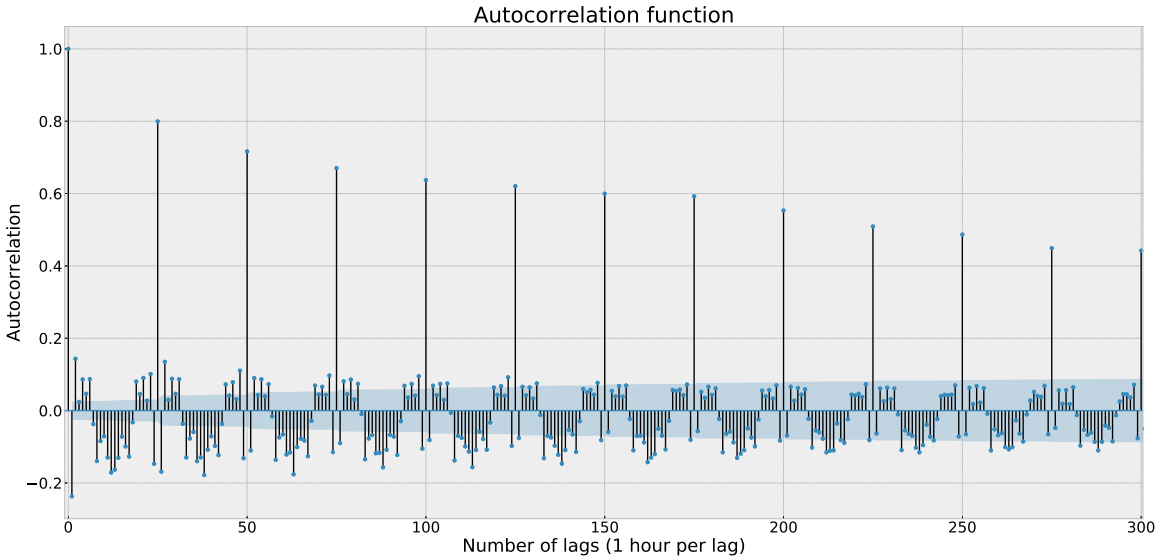

Figure 6.10: Autocorrelation plot for the maximum number of connected users differentiated by one lag.

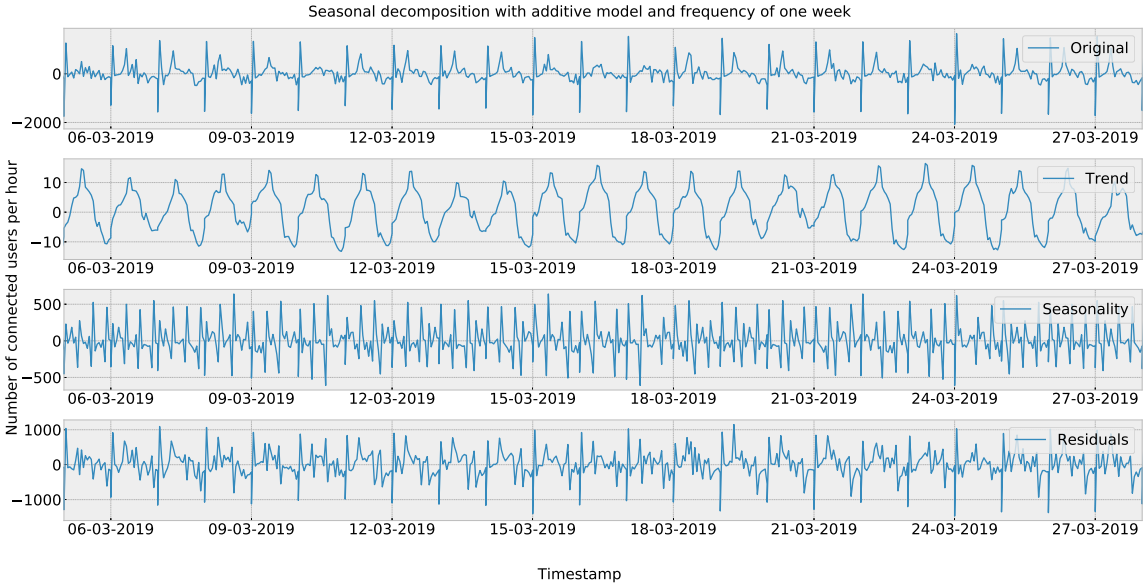

Figure 6.11: Seasonal decomposition using moving averages with an additive model and frequency of one week. The first plot shows the original data; the second plot shows the captured trend; the third plot shows the seasonality; the last plot shows the residuals.

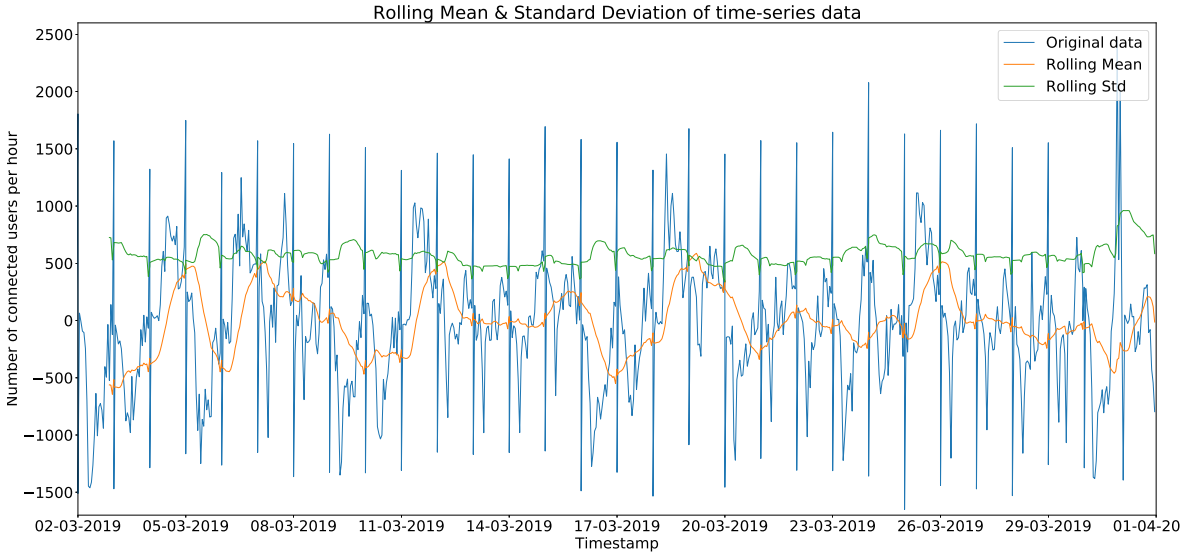

Figure 6.12: Rolling mean and standard deviation of the number of connected users along the month with 24-lag differentiation.

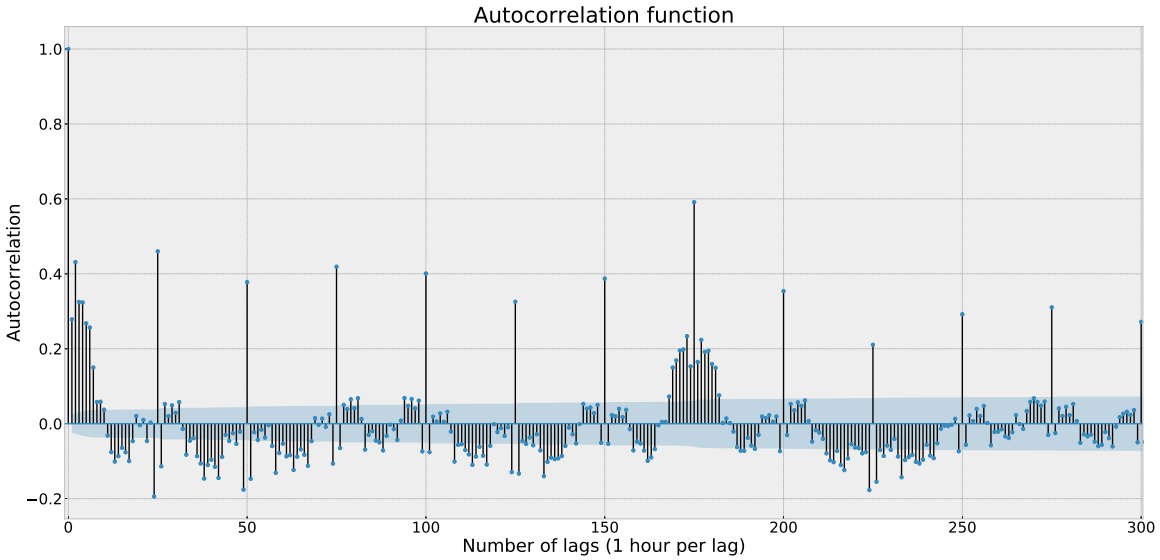

Figure 6.13: Autocorrelation plot for the maximum number of connected users differentiated by 24 lags.

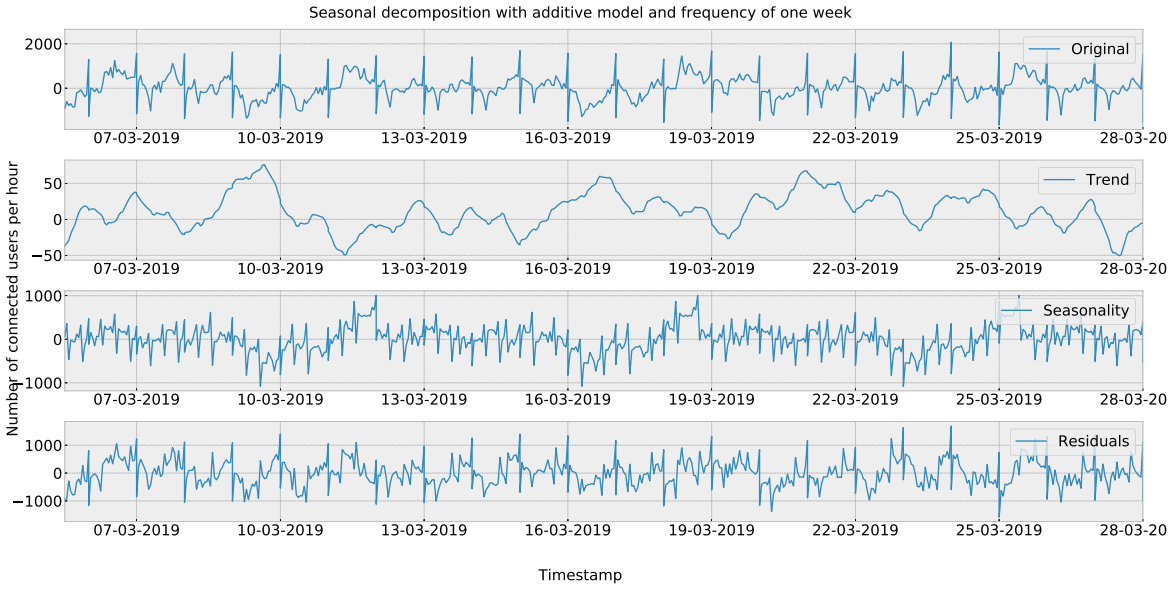

Figure 6.14: Seasonal decomposition using moving averages with an additive model and frequency of one week. The first plot shows the original data; the second plot shows the captured trend; the third plot shows the seasonality; the last plot shows the residuals.

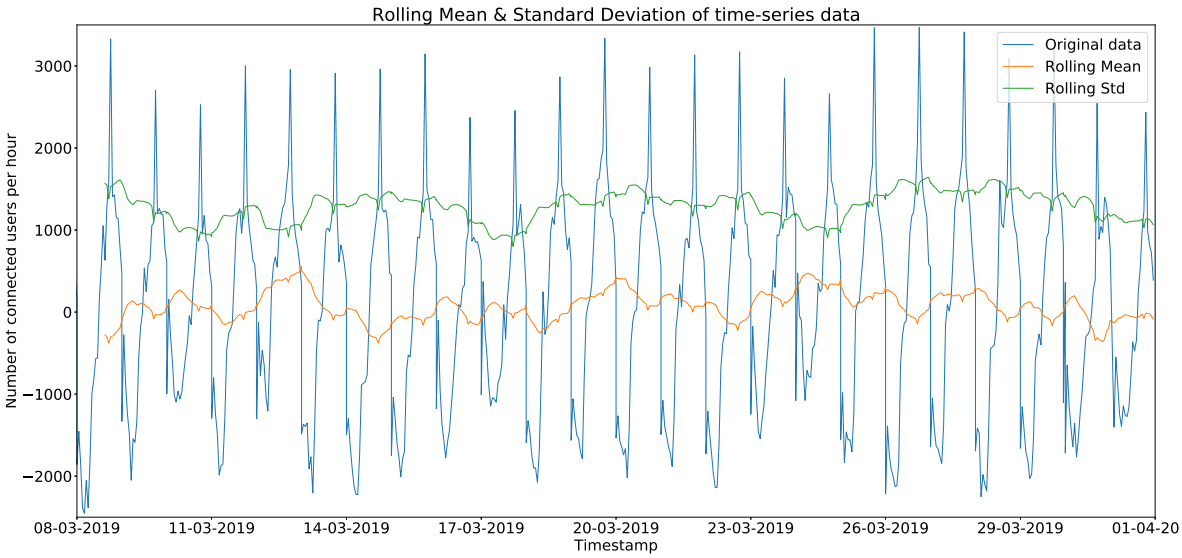

Figure 6.15: Rolling mean and standard deviation of the number of connected users along the month with 168-lag differentiation.

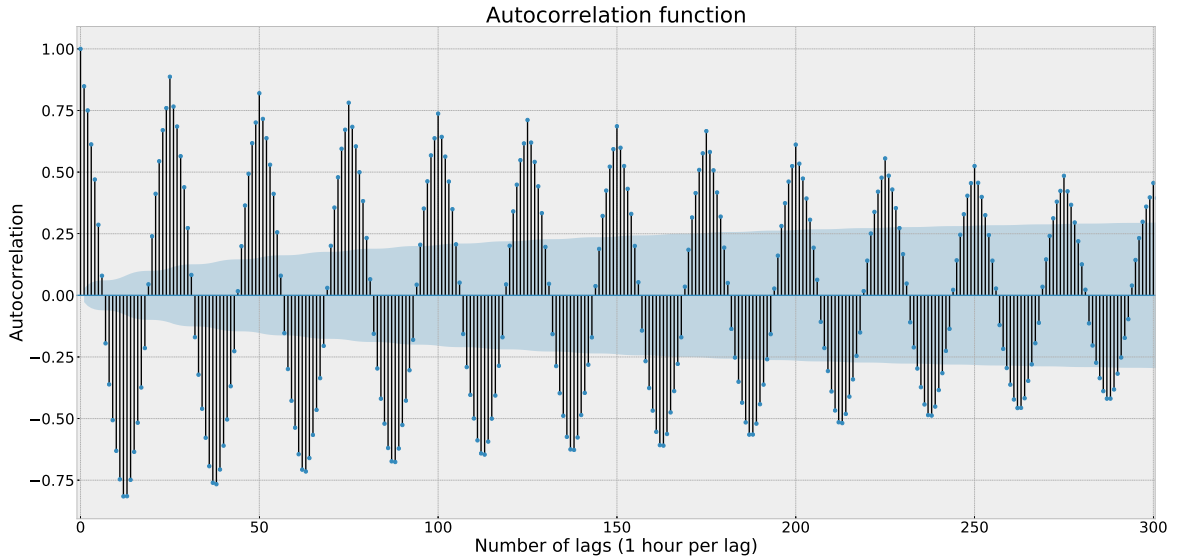

Figure 6.16: Autocorrelation plot for the maximum number of connected users differenced by 168 lags.

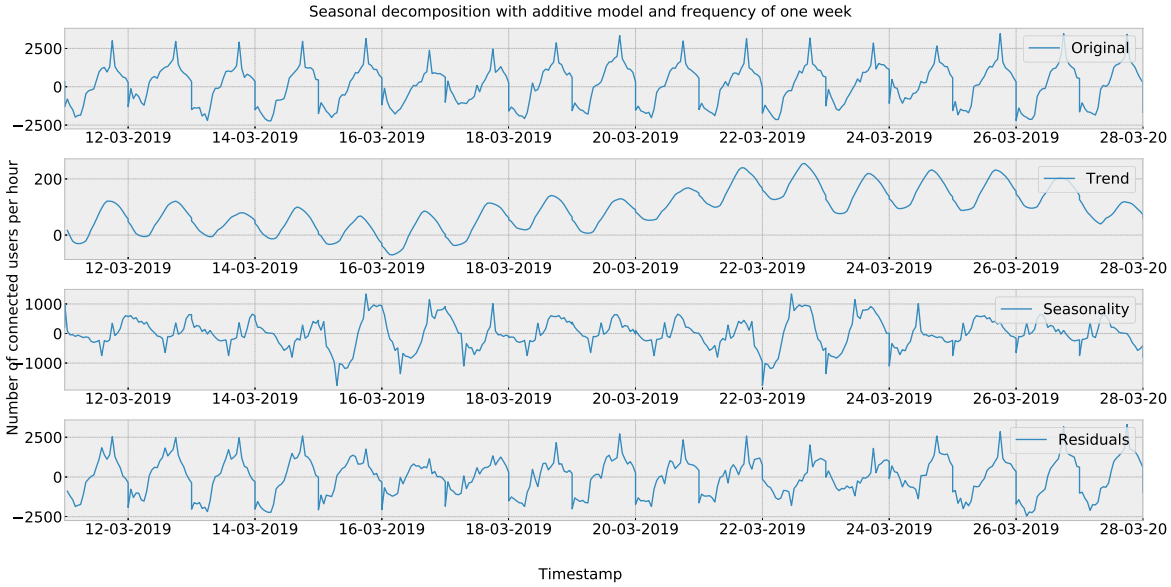

Figure 6.17: Seasonal decomposition using moving averages with an additive model and frequency of one week. The first plot shows the original data; the second plot shows the captured trend; the third plot shows the seasonality; the last plot shows the residuals.

The 1-lag and 24-lag differentiations seem to approximate the data to a stationary distribution better than the 168-lag differentiation, because its autocorrelation function (Figure 6.16) still shows high autocorrelation for many lags. The three differentiations will be applied to the data in the forecasts made in the next section, to obtain the most accurate possible forecasting of the maximum number of connected users.

6.4 Forecasting approaches

In this section the persistence, ARIMA, classical machine learning, feed-forward neural, and recurrent neural networks will be used to forecast the maximum number of connected users in the 4G network.

The dataset will be split in train, cross-validation and test sets. The division will be made according to Figure 6.18: the last week of March will be used as test set (from the 25^{th} of March to the 31^{st} of Match), the previous week as cross-validation set (from the 18^{th} of March to the 24^{st} of Match), and the train set will be from the beginning of the month to the 17^{th} of March. The training set will contain 17 days of data (54.84%), while the cross-validation and test set will contain other 7 days of data each (22.58%). The reason for that dataset division is that it was important to have at least one full week in the cross-validation and test sets.

The main performance metric used for this task will be the RMSE, as described in Section 4.4, with the Equation 4.4. However, other metrics will also be monitored, such as MAPE (Equation 4.5), RMSET (described in Subsection 5.1.1, with the Equation 5.1) with the threshold values of 0.2, 0.5 and 0.8, and DRMSET (also described in Subsection 5.1.1, with the Equation 5.2) and the threshold values of 0.2, 0.5 and 0.8.

Since in the data there is an hour every day where the maximum number of connected users is zero, that value will be statically defined, overriding the forecasts of the models, to better simulate a real utilization of a predictor.

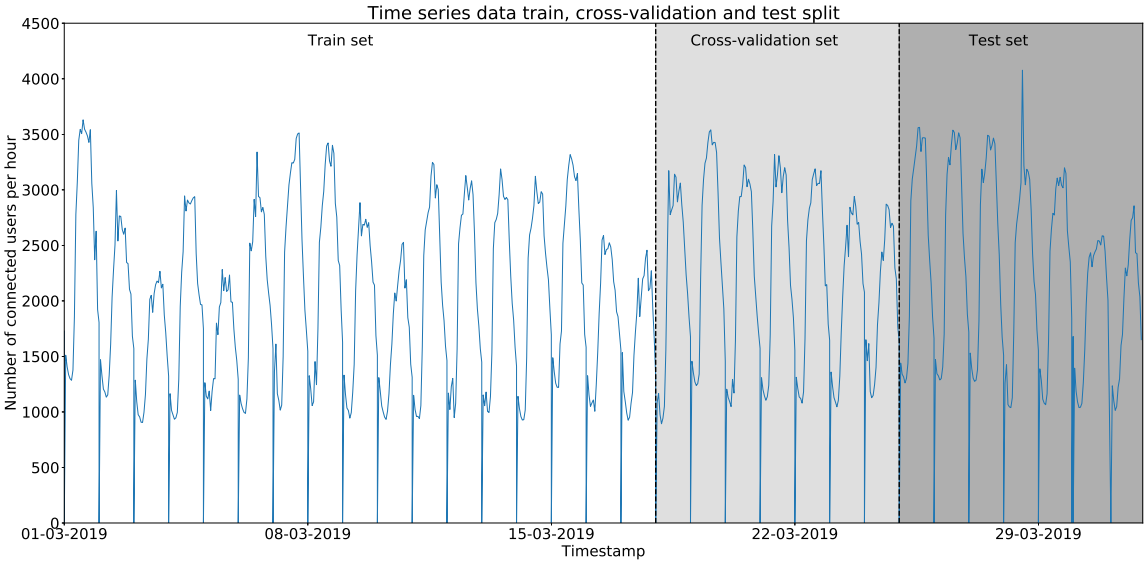

Figure 6.18: Dataset division in train, cross-validation and test set.

6.4.1 Persistence model approach

Just like in Subsection 4.4.1, the first approach used to establish a benchmark will be the persistence model, where the forecast will be given by predicting the same value K time periods before, as described in Equation 4.6.

The tests used a K value of 1, 24 and 168. The tests were done using the cross-validation set and the results can be seen in Table 6.1. As the K value increases, the RMSE also increases, suggesting that the maximum number of connected users is more similar in the previous hour than in the previous day, or in the previous week. The same can be seen in the MAPE and RMSET metrics. However, in the DRMSET, for the three threshold values, the 24-hour lag has better results than the 1-hour lag or the 168-hour lag. Figure 6.19 shows the forecasts made by the persistence model with 1-hour lag when compared with the real values.

The RMSE of 506.83 users per hour will be used as a baseline for testing the models in the next subsections.

K	**RMSE**	**MAPE**	**RMSET σ=0.2**	**RMSET σ=0.5**	**RMSET σ=0.8**	**DRMSET σ=0.2**	**DRMSET σ=0.5**	**DRMSET σ=0.8**
1	506.83	16.90	383.42	338.95	271.44	1424.18	1697.35	1440.00
24	595.49	22.57	539.29	535.67	409.77	867.7	1128.77	21.00
168	1299.12	62.45	1291.16	1251.63	1386.03	1129.65	1207.55	976.00

Table 6.1: Performance metric values of the persistence model for different value of K.

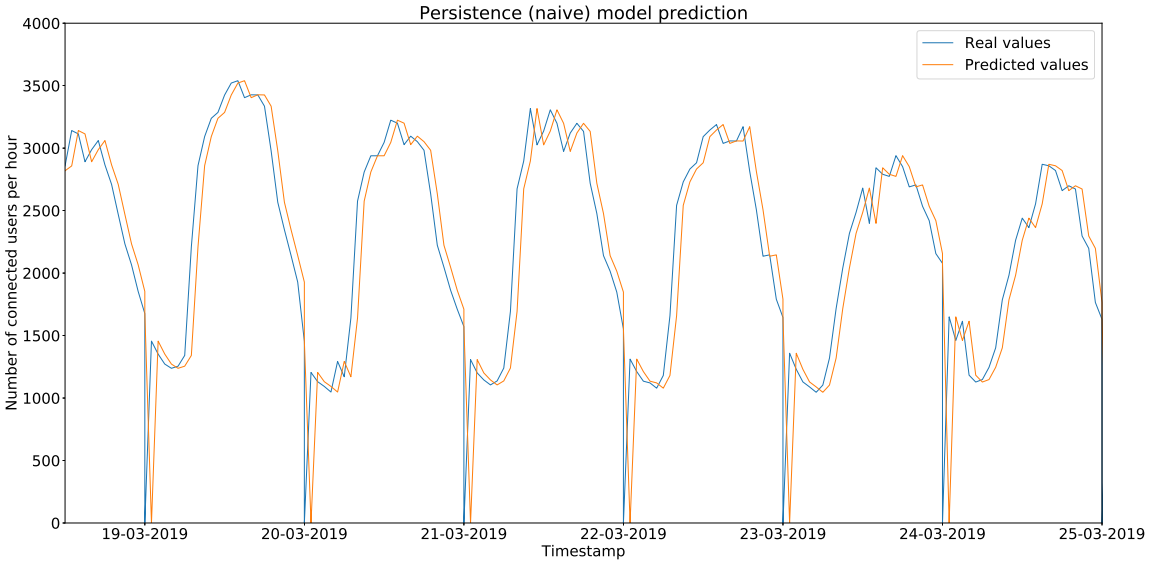

Figure 6.19: Forecasts made by the persistence model with K=1 in the cross-validation set, compared with the real values.

6.4.2 ARIMA approach

After the persistence model approach, it will be tested a statistical method for forecasting the number of users in the network. The ARIMA model was described in Subsection 2.4.4. In these tests, the same configurations and hyper-parameters than the ones presented in Subsection 4.4.2 will be used.

Tables 6.2 and 6.3 show the average RMSE for the p and q parameter degrees, respectively. The average RMSE is lower when the p degree is higher. For the parameter q, there is no clear trend on the results. Table 6.4 shows the average RMSE for various differentiation values. As the differentiation parameter increases, the average RMSE also increases, indicating that the data is best modeled with a higher p value and a lower differentiation lag.

Table 6.5 shows the five best RMSE results for the cross-validation set, with the respective

p	**RMSE**
0	527.13
1	474.80
2	451.83
3	436.96
4	431.49
5	422.40

Table 6.2: RMSE results for the forecast with the ARIMA model for different p values for the 4G network.

q	**RMSE**
0	465.65
1	463.58
2	414.40
3	429.05
4	416.22
5	447.81

Table 6.3: RMSE results for the forecast with the ARIMA model for different q values for the 4G network.

d	**RMSE**
0	350.98
1	356.07
24	438.84
168	559.66

Table 6.4: RMSE results for the forecast with the ARIMA model for different d values for the 4G network.

d	p	q	**RMSE**	**MAPE**	**RMSET σ=0.2**	**RMSET σ=0.5**	**RMSET σ=0.8**	**DRMSET σ=0.2**	**DRMSET σ=0.5**	**DRMSET σ=0.8**
0	2	4	260.39	17.19	265.99	249.98	236.04	534.06	361.13	564.82
0	3	2	261.43	17.27	267.05	253.08	239.64	545.92	382.58	593.84
0	2	3	263.88	16.98	269.55	264.72	255.51	514.64	348.06	549.66
0	4	2	264.08	17.60	269.76	249.68	239.27	538.83	361.38	555.93
0	2	2	271.06	16.83	276.89	286.12	292.20	433.72	218.02	372.34

Table 6.5: Performance metric values of the five models with lower RMSE result and the respective hyper-parameters.

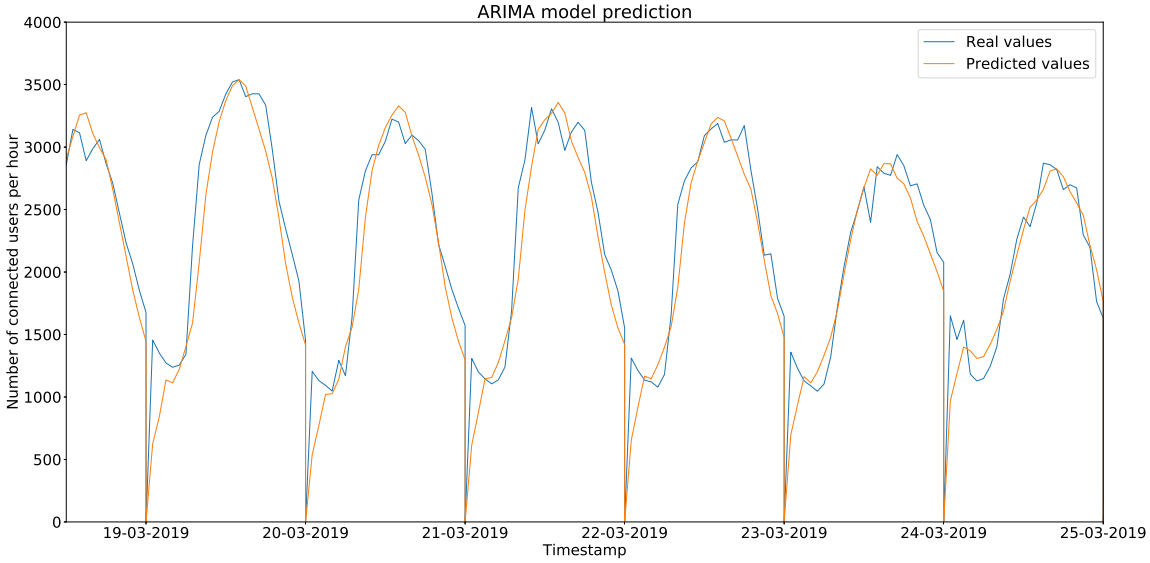

Figure 6.20: Prediction made by the ARIMA model in the cross-validation set with the order parameters (2,0,4) compared with the real values.

performance metrics. The best results have no differentiation, and the p and q values vary from 2 to 4.

The predictions of the best model can be seen in Figure 6.20, with a RMSE of 260.39 users per hour.

While in Subsection 4.4.2 the best results for the ARIMA method had 168-lag differentiation, for this dataset the differentiation transformation does not improve the results. It is shown that the best differentiation parameter for the forecasting task depends not only on the approach used, but also on the dataset.

6.4.3 Classical Machine Learning Models

The same algorithms used in Subsection 4.4.3 will be used in this subsection to forecast the maximum number of connected users in the 4G network. A more detailed description about the algorithms and the test configurations is in Subsection 4.4.3.

The hyper-parameters below will be tested for all the algorithms:

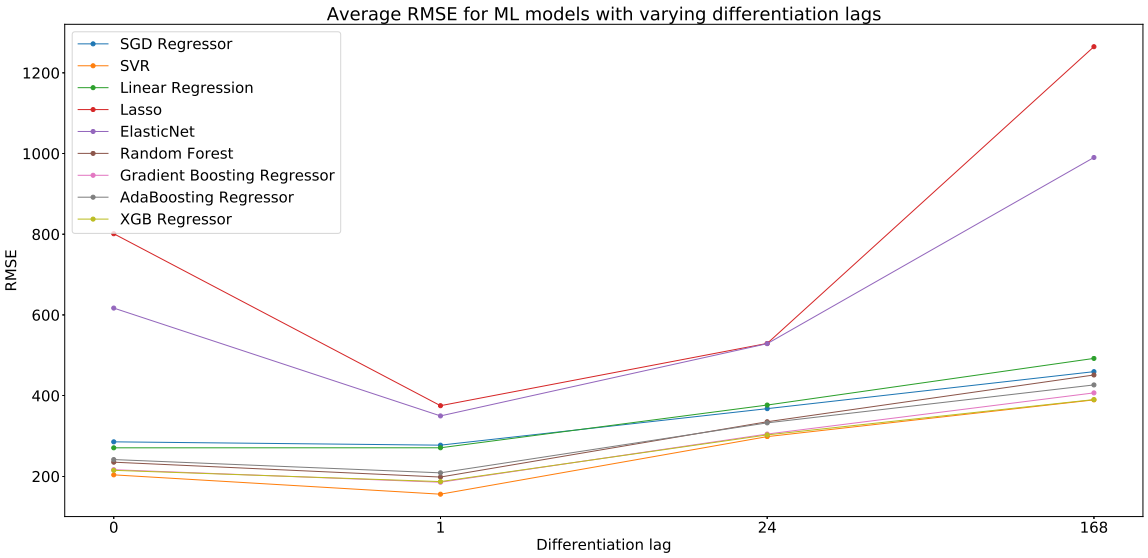

Figure 6.21: Average RMSE for classical machine learning algorithms varying the differentiation lag.

- differentiation lag (l): 0 (no differentiation), 1, 24, 168;
- number of previous values (l): 1, 12, 24, 168.

Besides the lagged values of the time-series data as input features, another way to improve the results is to add additional features related with the dataset. The same additional features presented in Subsection 4.4.3 (day of the week, workday and hour of the day) will be used to try to improve the results. In the tests made, five combinations of these features were tested:

1. no additional features added;
2. day of the week added as feature;
3. work day added as feature;
4. hour of the day added as feature;
5. all the previous added as features.

Figure 6.21 shows the average RMSE for different algorithms, varying the differentiation lag, when no additional features were added. The best results have one lag as differentiation. The same happens for other combinations of additional features. Figure 6.22 shows the average RMSE for the different algorithms tested, varying the number of lags, when no additional features were used. Another trend that can be seen across different combinations of additional features is that the average lower RMSE happens when the number of lags is 24 for most algorithms. The algorithms with the best average results are the SVR, the XGB Regressor and the Gradient Boosting Regressor.

Table 6.6 shows the best results for each set of additional features used, as well as the algorithms and its hyper-parameters. The best result is achieved with only the work day as feature, with the SVR algorithm. Furthermore, most of the best results are achieved with one-lag differentiation and 24 lags as input.

The best result (RMSE of 127.49 users per hour) was achieved using the SVR algorithm, with one-lag differentiation, an input lag of 24 and the work day as additional feature, clearly outperforming the ARIMA approach, that had a RMSE of 260.39 users per hour. The forecasts are in Figure 6.23 compared with the real values.

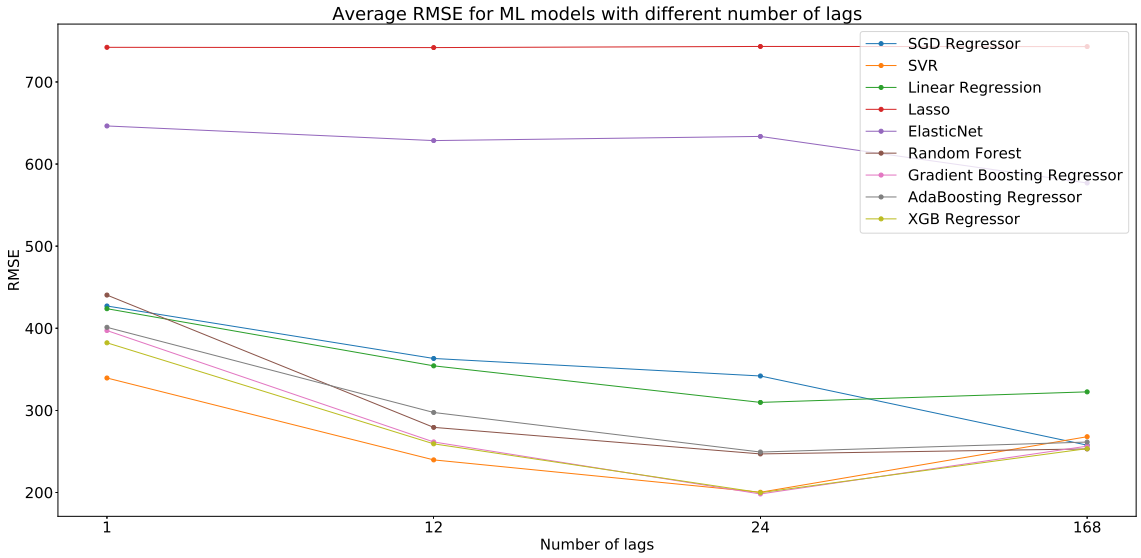

Figure 6.22: Average RMSE for classical machine learning algorithms varying the number of lags as input.

Set of tests	RMSE	Algorithm	Differentiation	Lag Number
1	132.85	Random Forest	0	12
2	132.56	Random Forest	1	24
3	127.49	SVR	1	12
4	135.44	Gradient Boosting Regressor	1	24
5	134.91	XGBRegressor	1	24

Table 6.6: Best result for each set of tests and its hyper-parameters, varying the additional parameters.

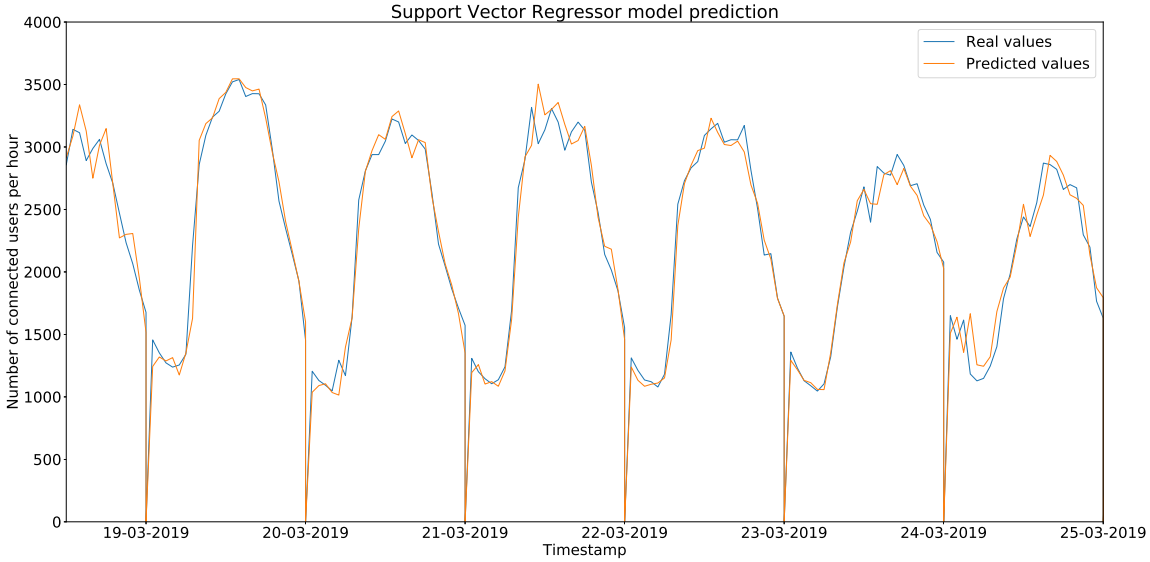

Figure 6.23: Forecasts made by the SVR model in the cross-validation set compared with the real values, with a RMSE of 127.49 sessions per hour.

6.4.4 Feed-Forward Neural Networks

In this subsection, it will be used *Time Lagged Feed-forward Networks* to try to improve the forecasts. The tests configurations will be similar to the tests done in Subsection 4.4.4, with the following hyperparameters:

- differentiation lag (d): 0, 1, 24, 168;
- number of previous values (l): 1, 12, 24, 168;
- model numbers: 0, 1, 2, 3, 4, 5, 6, 7.

Each one of the eight models has a different configuration, varying the number of hidden layers and the number of neurons per layer in the network. In Table 6.7 each model is described with its hyper-parameters.

The models have other configurations. After each hidden layer, there is a dropout layer with the value of 0.2. All hidden layers are Dense and have as activation function the ReLU function. The output layer does not have any activation function.

Table 6.8 shows the average RMSE for various differentiation values. Just like in the previous subsection, the best results are achieved when the differentiation lag is one. The best lag number (24) is also the same as in the previous subsection (Table 6.9). When comparing the various models used, Table 6.10 shows that the best average performing models are the models' number 2, 3, 6 and 7. These models have 2 or 4 hidden feed-forward layers.

The ten results with lower RMSE are in Table 6.11. All the ten best results have one-lag as differentiation, and the lag number varies between 12, 24 and 168.

The same combinations of additional features as in the previous subsection were tested in the ten best configurations, to try to improve the results. The combination are:

1. no additional features added;
2. day of the week added as feature;
3. workday added as feature;
4. hour of the day added as feature;
5. all the previous added as features.

Model Number	**Hidden feed-forward layers**	**Neurons per layer**
0	1	20
1	1	200
2	2	20
3	2	200
4	3	20
5	3	200
6	4	20
7	4	200

Table 6.7: Configurations tested for the feed-forward neural network for the prediction of the number of connected users in the 4G network.

d	**RMSE**
0	214.40
1	162.22
24	277.12
168	337.50

Table 6.8: RMSE results for the forecast with the Feed-forward Neural Networks for different d values for the 4G network.

l	**RMSE**
1	335.11
12	221.81
24	184.61
168	249.72

Table 6.9: RMSE results for the forecast with the Feed-forward Neural Networks for different number of input values for the 4G network.

Model number	**RMSE**
0	251.74
1	256.33
2	245.50
3	238.47
4	250.84
5	254.70
6	244.33
7	240.58

Table 6.10: RMSE results for the forecast with the Feed-forward Neural Networks for different models for the 4G network.

Best Results	**Model Number**	**Differentiation**	**Lag Number**	**RMSE**
1	6	1	12	126.92
2	7	1	24	127.29
3	7	1	168	129.57
4	3	1	168	129.89
5	6	1	168	130.22
6	3	1	24	131.57
7	2	1	24	132.28
8	5	1	168	133.19
9	1	1	168	133.48
10	7	1	12	133.75

Table 6.11: Ten best results on the tests of forecasting the maximum number of connected users in the 4G network, using a feed-forward neural network.

Table 6.12 shows the best results achieved with the RMSE values for the different sets of combinations for the ten best configurations. Just like in the previous subsection, the workday feature has the lowest average RMSE and also the best result overall, with an RMSE of 115.54 connected users per hour, an improvement from the result of RMSE of 127.49 users per hour in the previous subsection. The forecasts for this configuration are in Figure 6.24, compared with the real values.

The best hyper-parameters from the previous subsection were still the best for the Feed-forward Neural Network models. The best additional feature, differentiation value and number of lags as input, were the same for the machine learning approaches.

Best Results / Set of tests	1	2	3	4	5
1	126.92	142.70	125.73	150.51	124.81
2	127.29	119.47	119.44	121.22	123.29
3	129.57	121.80	118.66	122.69	121.03
4	129.89	127.15	121.62	123.99	124.33
5	130.22	140.21	137.70	136.08	138.84
6	131.57	126.03	128.34	127.04	126.36
7	132.28	132.06	124.97	134.63	140.26
8	133.19	122.67	123.51	125.12	121.26
9	133.48	130.51	134.19	128.52	127.94
10	133.75	123.63	115.54	116.50	125.77
Average	**130.81**	**128.62**	**124.97**	**128.63**	**127.39**
Best Result	**126.92**	**119.47**	**115.54**	**116.50**	**121.03**

Table 6.12: RMSE result for the different set of tests with additional features made only to the ten best configurations

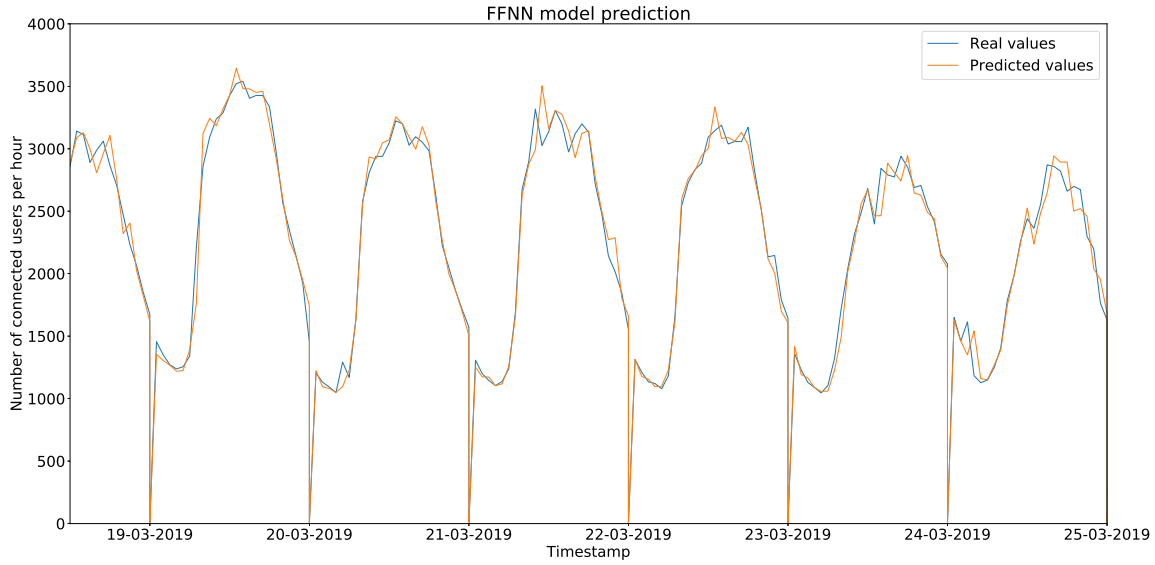

Figure 6.24: Forecasts made by the best feed-forward neural network model in the cross-validation set compared with the real values, with a RMSE of 115.54 sessions per hour.

6.4.5 Recurrent Neural Networks

Another type of neural network used to improve the forecasts will be recurrent neural networks (described in Subsection 2.4.2). It will be used an LSTM network. The tests configurations are similar to the tests configurations in Subsection 4.4.5. Due to the time restrictions when using LSTMs, which is very time-expensive to train, the hyperparameter space must be reduced in comparison with the previous subsection. Because in the previous subsection the best results were achieved using a differentiation lag of 1, in these tests the differentiation lag will have the value of 1. The following hyper-parameters will be used:

- lag number (l): 12, 24, 168;
- model numbers: 0, 1, 2, 3, 4, 5, 6, 7.

The models are similar to what was used in the previous subsection (Table 6.7), with the difference that the hidden layers will be LSTM layers and its activation function will be the *tanh* function, instead of ReLU.

l	**RMSE**
12	139.17
24	123.79
168	157.33

Table 6.13: Average RMSE for LSTM networks tests with different number of input lags.

Model number	**RMSE**
0	134.84
1	128.54
2	126.85
3	134.12
4	130.22
5	132.52
6	129.14
7	204.54

Table 6.14: Average RMSE for LSTM networks tests with the different models.

Best results	**Model Number**	**Lag Number**	**RMSE**
1	2	168	113.7012
2	5	24	116.1601
3	4	168	116.4571
4	3	24	117.3135
5	6	24	119.2779

Table 6.15: RMSE for the best five results of the test with LSTM networks, with the model number and the lag number.

Table 6.13 shows the average RMSE for different values of lag numbers as input. The best number of lags as input, on average, is 24, just like in the previous tests. Table 6.14 shows the average RMSE for the different models. It is possible to see that the average RMSE difference among the models is very small (with exception to the model number 7).

Table 6.15 shows the best five results, where the best result (RMSE of 113.70, with a lag number of 168 and the model 2) outperforms the previously lower RMSE result (RMSE of 115.54 maximum connected users per hour).

The same sets of additional features used in the previous subsection (Subsection 6.4.4) will be tested on the configuration that provided the best result. Table 6.16 has the RMSE results with the additional sets of features. The additional features only increase the RMSE, showing that for this network configuration, the best result is achieved when no additional features are added to the model.

The forecasts done by the best model are in Figure 6.25. In this subsection, the best result using LSTMs outperformed the best result using Feed-forward Neural Networks, which is the opposite to what was tested in Subsection 4.4.5.

Set of tests	**1**	**2**	**3**	**4**	**5**
RMSE	113.7012	122.9188	122.8575	118.4815	121.5637

Table 6.16: RMSE for the tests with different combinations of additional features, for the best model, for predicting the number of users in the 4G network.

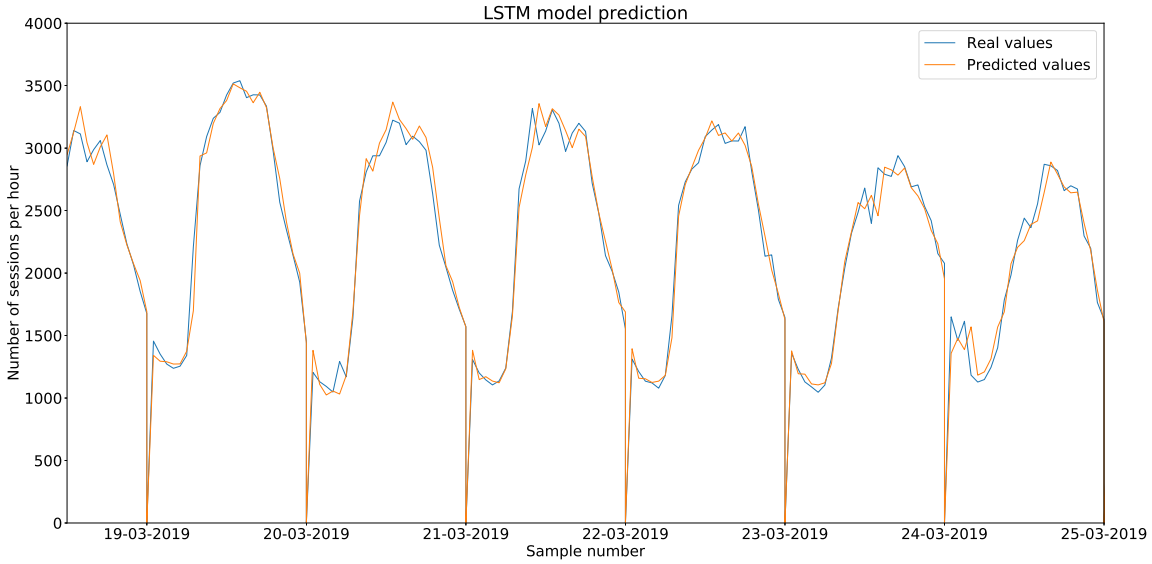

Figure 6.25: Forecasts made by the best recurrent neural network model in the cross-validation set compared with the real values, with a RMSE of 113.70 connected users per hour.

6.4.6 Ensemble tests

The last method is to create ensemble forecasts with the best models. The output will be the average of the forecasts of the various models.

The models used in these tests will be the best feed-forward neural network model (RMSE of 115.54 maximum connected users per hour), the best LSTM model (RMSE of 113.70 maximum connected users per hour) and the best result with the classical Machine Learning (ML) models (SVR algorithm, with a RMSE of 127.49 maximum connected users per hour). Because the neural network models have better results than the SVR algorithm, an ensemble of only both neural networks will be tested. Then, an ensemble of the three models will also be created to check if the SVR forecasts improve the final result.

The results are in Table 6.17. The forecasts with the SVR algorithm increase the RMSE error.

The best result in the cross-validation set is achieved with an ensemble of the feed-forward neural network and the LSTM network, where it is possible to achieve an RMSE as low as 104.94 maximum connected users per hour. The forecasts are in Figure 6.26, compared with the real cross-validation values.

Models	**RMSE**	**MAPE**	**RMSET σ=0.2**	**RMSET σ=0.5**	**RMSET σ=0.8**	**DRMSET σ=0.2**	**DRMSET σ=0.5**	**DRMSET σ=0.8**
FFNN and LSTM ensemble	104.94	7.55	107.20	113.18	114.47	131.83	82.16	123.10
FFNN, LSTM and SVR ensemble	108.15	7.61	110.47	115.77	116.58	140.28	95.79	148.52

Table 6.17: Performance metrics results for the forecasts done with the ensemble models.

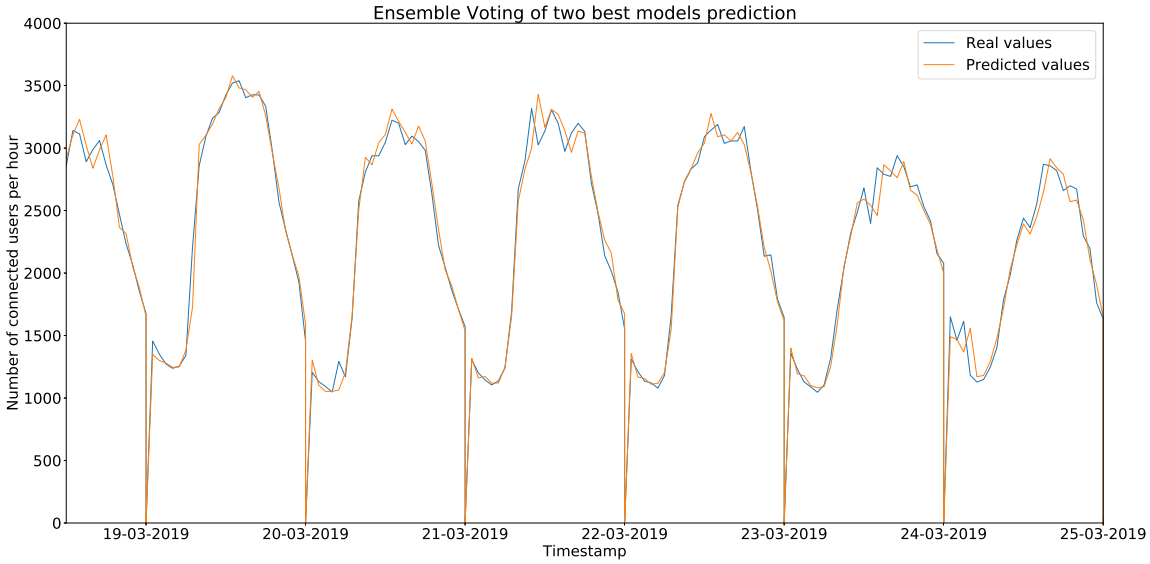

Figure 6.26: Foreacsts made by the ensemble model with two predictors in the cross-validation set compared with the real values, with a RMSE of 104.94 maximum connected users per hour.

6.4.7 Final tests

The final tests will be done by training the best models with the train and cross-validation sets, and testing with the test set (from the 25^{th} of March to the 31^{st} of Match, as described in Figure 6.18). The following models will be tested:

- best feed-forward neural network (FFNN) configuration;
- best recurrent neural network (LSTM) configuration;
- algorithm with the best result of the classical machine learning approach (SVR);
- ensemble with the best feed-forward neural network and the best recurrent neural network;
- ensemble with the best feed-forward neural network and, the best recurrent neural network and SVR.

The ensemble forecasts will be done by averaging the forecasts of all models in the ensemble. The results are in Table 6.18. On average, the error is higher than in the cross-validation set, due to the data distribution not being so similar to the training set. The ensemble with the feed-forward neural network and the recurrent neural network has the best result, just like in the previous subsection. The forecasts made by that ensemble model can be seen in Figure 6.27.

The performance metric DRMSET (σ=0.8) is not in the results because there were no data points to be considered above that threshold.

Models	**RMSE**	**MAPE**	**RMSET** $\sigma = 0.2$	**RMSET** $\sigma = 0.5$	**RMSET** $\sigma = 0.8$	**DRMSET** $\sigma = 0.2$	**DRMSET** $\sigma = 0.5$
FFNN network	153.47	9.13	156.77	168.34	153.81	262.28	174.76
LSTM	177.54	10.65	181.36	203.00	158.31	379.83	512.12
SVR	197.13	11.88	201.37	219.26	171.68	409.55	506.07
FFNN and LSTM ensemble	152.41	9.08	157.73	174.16	147.70	308.36	342.48
FFNN, LSTM and SVR ensemble	164.53	9.62	168.07	186.19	153.38	337.72	395.59

Table 6.18: Performance metric results with the best models for the test set.

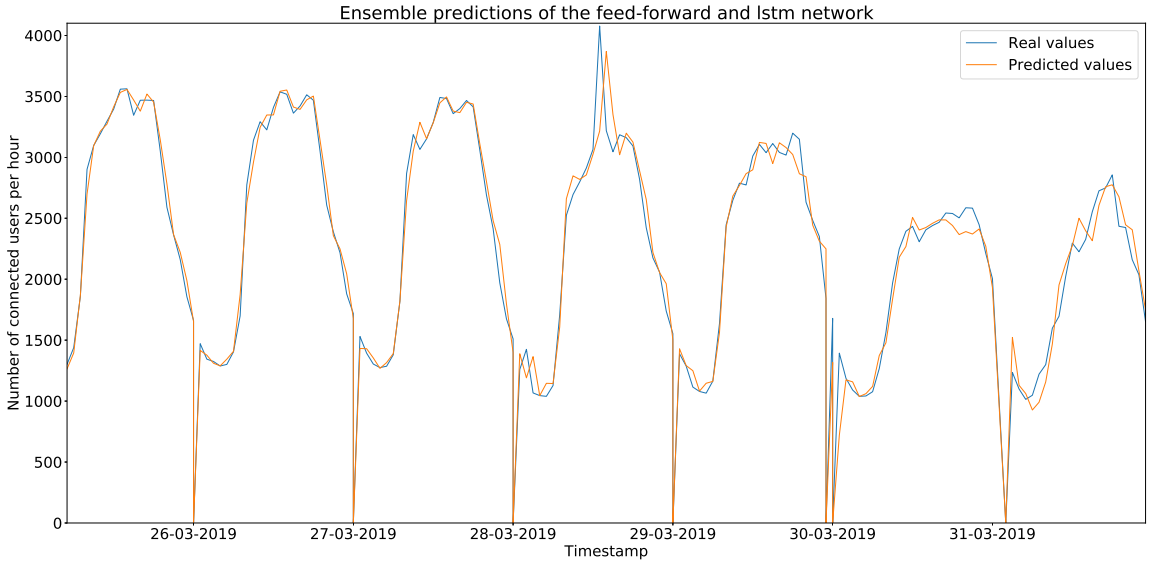

Figure 6.27: Forecasts made by the ensemble model with two predictors in the test set compared with the real values, with a RMSE of 152.41 connected users per hour.

6.5 Conclusion

In this chapter, the goal was to design a model capable of forecasting the maximum number of connected users in the next hour in a 4G network. The results show that the best model has an average RMSE of 152.41 connected users, using an ensemble model with an FFNN and an LSTM.

The results show that the LSTM network is more accurate than the FFNN, something that was not true for the forecasting done in the Chapter 4 (the FFNN had the best result). That can be explained by two factors. Firstly, the data is different. Because the maximum number of connected users in a network is an aggregate among various cells, it has smoother patterns and fewer peaks than the number of sessions in the Chapter 4. Secondly, because the number of tests with the FFNN was bigger than the number of tests with the LSTMs in the Chapter 4, the best result achieved maybe does not indicate that the FFNNs are better at the task than LSTM, but their result was better due to the higher hyperparameter search space.

With the model designed in this chapter and the model designed in Chapter 4, it is easier for the network operator to manage the network traffic in a 5G network with different slices (in this case, a slice for legacy mobile networks and a slice for vehicular networks), by forecasting the number of users/sessions in each slice. However, other metrics are also important to forecast in a network.

For example, for a network operator it is important to understand not only when, but why the network accessibility is lower than usual. In the next chapter, a study is made about what KPIs are most important to forecast low network accessibility, to find out what causes the low accessibility of a network.

7

Root Cause Analysis of Low Network Accessibility

Besides predicting network metrics, it is also important for the network operator to understand what are the causes of problems in the network. In this chapter, using the 4G network dataset (Section 6.1), will be used machine learning techniques to understand what network KPIs can indicate that a low accessibility event will happen in the future. The results will show what are the most important KPIs to forecast low network accessibility in the whole network, as well as per cell.

7.1 Description

7.1.1 Problem Description

The metric used to indicate low accessibility in the network will be the number of E-UTRAN Radio Access Bearer (E-RAB) setup failures per hour in the network. The E-RAB setup in a 4G network is a major KPI for accessibility. The E-RAB is a bearer that the User Equipments (UEs) need, to establish communications in the network. Figure 7.1 shows the E-RAB setup phase. After the UE has established a connection with the E-UTRAN Node B (eNB), it is needed to set up a context with the Mobility Management Entity (MME), to enable the UE to communicate and send data to the network.

When the MME sends a context setup request, it is called an E-RAB setup attempt. After some configuration messages between the UE and the eNB, the context setup response from the eNB to the MME is called an E-RAB setup success. There are more E-RAB setup attempts than E-RAB setup successes. When the network is congested, the difference between the E-RAB setup attempts and the E-RAB setup successes is higher, because some messages after the E-RAB setup attempt are lost due to network problems, such as congestion.

A new accessibility metric will be used to measure the accessibility of the network: the number of E-RAB setup failures. If the number of E-RAB setup failures has a high value, the network is congested. Its formula is described in Equation 7.1, where eef is the number of E-RAB setup failures, eea is the number of E-RAB setup attempts and ees is the number of E-RAB setup successes.

$$esf(t) = esa(t) - ess(t) \tag{7.1}$$

The E-RAB setup failure in the period from the 23^{rd} of April of 2019 to the 23^{rd} of May of 2019 is in Figure 7.2. The biggest congestion happened on the 9^{th} of May of 2019. The objective is to understand what KPIs most contribute to the forecast of this metric.

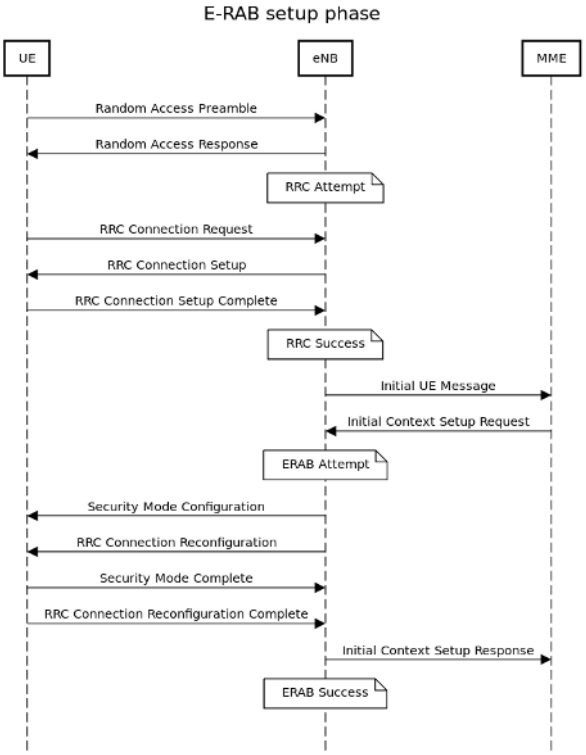

Figure 7.1: Sequence diagram indicating the E-RAB setup phase.

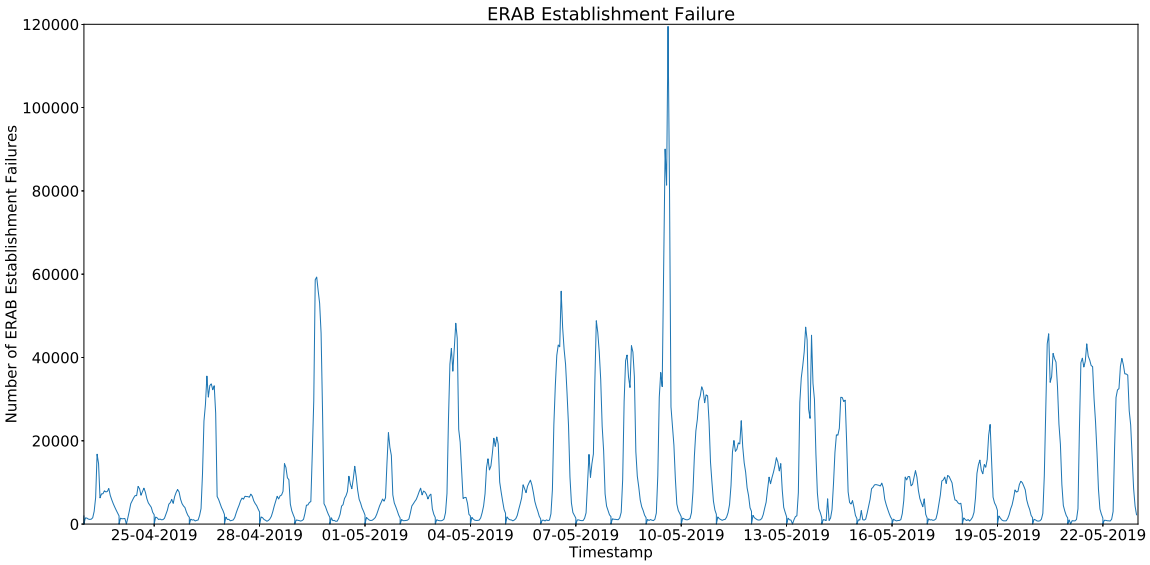

Figure 7.2: Number of E-RAB setup failures.

7.1.2 Forecasting Approach to Predict Lower Accessibility

The first test to check if any KPI can accurately forecast the number of E-RAB setup failures is to calculate the Pearson correlation coefficient between the number of E-RAB setup failures and all other KPIs. Because the goal is to understand what KPIs can forecast low accessibility before the number of E-RAB setup failures increases, the KPI values will be shifted (lagged) behind the number of E-RAB setup failures by one hour (the values are sampled hourly; one hour is the minimum time interval).

There are 12 KPIs with a correlation coefficient higher than 0.6 (Table 7.1; all KPIs are described in Appendix A - 4G Network KPIs). The number of E-RAB setup failures is highly correlated with itself lagged by one hour. The handover intra-frequency failure is the KPI that has a higher correlation coefficient with the number of E-RAB setup failures. The number of E-RAB abnormal releases, the maximum number of connected subscribers and RRC counters are also highly correlated with the number of E-RAB setup failures.

There are two disadvantages of analyzing the KPIs importance with the Pearson correlation coefficient. For each KPI, it is only measured its linear contribution to the number of E-RAB establishment failures. Besides, combinations of KPIs that can be important are not being taken into account, because it is assumed that the KPIs are independent of each other. There are other approaches to measure the importance of input features to an output value that take into account these considerations.

These approaches take advantage of machine learning techniques and they can be divided into two categories: some approaches take into account the error of the model in a test set to calculate the importance of the input features; other approaches measure the feature importance with an internal calculation (algorithm-specific) of the coefficients associated with each input feature.

One of the approaches that fall into the first category is the drop column feature importance. In this approach, the importance of a feature is measured by comparing the test error of a model when all features are available as input, with the test error of a model when one feature is dropped for training. The higher the error for the model with one feature dropped, the more importance is given to that feature.

A big disadvantage of this approach is that, for each feature it is needed to train a new model with that feature dropped, which causes the approach to be inefficient for many features, or for models that take considerable time training. In [113] (permutation feature importance approach), a similar approach is used without the need to re-train the model for each feature. Instead of dropping the

KPIs (lag=1)	**Pearson correlation coefficient**
E-RAB Setup Failure	0.912007
Handover intrafreq failure	0.797241
E-RAB Abnormal Release	0.718813
Maximum connected subscribers	0.697418
CSFB Prep Success	0.687440
RRC Setup Attempt	0.668815
RRC Setup Success	0.668736
E-RAB Normal Release	0.661847
RRC Setup Failure	0.640163
PDCP Download TX Time	0.626372
Handover interfreq attempt	0.603350
Handover interfreq success	0.602473

Table 7.1: Pearson correlation coefficient between the KPIs and the number of E-RAB setup failures with lag of one hour, for all the KPIs with Pearson correlation coefficient higher than 0.6.

feature, it is applied random shuffling to the test values of that feature among the various examples, to preserve the distribution of that variable. If the model error is almost unchanged, the feature is not much important for the forecast. However, if the model error increases much, it is a sign that the feature is important for the forecast.

Both previous approaches have some advantages. It is possible to apply them to black-box models, given that the feature importance is measured by the model error. They also take into account all interactions between features, an advantage when compared with the correlation tests.

In the permutation feature importance approach, a disadvantage is that the results are dependent on the shuffling of the features. If the tests are repeated, the results may vary.

The approaches that measure the feature importance by inspecting the internals of the models are algorithm-dependent. For some algorithms, like neural networks or SVMs, it is impossible to calculate the importance of each feature, due to the non-linear transformations applied. However, for other algorithms such as Logistic Regression, Extra Trees, Random Forest, Gradient Boosting or AdaBoost, it is possible to estimate the importance of each feature. For linear regression, the importance of each feature can be measured by the absolute value of the coefficient associated with each input (if all features are within the same scale). For the other four tree-based ensemble algorithms (Extra Trees, Random Forest, Gradient Boosting and AdaBoost), the feature importance is calculated with resource to the mean decrease impurity (or Gini impurity). The importance of a node j in a decision tree is computed as described in Equation 7.2, where w_j is the weighted number of samples in node j, C_j is the impurity of this node, and $left(j)$ and $right(j)$ are the respective children nodes.

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)} \tag{7.2}$$

The feature importance of feature i across all nodes is computed as described in the Equation 7.3[1].

$$Fi_i = \frac{\sum_{j:\text{node j splits on feature i}} ni_j}{\sum_{j \in \text{all nodes}} ni_j} \tag{7.3}$$

For this approach, it is important that the features are all normalized within the same scale, and it is recommendable that they are from the same type (continuous/categorical) for better importance estimation.

The advantage of the approaches that measure the feature importance by inspecting the internals of the models is that they do not depend on the test set, only on the model. If the model is accurate and it is not underfitting or overfitting, the feature importance can be calculated more reliably than the previous methods. Otherwise, the feature importance will be highly biased. The biggest challenge is to create accurate models that do not overfit the train set. The second category of approaches (measure feature importance with an internal calculation algorithm-specific) will be used to measure the importance of the KPIs.

If all KPIs are included as features, the performance of the model will be degraded, because some features are uncorrelated with the output and do not contribute to the output classification. It is important to perform feature selection. The ideal scenario is to test all combinations of features in the input, and determine the best features by the test error. However, it is not feasible to test due to the high number of combinations.

The approach chosen was to use a dimensionality reduction algorithm to reduce the number of features. Principal Component Analysis (PCA) [114] will be used as the algorithm to perform

[1] `https://stats.stackexchange.com/questions/311488/summing-feature-importance-in-scikit-learn-for-a-set-of-features/`

dimensionality reduction. The five algorithms chosen to train the model and to calculate the feature importance were the following: Logistic Regression, Extra Trees, Random Forest, Gradient Boosting and AdaBoost. The feature importance of each model will calculate the importance of each PCA component. Each component importance is then multiplied by the PCA coefficients, to get the KPIs importance for each component. Finally, all the importances of the same KPIs are added, as described in Equation 7.4, where fi_j is the feature importance of the j PCA component and $pca\_coefficient(j,i)$ is the PCA coefficient i for the component j, where i is the number of a KPI and j is a PCA component.

$$kpi\_i_i = \sum_{j=0}^{n\_pca\_components} fi_j * pca\_coefficient(j,i) \tag{7.4}$$

To get the best possible model, the number of PCA components cannot be too small, because relevant features can be lost, but it also cannot be too big, with the risk of creating overfitted models. The number of PCA components tested will vary from 1 to the number of KPIs, for the five algorithms. The model with a lower test error will be used to calculate the KPI importance.

The input values will be based on the KPIs. There will be two types of input values: normalized values and normalized variations. For each KPI, the normalized value of the previous hour will be used as input. The number of lags could be increased besides one hour, but in this test it is considered that the low accessibility indicators appear at most one hour before the congestion. For each KPI, the normalized variation will be calculated according to the Equation 7.5. The normalized variations are added as features because the network accessibility can depend not only on the previous values, but also on sudden variations of other KPIs.

$$normalized\_variation(t) = \frac{value(t-1) - value(t-2)}{value(t-2)} \tag{7.5}$$

Two types of classification problems will be used to calculate the importance of the KPIs. It is important to understand what KPIs are more important when forecasting the possibility of a low accessibility event in a network. It is also important to understand what KPIs are more important to forecast sudden increases and decreases in network accessibility. For each case, a different output value will be calculated. For the first case, it will be set a threshold in the 90^{th} percentile of the data, with all the values above the threshold being classified as one, and all other values as zero, turning the problem into a binary classification problem. For the second case, the output will be classified as one if the absolute difference between two consecutive values is higher than the 90^{th} percentile of the data, and zero otherwise. Doing that, it is possible to analyze what KPIs are most important for classifying low accessibility events and also for forecasting bigger increases and decreases in data, which can be important for resource allocation.

The two classification problems will be applied in two different ways, according to the data split. Firstly, the tests will be done with aggregated data. The network KPIs will be added and will be taken into account in the whole network. In this way, it is possible to forecast low accessibility of the network. Another way of performing the tests is to split the data per cell. In this approach, the data will not be aggregated, and 75% of all cells in the network will be used to train, with the other 25% cells to test. The threshold for the tasks will be defined using the specific values of each cell (each cell will have a different threshold, based on its values). With that, it is constructed a model that is capable of detecting low accessibility per cell. This case is expected to perform worse than the aggregated network, because forecasting low accessibility per cell is harder than forecasting low accessibility in the network. However, for a network operator it is very important to forecast low accessibility per cell, for various reasons. For a cell in a crowded region, it can be made management adjustments to avoid

the low accessibility of the cell, such as the installation of another temporary cell, or the allocation of resources for that cell. Besides, it can be made a time-series analysis about the future accessibility of the cells in a region, for expansion purposes.

For both classification problems, a performance metric must be used. Because in the problems described the number of positive samples is lower than the number of negative samples, the accuracy performance metric would give similar cost to the false negative and false positive errors. The performance metric used will be the F1-score (Equation 7.6). The F1-score is the harmonic mean of precision (Equation 7.7) and recall (Equation 7.8), and it encompasses the False Negative and False Positive errors, weighted according to the number of samples of each class.

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{7.6}$$

$$precision = \frac{true\_positives}{true\_positives + false\_positives} \tag{7.7}$$

$$recall = \frac{true\_positive}{true\_positives + false\_negatives} \tag{7.8}$$

7.2 Results

7.2.1 Aggregated network tests results

In this subsection, the results for the aggregated network tests will be presented. As explained before, two scenarios were proposed. Firstly, it will be presented (i) the most important KPIs for predicting if the number of E-RAB establishment failures is above a threshold. Secondly, it will be presented (ii) the most important KPIs for predicting if the number of E-RAB establishment failures has high variations.

The best model for scenario (i) was with the Extra Trees algorithm, achieving an F1-score of 86.6%, with 23 PCA components. With a higher number of PCA components, the performance of most of the algorithms starts to deteriorate (Figure 7.3).

Table 7.2 shows the ten KPIs that were considered more important for the task (i). The handover failure is considered to be the most important KPI, both inter and intra frequency. The third KPI more important is the Circuit Switched Fallback (CSFB) preparation success. The CSFB is a technology to create circuit-switched calls over a 4G network that does not support LTE voice call standard (VoLTE), which has to fall back on the 3G network. This indicates that the number of phone calls and SMS messages has a high impact on network accessibility.

KPI	**Importance**
Handover interfreq failure	0.194
Handover intrafreq failure	0.188
CSFB Prep Success	0.181
PDCP Download TX Time	0.169
Variation Handover interfreq failure	0.150
Cell availability	0.131
ERAB Normal Release	0.124
Handover intrafrequency success	0.121
Handover intrafrequency attempt	0.121
PDCP Upload Volume (Mb)	0.115

Table 7.2: Ten KPIs that were considered more important for the scenario (i) for the aggregated network.

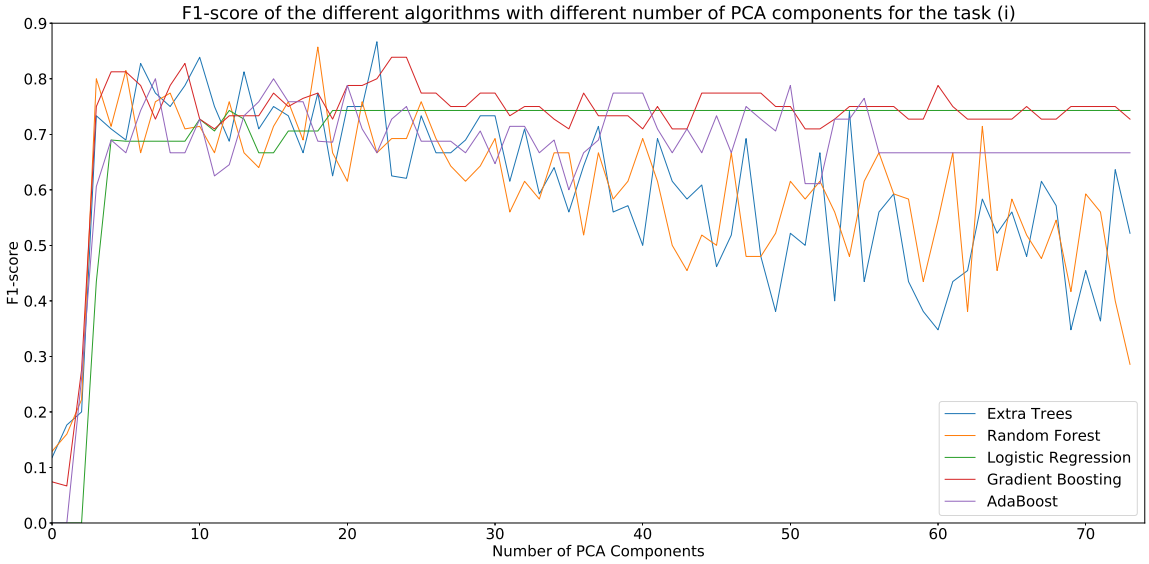

Figure 7.3: F1-score with different algorithms varying the number of PCA components for the task (i) for the aggregated network.

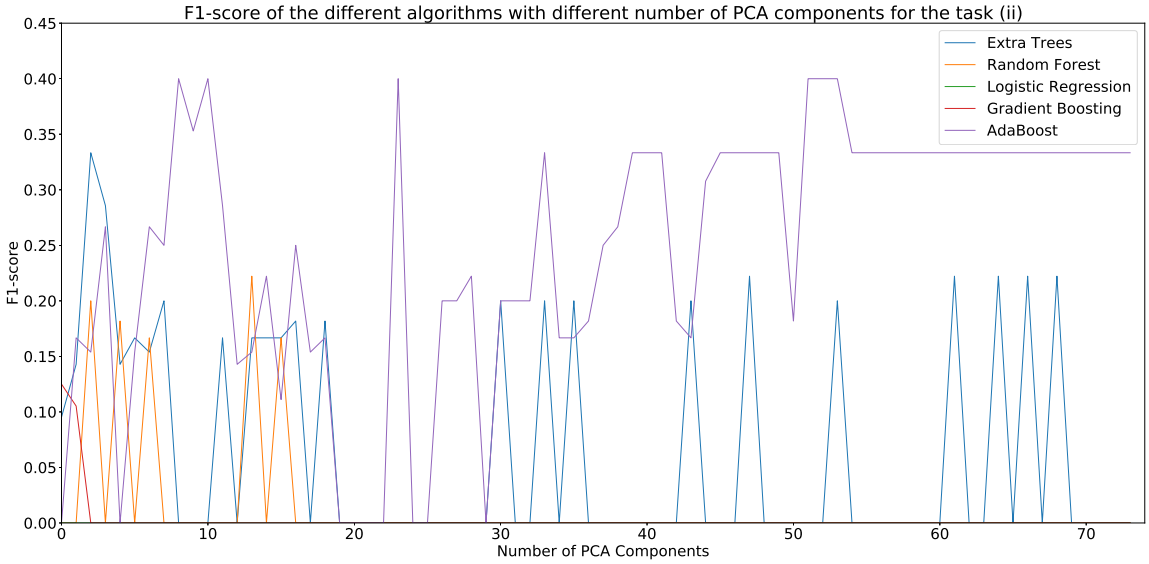

Figure 7.4: F1-score with different algorithms varying the number of PCA components for the task (ii) for the aggregated network.

The best model for the scenario (ii) was achieved with the AdaBoost algorithm, achieving an F1-score of 40.0%, with 24 PCA components. The F1-score is lower than in the scenario (i) because the task of predicting variations is harder than predicting if the value is above a threshold. The number of PCA components is almost the same as in the scenario (i); however in Figure 7.4 it can be seen that the algorithms' F1-score is very unstable, and that it is hard to build a model to predict accurately the variations of the number of E-RAB establishment failures for the aggregated network.

Table 7.3 shows the ten KPIs that were considered more important for the task (ii). The handover failures are still important, but they are not the most important KPI. The CSFB preparation success is the most important KPI to predict the variations in the number of E-RAB setup failures. Similar to the task (i), the PDCP download transmission time, the variation of the inter-frequency handovers

KPI	Importance
CSFB Prep Success	0.188
Handover intrafreq failure	0.170
PDCP Download TX Time	0.157
Handover interfreq failure	0.148
Variation Handover interfreq failure	0.141
Download Active Subscribers (Max)	0.138
Active subscribers (Max)	0.125
RRC Setup Failure	0.121
Cell availability	0.120
Upload active subscribers (Max)	0.119

Table 7.3: Ten KPIs that were considered more important for the scenario (ii) for the aggregated network.

and the cell available time are also important KPIs to predict variations in the number of E-RAB setup failures.

From the results from both tasks, it can be concluded that most KPIs that cause the number of E-RAB setup failures in a network to be above a threshold are the same that cause it to have high variations. Those KPIs are the number of failure handovers (intra and inter-frequency), the CSFB preparation success, the PDCP download volume and the cell availability. The maximum number of active subscribers (downloading subscribers and overall subscribers) also causes the number of E-RAB setup failures to varying.

Interpreting the KPIs, the results achieved are according to the intuition about lower network accessibility. When the number of failure handovers is high, the cells are crowded with user sessions and cannot accept any more sessions, which leads to lower network accessibility in the next hour. The high number of CSFB preparation success shows that there is a clear relationship between the high number of phone calls and SMS messages in the network with its lower accessibility. The KPIs of PDCP download volume and the maximum number of active subscribers also show that the number of active subscribers and their download volume influence the network accessibility (more than the number of connected subscribers). Finally, the cell availability indicates that if many cells are unavailable in the current hour, it is likely that the network accessibility will be lower in the next hour.

7.2.2 Individual cells tests results

In this subsection, the results for the individual cells will be presented. 75% of the cells in the network will be used for training and the other 25% for testing. Just like in the previous subsection, firstly it will be presented (i) the most important KPIs for predicting if the number of E-RAB establishment failures is above a threshold. Secondly, it will be presented (ii) the most important KPIs for predicting if the number of E-RAB setup failures has high variations.

The best model for scenario (i) achieved an F1-score of 30.79%, using the Gradient Boosting algorithm, with 66 PCA components. For cell prediction, more information is needed to obtain the best model when compared with the previous subsection. Figure 7.5 shows the F1-score varying with the number of components and the different algorithms. In the PCA component 62 there is an increase in the F1-score of all algorithms.

Table 7.4 shows the ten KPIs that were considered more important for the task (i). Two KPIs have much more importance than all others: Radio Resource Control (RRC) setup success ratio ($\frac{\text{RRC setup success}}{\text{RRC setup attempt}}$) and E-RAB setup success ratio ($\frac{\text{ERAB setup success}}{\text{ERAB setup attempt}}$).

The best model for scenario (ii) has an F1-score of 20.39%, using the AdaBoost algorithm with 68 PCA components. Figure 7.6 shows that the F1-score for different PCA components is similar to the

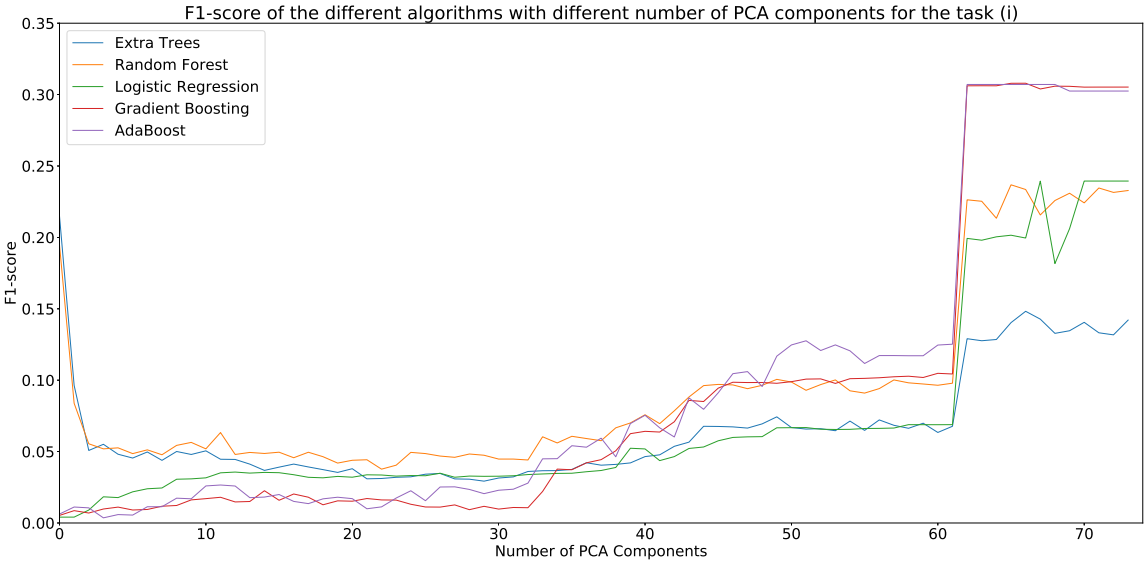

Figure 7.5: F1-score with different algorithms varying the number of PCA components for the task (i) for the individual cells tests.

KPI	**Importance**
RRC Setup Success Rate	0.560
ERAB Setup Success Rate	0.547
Connected subscribers (Max)	0.069
Connected subscribers (Avg)	0.066
Connected active subscribers (Avg)	0.043
Variation Connected Subscribers (Max)	0.034
Variation Connected active Subscribers (Max)	0.032
ERAB normal release	0.031
Variation Radio Bearers (Avg)	0.031
Variation Connected subscribers (Avg)	0.031

Table 7.4: Ten KPIs that were considered more important for the scenario (i) for the individual cells tests.

scenario (i), where the F1-score improved its performance significantly after 60 PCA components.

Table 7.5 shows the ten KPIs that were considered more important for the scenario (ii). Like in the previous scenario, RRC setup success ratio and E-RAB setup success rate are the most important KPIs for the output. In this task, other KPIs have also similar importance, such as the variation of the cell availability, the average connected subscribers and the average number of radio bearers. A radio bearer is a connection between the eNB and the UE at layer 2, and defines the communication configurations for upper layers.

The results of the forecasts of low network accessibility for individual cells had higher error than the results with the aggregated network. In the tests done, the best models needed more data than the aggregated network models to achieve the best result, because more features were needed to be able to generalize the predictions for different cells.

The results from both tasks show that, just like in the aggregated network tests, the most important KPIs for forecasting the number of E-RAB setup failures in a cell to be above a threshold are the same that cause it to have high variations. However, the most important KPIs that cause lower accessibility in a network are different from the KPIs that cause lower accessibility in a cell. It is essential for a network operator to monitor the right KPIs for the different tasks: forecast lower accessibility in a

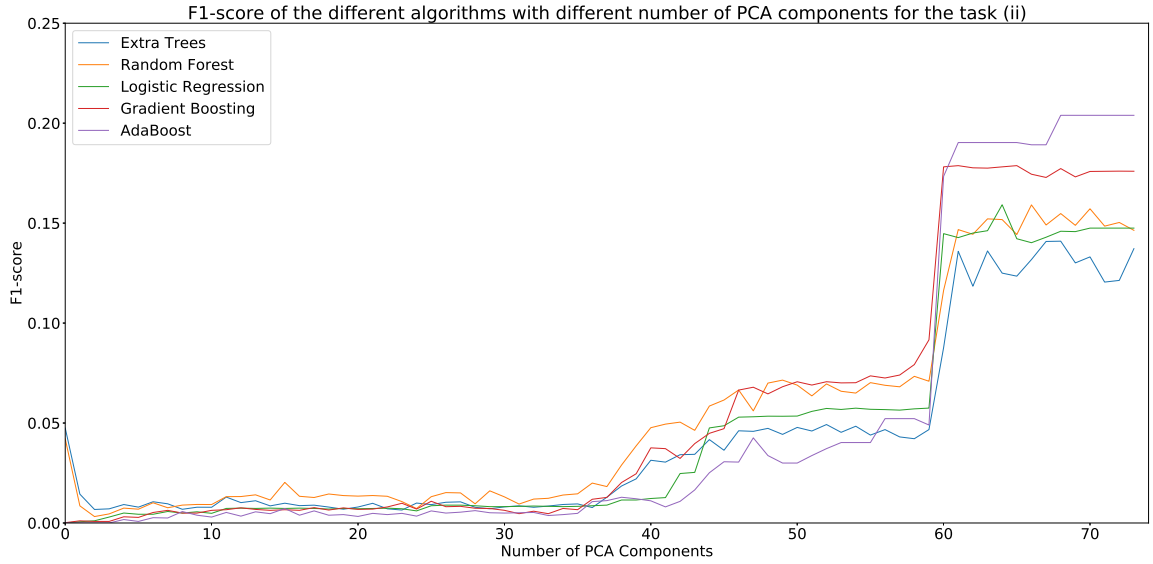

Figure 7.6: F1-score with different algorithms varying the number of PCA components for the task (ii) for the individual cells tests.

KPI	**Importance**
ERAB Setup Success Rate	0.147
RRC Setup Success Rate	0.147
Variation Cell availablility	0.138
Connected Subscribers (Avg)	0.102
Radio Bearers (Avg)	0.095
Variation RRC Setup Success Rate	0.093
Variation ERAB Setup Success Rate	0.092
ERAB Normal Release	0.088
Connected subscribers (Max)	0.087
Variation Radio Bearers (Avg)	0.081

Table 7.5: Ten KPIs that were considered more important for the scenario (ii) for the individual cells tests.

network or forecast lower accessibility per cell.

The two most important KPIs for both tasks are the RRC and the E-RAB setup success rate. These KPIs cannot be understood as the cause for lower network accessibility, but as a consequence. Because they are intrinsically related to the E-RAB setup failure, being themselves accessibility metrics, they can be understood as an indicator of the high autocorrelation between consecutive hours. These results show that the network accessibility per cell is highly dependent on the network accessibility of that cell in the previous hour. If a cell has lower network accessibility in an hour, it is likely that the network accessibility in the next hour for that cell is still low, and vice-versa.

As opposed to the results for the aggregated network, the important KPIs for low network accessibility are not related with the CSFB preparation success, or with any of the handover metrics. The results by cell show that besides the RRC and the E-RAB setup success ratio, the most important KPIs are counters related to the number of users in a cell and its utilization: maximum number of connected subscribers, average number of connected subscribers, average number of active subscribers downloading data, variation of the maximum number connected subscribers, variation of the maximum number of active download subscribers, average number of radio bearers or variation of the average number of radio bearers. It is expected that, as these KPIs have higher values, the accessibility of a

cell decreases.

7.3 Conclusion

In this chapter, it was concluded that for predicting low accessibility in a network, it is needed to monitor different KPIs than when predicting low accessibility in individual cells. Predicting low accessibility in individual cells is a harder task, because the prediction goal is more diverse (there are more different values, due to the different patterns between cells). However, it was possible to predict if the number of E-RAB setup failures was above or below a threshold with 86.6% of F1-score in the network, and with 30.79% of F1-score in individual cells.

For the network, the most important KPIs to monitor are the handover failures, the CSFB preparation success and the PDCP download transmission time. For the individual cells, the two most important KPIs were the E-RAB setup success ratio and the RRC setup ratio. However, these KPIs are consequences of low accessibility, not causes. Other important KPIs that can be monitored to predict low network accessibility in a cell are network counters about the usage of the cell, such as the maximum number of connected subscribers, the average number of connected subscribers, the average number of active subscribers downloading data or the variation of the maximum number of connected subscribers.

As future work, the next step would be not only to understand what KPIs are the most important for predicting low accessibility, but also to detect the patterns in those KPIs that indicate future low accessibility, to be able to predict it and adapt the network to prevent it to happen. For example, if the network operator knows that the network accessibility will be lower in the next hour when the number of handover failures intra-frequency and the maximum connected users both exceed a threshold, he can take pro-active measures to adapt the network and avoid the low accessibility.

In the next chapter, it will be presented a prediction architecture designed for easy integration with the 5G management frameworks previously presented (Subsection 2.3.4), where it is possible to include different prediction models for different network traffic metrics, such as the models presented in Sections 4, 5 and 6.

8

Real-time Distributed Time-Series Prediction Architecture

In 5G networks, with the expected increase in data volume consumption and with the slicing of the network according to specific requirements, the prediction of relevant metrics is essential for the pro-active management of a network. To perform the prediction of network metrics, it is needed to implement a reliable and scalable architecture, compliant with the real-time needs and distributed nature of 5G networks, as well as with the current *de-facto* industry standards. Particularly, the interface of the architecture to perform predictions must be a uniform simple online service, accessible via Hypertext Transfer Protocol (HTTP), to easily integrate and communicate with the 5G management frameworks. In this chapter, it is described the proposed real-time distributed time-series prediction architecture.

The goal of the prediction architecture is to predict various metrics in the network, being composed of several metric predictors. Each metric can be specific to a slice (e.g. the number of sessions per hour in a particular slice), or it can be an aggregated value (e.g. network traffic in a particular node in the network). The models developed in Sections 4, 5 and 6 will be implemented in this architecture.

8.1 Architecture description

The holistic view of the architecture (Figure 8.1) is composed of five main components: the Prediction API, the Predictor Message Broker (PMB), the Metric Prediction Component (MPC), the Metric Message Transformation Component (MMTC) and the Metric Prediction Orchestrator (MPO).

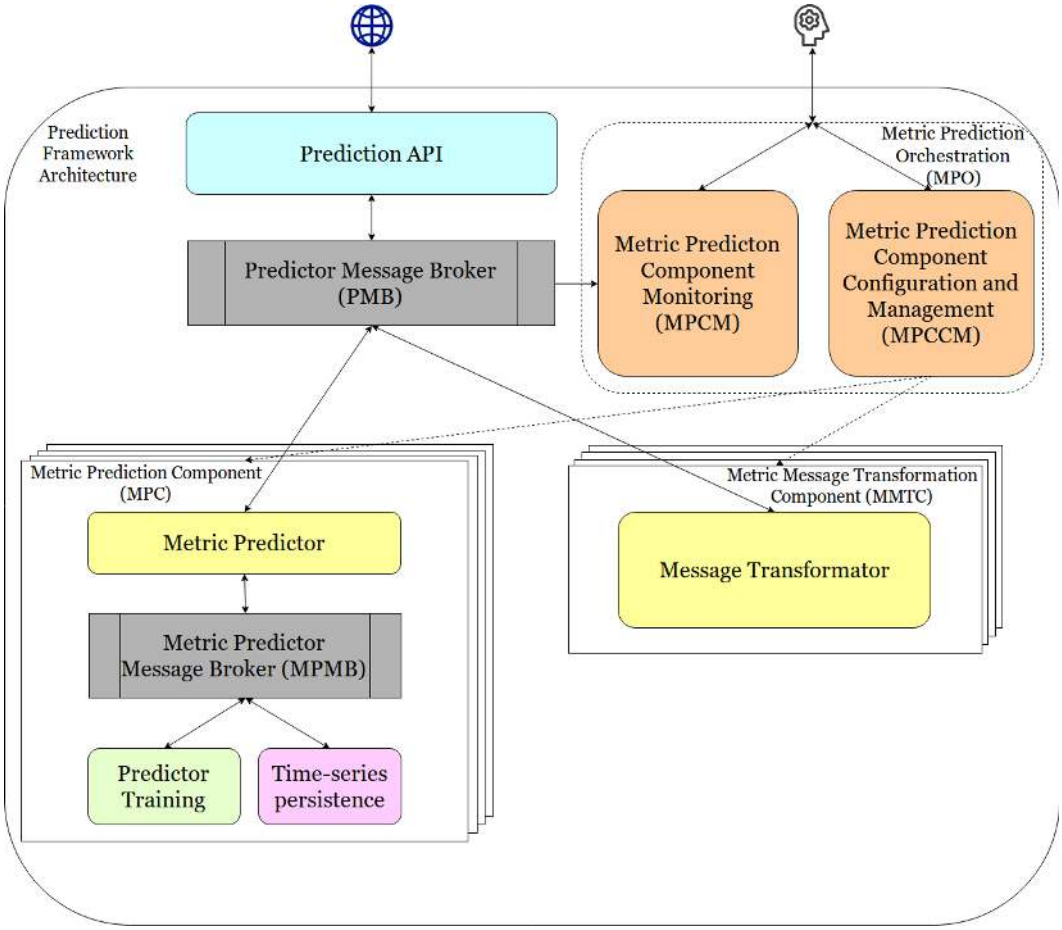

Figure 8.1: Overall architecture of the prediction architecture.

Prediction API

The purpose of this component is to abstract and encapsulate in a convenient way the functionalities of the metric predictors without any operability loss. To enable the integration with the 5G management frameworks, the prediction architecture interface must implement several endpoints in a Representational State Transfer (REST) web-service.

There are eight endpoints:

- **Train** - trains a metric predictor component. The client can specify the start date and the end date of the training data. It can also send a list of attributes that are specific to each metric predictor;
- **Change training task** - changes the configurations of the training schedule. The client can specify the date of the next train, the time between two consecutive training phases and the period of data to consider in each training. It can also specify additional attributes specific to each metric predictor;
- **Predict** - predicts the next value of the time-series sequence. The client sends the actual time-series point and receives the prediction of the next one. A time-series point is characterized by a numeric value, a timestamp and additional attributes, specific to the metric predictor;
- **Save** - saves the time-series point in the predictor, for further training;
- **Predict and save** - predicts the next time-series point and saves the actual time-series point it in the predictor;

- **Rebuild internal state** - each predictor can have an internal state, composed of the previous time-series points. This action rebuilds the predictor internal state. The client can specify a start date and an end date for the data to rebuild the internal state;
- **Clear internal state** - clears the predictor internal state;
- **Get predictor information** - retrieves information about the metric predictor. The information is composed of its name, the date of next scheduled training, the period between training phases, the period of data to fetch in each training and additional attributes specific to each predictor.

Predictor Message Broker (PMB)

When the Predictor API receives a request, according to an internal message protocol, it sends a message to the PMB. This component will mediate the messages between the Network Predictor API, the MPC and the MMTC. It makes no assumption about the messages. It enables the multicast and broadcast distribution of messages among the Metric Predictors, allowing the network manager to build several predictors for the same metric, building an ensemble predictor.

The Predictor API and message broker must provide fault tolerance, disaster recovery, low response times and high scalability, because the message throughput in these components can be high, and the failure or high latency of any of these components will prejudice the entire framework performance.

Metric Prediction Component (MPC)

The core component of the framework is the MPC. Each instance of the MPC is responsible for predicting a network metric. It receives the messages from the PMB and predicts the time-series points. This component is composed of four subcomponents: the Metric Predictor Message Broker (MPMB), the Metric Predictor, the Predictor Training and the Time-series Persistence.

The MPMB is the internal message broker of the component and it routes messages between the other three sub-components. The Metric Predictor has the goal of processing the requests coming from the PMB and making the predictions according to a specific model. The Predictor Training does online or offline training of the model to update it with the newly received data. For offline training, the training task parameters (date of the next training, training data window, etc.) can be adjusted via the Prediction API. The new model will be uploaded to the Metric Predictor via MPMB. The Predictor Training subcomponent is not included in the Metric Predictor due to the requirements of the Metric Predictor of demand, response time and flexibility. To handle fast prediction for many requests per second, it is recommendable that such CPU-intensive tasks, such as the training of a model, to be offloaded to a different computing engine. The Time-series persistence has the intent of storing the time-series values for later use in training phases and on building the internal state of the metric predictor. The update and deletion of values are rarely used, and the most common operations are the insertion and retrieval of values. It must also be noted that the additional attributes for each predictor also have to be stored along with the values and the timestamp.

Metric Message Transformation Component (MMTC)

To enable an ensemble prediction, it is also needed to perform a reducing step, to join the predictor results and perform the adequate transformation. The MMTC performs that transformation by receiving the messages from the broker and applying custom functions for each predictor component messages, before inserting the transformed messages into the broker. The transformer can also be used for other data transformation operations, such as data unit transformations or data anonymization.

Metric Prediction Orchestrator (MPO)

To orchestrate the MPC and the MMTC, the MPO can be used by the prediction framework manager. It has two internal components: the Metric Predictor Component Monitoring (MPCM) and the Metric Predictor Configuration and Management (MPCCM). The MPCM can access the logs produced by the MPC and by the MMTC (the logs are sent to the PMB) and their exchanged messages, to better understand the current state of operation of the various components. The MPCCM is able to instantiate and manage the life cycle of the MPC and MMTC. These components can be dynamically added or removed when ordered by the prediction framework manager. After the MPCCM instantiates a component, it also ensures its availability and reliability, restarting it when it crashes. When a component is instantiated, it starts its execution by establishing a connection with the PMB, to be able to report its activity to the MPCM.

8.2 Implementation

In this section, the implementation of the prediction architecture is described, as well as the choice of the tools used.

Network Predictor API

The Network Predictor API was implemented using the programming language Elixir[1]. Elixir is a functional language created in 2012 that runs on the Erlang[2] virtual machine. Its main characteristics are the native concurrency, fault tolerance, high availability, fast development, elegant syntax and low latency [115]. It was also used the Phoenix Framework[3], known by its simplicity and its speed. With the main task of this component being message passing, Elixir with Phoenix are a great fit, allowing for reduced latency times, many simultaneous connections and very little downtime.

Predictor Message Broker (PMB)

For the implementation of the Predictor Message Broker, it was used the software platform Apache Kafka [116]. Kafka is a scalable, fault-tolerant, publish-subscribe messaging system designed to handle real-time data streams. It is widely used in big data architectures for publish/subscribe messaging queuing, due to its reliability and low-latency, as it can be seen in various works [117] [118] [119], and its growing adoption [120].

In the current implementation, each MPC has a queue for receiving and sending messages from and to the Network Predictor API, identified by name of the MPC. Each message is composed of three attributes:

- Type of the message - each message has a type for each action to be made. The types are:
  - 1 - Predict, Save and Predict & Save messages;
  - 2 - Train the predictor;
  - 3 - Change the training task of the predictor;
  - 4 - Clear the internal state of the predictor;
  - 5 - Rebuild the internal state of the predictor;
  - 6 - Get information about the prediction;
- Sequence number - the sequence number is a random number of 32 bits generated by the requester. The response message must have as sequence number the sequence number of the request incremented by one unit;

[1] `https://elixir-lang.org/`
[2] `https://http://www.erlang.org/`
[3] `https://phoenixframework.org/`

- Content - the content of the message. This field contains the valid attributes sent by the user to the metric predictor. In the messages of type 1, it also includes the "save" and "predict" flags, that are set to true if the time-series point sent is to be saved in the predictor, or to predict the next value accordingly, respectively.

Metric Prediction Component (MPC)

The MPMB was implemented using RabbitMQ[4]. RabbitMQ is an open-source general-purpose message broker implemented in Erlang, known by its reliability, lightweight execution and delivery guarantee. There are some differences between RabbitMQ and Kafka.

Kafka is mostly seen as a streaming platform. It is optimized for high-ingress data streams and is a durable message broker, where applications can re-process streamed data on disk. It is also used to support consumers that could be offline or that want to receive messages with low latency. It is designed with horizontal scaling in mind. Because the PMB is designed for a high volume and high throughput of data, due to the many possible MPCs that can be installed, Kafka is the most adequate message broker to use for the PMB. Besides that, it can also handle cases where the MPCs are offline by storing the messages in its queues, and horizontal scaling is straightforward.

On the other hand, RabbitMQ was designed for vertical scaling. It keeps all states of the messages for its consumers, which can overwhelm the system if there are many messages for many consumers. However, for simple use cases where the throughput of messages is not high, it is an adequate match, due to the simplicity of usage, the lightweight execution framework and low latency for a small number of consumers/producers [121]. For the MPMB, it is not expected the training of the model to be done at small time intervals neither database changes, so RabbitMQ was adopted for this component.

The Metric Predictor, Predictor training and Time-series persistence are implemented using Python 3[5]. The Training and Prediction models are implemented using the library Keras[6], using TensorFlow [107] as backend.

The Time-series persistence uses Timescale[7] for time-series data storage, an open-source time-series SQL database based on PostgreSQL[8]. Timescale is designed for fast ingest of data, large data volume storage and time-oriented reads and writes of immutable data. It does that while keeping a familiar query language and compatibility with PostgreSQL, being able to store numeric values and other types of values for each timestamp, making it suitable to store the time-series values with a timestamp, a numeric value and other fixed attributes variable for each metric predictor.

There are other time-series databases available, but they do not meet the requirements for the architecture. InfluxDB[9], another open-source time-series database, is a non-relational database, as opposed to the relational model used by Timescale, that offers more functionalities, flexibility and control. The InfluxDB query language is also database-specific and different from SQL, which introduces overhead and harder readability. Furthermore, the CPU and memory consumption are reduced when using Timescale. Finally, for insertions with moderate or high cardinality, the Timescale database outperforms InfluxDB by a large margin. Other options were also considered, such as Prometheus[10] or OpenTSDB[11].

[4] https://rabbitmq.com
[5] https://www.python.org/
[6] https://keras.io/
[7] https://www.timescale.com/
[8] https://www.postgresql.com/
[9] https://www.influxdata.com/
[10] https://prometheus.io/
[11] http://opentsdb.net/

Figure 8.2: Data visualization with Grafana plugin.

Metric Message Transformation Component (MMTC)

The MMTC is implemented in Java[12], using the Kafka Streams library[13] to allow for real-time transformation of Kafka queues.

Metric Prediction Orchestrator (MPO)

The MPCM uses the ELK (Elastic Search, Logstash and Kibana) stack[14] to monitor the behavior of the MPCs and MTCs [122], retrieving the messages exchanged and the available logs from the PMB, and performing aggregation techniques, making them available to the prediction manager.

The Network Prediction API, the PMB, the MPC and the MMTC are deployed as Docker[15] microservices, to allow for fast deployment, compatibility across different environments, security and isolation. The MPCCM is implemented using Docker Swarm to easily manage MPC and MMTC behavior and life-cycle.

It were also used the tools pgAdmin[16] for database monitoring and Grafana[17] for data visualization and analytics (Figure 8.2).

8.3 Conclusion

This chapter described the modules and the implementation of the Real-time Distributed Time-Series Prediction Architecture to be used in 5G Network Management Frameworks. Its purpose is to handle a large number of network predictors, according to the needs of the network and its slices. The presented architecture is flexible for horizontal addition of metric predictors and fault-reliable, presenting a simple and clean interface for all the predictors.

According to the proposed architecture, the relevant characteristics for the predictor architecture are:

[12] `https://www.java.com/`
[13] `https://kafka.apache.org/documentation/streams/`
[14] `https://www.elastic.co/elk-stack`
[15] `https://docker.com`
[16] `https://www.pgadmin.org/`
[17] `https://grafana.com/`

- simple and flexible API to be used for clients that do not need to understand the internals of the predictor;
- easy addition and removal of predictors, without downtime, according to the needs of the clients;
- each metric predictor can have online or offline training;
- each metric predictor can store the previous metrics to train the prediction model;
- creation of ensemble predictions by taking into account the predictions made by multiple models;
- distributed prediction and training - it is possible to distribute the predictors in different systems, or even to distribute the training phase and the prediction phase for the same predictor, to allow for fast prediction and fine tuning of the model;
- horizontal scaling - the predictor API and the message broker can scale to handle the growing number of predictors;
- the training of a predictor can be requested in two different ways: with a training schedule defined by the client or by direct client request.

The predictors done in Chapters (4, 5 and 6) are integrated in the prediction architecture and can be used in a real scenario.

In the next chapter, two scenarios will be implemented and tested where the network operator can improve the network behavior by forecasting the network metrics and adapting the network resources.

9

5G Network Slicing with Network Metrics Forecasts

With the predictors from both networks (the vehicular network, in Chapter 4, and the 4G network, in Chapter 6) implemented in the prediction architecture (Chapter 8), it is needed to understand how to act automatically to improve the quality of service for the users on each network. In the 4G network dataset, each connected user will be counted as a network session, to be able to use the same metric in both datasets. Each network will be treated as a slice from a bigger network. In this chapter, two scenarios will be tested.

In the first scenario, the network has limited resources, and it is needed to divide those resources for multiple slices as optimally as possible, maximizing the network utilization and minimizing the number of degraded network sessions. The second scenario has the aim of preventing network congestion while minimizing the use of resources for each slice, managing dynamically the resources needed.

9.1 Dividing a resource for multiple slices

For this scenario, the network resource chosen is the maximum number of sessions per hour in the network, which will be limited to a maximum value. The maximum number of allowed sessions will have to be split across all slices to reduce the number of lost sessions in the network. That split can be made statically (setting a fixed threshold for each slice) or dynamically (adjusting in real-time the threshold for each slice, according to the needs).

In this section, three splitting algorithms will be compared: a fixed-threshold algorithm, where the splitting of the maximum number of sessions between the slices is done with a fixed threshold for all slices, defined by an optimization search on the past values of that resource for that slice; an optimal dynamic threshold algorithm, that takes its decisions of the splitting based on the future values of the metrics of all slices (as if the forecasts were always exactly correct); and a best-possible dynamic threshold algorithm, that makes the threshold decisions based on the forecasts made for the next hour of the slices.

Furthermore, the maximum number of sessions available for the network in the tests will also vary from 0 to the maximum number of sessions needed for the network to serve all users (when the number of sessions lost is 0).

Some assumptions about the threshold must be taken. For n slices, the sum of all thresholds at time t must be equal to the maximum number of sessions in the network at time t (Equation 9.1).

$$max\_number\_sessions[t] = threshold_{slice_1}[t] + threshold_{slice_2}[t] + ... + threshold_{slice_n}[t] \quad (9.1)$$

The performance metric that will be minimized for this problem is the number of sessions above the threshold for that slice, as represented in Equation 9.2 using Iverson bracket notation, where ns is the number of sessions of a slice.

$$L(ns, threshold, t) = \sum_{t=0}^{length(ns)} ns[t] - threshold[t][ns[t] > threshold[t]] \tag{9.2}$$

9.1.1 Fixed-threshold algorithm

The fixed threshold algorithm is the simplest algorithm to limit the maximum number of sessions per slice. The threshold imposed on all slices is static and does not change over time with the different number of sessions in each slice. The challenge is to find what is the best static division of resources, to assure that the minimum number of sessions is above the threshold for each slice.

To find the best division of resources, an optimization search will be done in the dataset with the past values of the resource for that slice. In both datasets used, the train and cross-validation sets (Figure 4.21 for the vehicular network and Figure 6.18 for the 4G network) will be used as historic values, and one week of the test set will be used for testing purposes. For a specific number of maximum sessions in the network, various thresholds will be tested. For each threshold, the performance metric of each slice will be evaluated. In the end, it is possible to find a minimum value where the aggregated number of lost sessions in the network for all slices is minimized. To assure that a minimum of quality of service is delivered to users in all slices, it is also possible to set the minimum number of sessions per slice, that prevents a threshold for a slice of being lower than that value. As an assumption for the algorithm, it is needed that the sum of all the minimum number of sessions for all slices is lower than the maximum number of sessions available in the network.

In Figure 9.1 it is depicted the percentage of lost sessions for different values of the 4G slice threshold, for a maximum of 3000 sessions allowed in the network. By Equation 9.1, the vehicular slice threshold has $3000 - threshold_{4Gslice}$ allowed sessions. When the maximum number of allowed sessions in the 4G slice increases, the percentage of lost sessions in that slice decreases, while the percentage of lost sessions in the vehicular slice increases. For a maximum of 3000 sessions allowed in

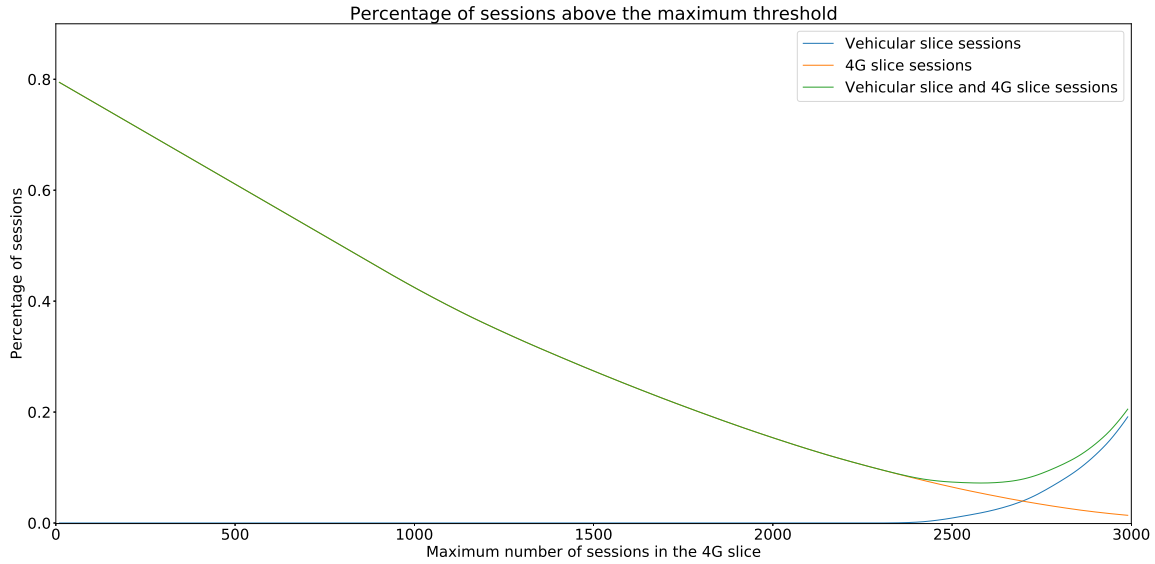

Figure 9.1: Percentage of lost sessions for both slices, varying according to the maximum number of sessions in the 4G slice, for a maximum of 3000 sessions in the network.

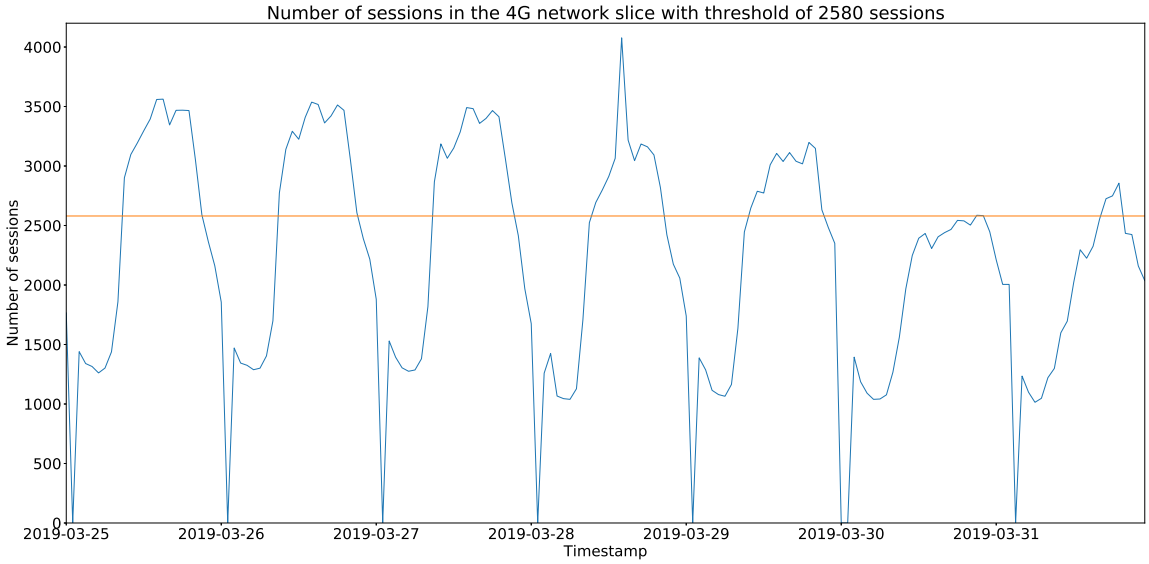

Figure 9.2: Number of sessions in a week of the test set of the 4G slice, with the threshold of 2580 sessions.

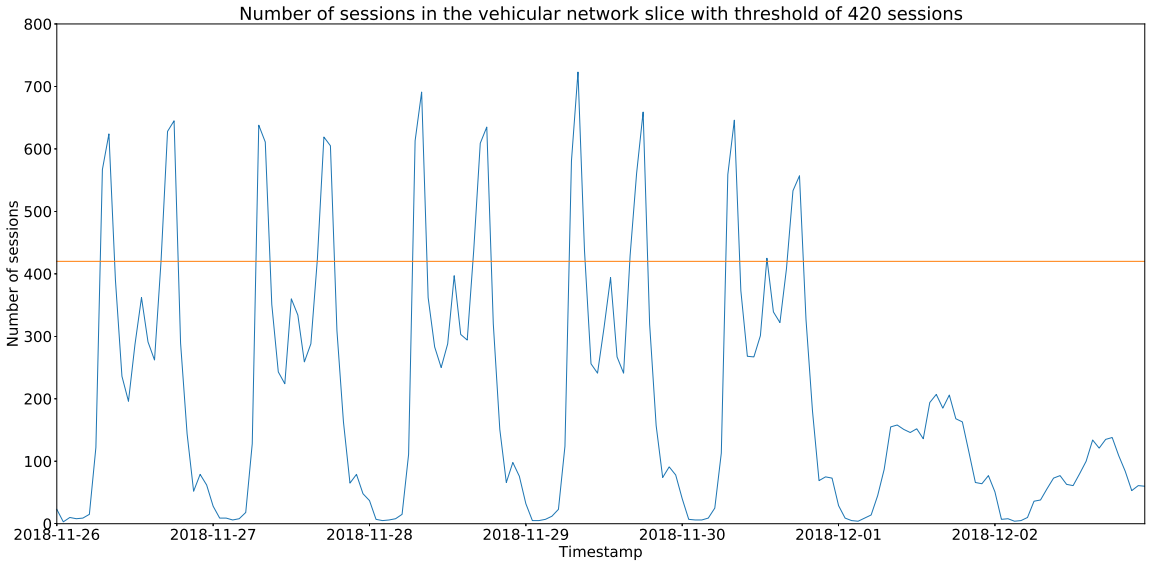

Figure 9.3: Number of sessions in a week of the test set of the vehicular slice, with the threshold of 420 sessions.

the network, the best trade-off for the threshold for both slices is when $threshold_{4Gslice} = 2580$ and $threshold_{vehicularslice} = 420$, as depicted in Figure 9.1 in the minimum value of the vehicular slice and 4G slice sessions. Figures 9.2 and 9.3 show those threshold values in one week of the test set from both slices.

This algorithm has the advantage of not being needed to adapt the network resources throughout the network operation, reducing the overhead in the network. The periodic threshold calculation is also avoided because the threshold is always the same. If the network resources are static and it is not possible to move resources between slices, this algorithm is the best approach for dividing a resource for multiple slices.

However, there are some disadvantages to the use of this algorithm. It works reasonably well with

stationary data, but it will produce large errors in non-stationary data. That happens because the threshold is defined by the past values; if the past and future values are much different, the thresholds chosen will result in a higher number of lost sessions. As a corollary, another disadvantage is that in the long-term the changes in the network traffic will not be accommodated by the threshold, due to the threshold being always fixed. The biggest drawback is the inability of the threshold to take advantage of the fluctuations in network traffic to adapt the network resources conveniently.

9.1.2 Dynamic threshold algorithm

The dynamic threshold algorithm takes a different approach to decide the threshold for different slices. Instead of a fixed threshold, its value will change according to the forecasts of the next hour, to better adjust to the network traffic fluctuations, for better utilization of the maximum number of available sessions in the network.

Just like in the fixed-threshold algorithm, it is possible to set the minimum number of sessions per slice, which prevents a forecast of a slice with a low number of sessions from blocking the slice for accepting new sessions.

It will be presented the algorithm steps for managing the number of sessions in two slices. For each time step, the algorithm has to perform the following calculations, also described in Algorithm 1:

1. fetch the forecasts of both slices;
2. if both forecasts are lower than the minimum number of sessions for their slice, divide equally the available number of sessions;
3. if one slice forecast is lower than the minimum number of sessions for their slice, set the available number of sessions for that slice equal to the minimum number of sessions for that slice, and all the remaining sessions for the other slice;
4. if both slice forecasts are above the minimum number of sessions for that slice,
   a) if the sum of the forecasts is lower or equal than the available number of sessions (ns) (there is enough available number of sessions for both slices), each slice will get the number of predicted sessions (ps) added to the number of available sessions (as) weighted for each slice, as described in Equation 9.3;

$$ns_{slice(x)} = ps_{slice(x)} + as * \frac{ps_{slice(x)}}{ps_{slice(x)} + ps_{slice(x\%2)+1}} \tag{9.3}$$

   b) if the sum of the forecasts is higher than the available number of sessions (ns) (there is not enough available number of sessions for both slices), each slice will get the number of predicted sessions (ps) minus the number of excedent sessions (es) weighted for each slice, as described in Equation 9.4;

$$ns_{slice(x)} = ps_{slice(x)} - es * \frac{ps_{slice(x)}}{ps_{slice(x)} + ps_{slice(x\%2)+1}} \tag{9.4}$$

   c) if any of the numbers of sessions calculated in the previous steps are below the minimum number of sessions for that slice, adjust that slice to have the minimum number of sessions allowed for that slice, and the other slice to have all the remaining sessions.

Compared with the fixed-threshold algorithm, this algorithm has the advantage of being able to adjust to the local fluctuation of the number of sessions, being able to optimize the use of the network in each time frame. It is also adaptable to the network traffic over time, since the forecasts can be adjusted according to the past behavior of the network. The biggest drawbacks of the algorithm are the overhead in the network of changing policies regularly and the periodic threshold calculation.

Algorithm 1 Dynamic Threshold Algorithm

procedure DYNAMIC_THRESHOLD_ALGORITHM($predictions\_slice\_1$, $predictions\_slice\_2$, $min\_sessions\_slice\_1$, $min\_sessions\_slice\_2$, $max\_sessions\_available$)
  $sessions_{p1} = []$
  $sessions_{p2} = []$
  for each $p1$ in $predictions\_slice\_1$ and each $p2$ in $predictions\_slice\_2$ **do**
    if $p1 \leq min\_sessions\_slice\_1$ and $p2 \leq min\_sessions\_slice\_1$ **then**
      $sessions_{p1} += [\frac{max\_sessions\_available}{2}]$
      $sessions_{p2} += [\frac{max\_sessions\_available}{2}]$
    else if $p1 \leq min\_sessions\_slice\_1$ **then**
      $sessions_{p1} += [min\_sessions\_slice\_1]$
      $sessions_{p2} += [max\_sessions\_available - min\_sessions\_slice\_1]$
    else if $p2 \leq min\_sessions\_slice\_2$ **then**
      $sessions_{p1} += [max\_sessions\_available - min\_sessions\_slice\_2]$
      $sessions_{p2} += [min\_sessions\_slice\_2]$
    else
      if $p1 + p2 \leq max\_sessions\_available$ **then**
        $temp\_sessions\_1 = round(p1 + ((max\_sessions\_available - (p1 + p2)) * \frac{p1}{p1+p2}))$
        $temp\_sessions\_2 = max\_sessions\_available - temp\_sessions\_1$
      else
        $temp\_sessions\_1 = round(p1 - (((p1 + p2) - max\_sessions\_available) * \frac{p1}{p1+p2}))$
        $temp\_sessions\_2 = max\_sessions\_available - temp\_sessions\_1$
      end if
      if $temp\_sessions\_1 < min\_sessions\_slice\_1$ **then**
        $temp\_sessions\_1 = min\_sessions\_slice\_1$
        $temp\_sessions\_2 = max\_sessions\_available - temp\_sessions\_1$
      else if $temp\_sessions\_2 < min\_sessions\_slice\_2$ **then**
        $temp\_sessions\_2 = min\_sessions\_slice\_2$
        $temp\_sessions\_1 = max\_sessions\_available - temp\_sessions\_2$
      end if
      $sessions_{p1} += [temp\_sessions\_1]$
      $sessions_{p2} += [temp\_sessions\_2]$
    end if
  end for
  Return $sessions_{p1}, sessions_{p2}$
end procedure

9.1.3 Evaluation of the algorithms

Three policy threshold models will be evaluated:

- the fixed threshold algorithm;
- the dynamic threshold algorithm, where the forecasts are the real values (in reality, this model would be impossible, because the algorithm is forecasting the future values with no error; however, it establishes a baseline for the best possible result with the dynamic threshold algorithm);
- the dynamic threshold algorithm with the best forecasts for each slice (for the vehicular slice, the best forecast is presented in Subsection 4.4.9 and its results are in Figure 4.41, achieved with an ensemble of a Feed-forward Neural Network model, an LSTM network model and a Random Forest model; for the 4G slice, the best forecast is presented in Subsection 6.4.7 and its results are in Figure 6.27, achieved with an ensemble of a Feed-forward Neural Network model and an LSTM network model).

The evaluation will be made by running the three algorithms with a different number of sessions available in the network (from no session available to all the sessions needed available). It will be

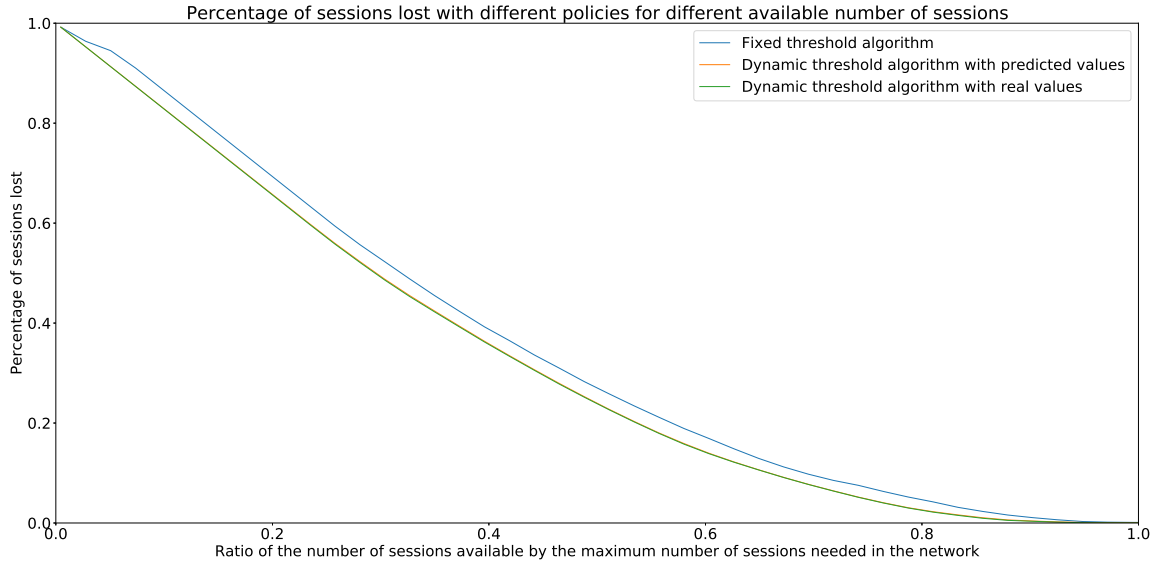

Figure 9.4: Percentage of sessions lost varying the sessions available in the network for different algorithms.

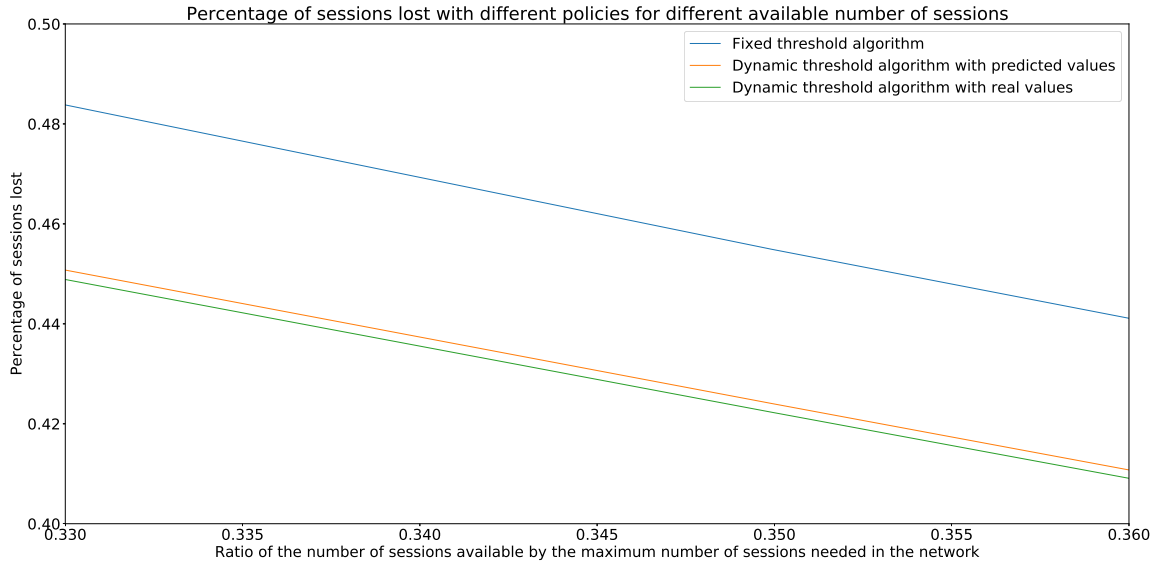

Figure 9.5: Percentage of sessions lost varying the sessions available in the network for different algorithms, in detail.

evaluated the total percentage of sessions in the network above the number of sessions allowed for each slice.

Figure 9.4 shows the results. The percentage of lost sessions decreases as the number of sessions available in the network increases for all algorithms, as expected. The fixed threshold algorithm performs worse than the dynamic threshold one, while the dynamic threshold algorithm performs only marginally worse with the forecasts than with the real values. Figure 9.5 shows the differences between both algorithms in detail, imperceptible in Figure 9.4.

This result shows the effectiveness of applying the dynamic threshold algorithm with forecasts for a 5G network. It is possible to reduce the number of lost sessions per slice in a congested network at around 5% when compared with the fixed-threshold policy, and also avoiding the starvation of the

slice resources with a minimum value for the number of sessions per slice.

9.2 Preventing network congestion

In the previous section, it was assumed that the network had limited resources and that it had to perform resource division for multiple slices. While that can be important for small networks and networks for local events such as music festivals or football games, or for networks of small ISPs, in large core networks the resources are overdimensioned, and such case of resource exhaustion is not likely to happen.

However, a scenario that is more likely to happen is network congestion. With theoretically infinite resources, the goal of the management of large networks is not to assure that each slice has enough resources to serve all users, but to give just enough resources for each slice to provide to its users the minimum congestion possible, while saving the most possible resources for other slices or for network emergencies. This is the purpose of this section.

Just like in the previous section, the resource chosen is the maximum number of sessions per slice. Because the available metrics do not allow to derive network congestion, it will be assumed that the network is congested when the number of sessions per hour is higher than 80% of the maximum value of the number of sessions in the dataset.

The test set used is the same week of the data from both slices that it was used for the previous section. Three evaluation methods will be proposed to calculate the congestion in the network.

The first method is when there are no policies that prevent congestion in the network. All the sessions above the threshold of 80% of the maximum number of sessions in the dataset are counted as congested sessions.

The second method is when there is a reactive action in the slice upon congestion. If the number of sessions in the slice is above the threshold, the resources in the network are adapted in the next time-frame to accommodate the value of the number of sessions in the previous time-frame. Figures 9.6, 9.7 and 9.8 show the threshold adaptation in the vehicular slice, in the 4G slice and in the entire network, respectively.

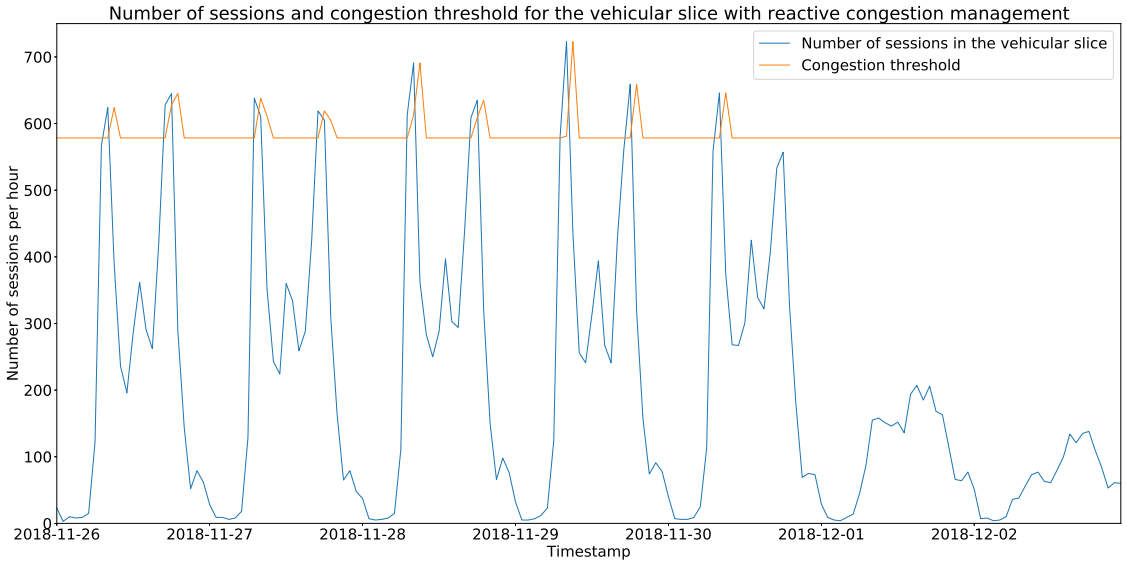

Figure 9.6: Reactive management of the congestion threshold in the vehicular slice.

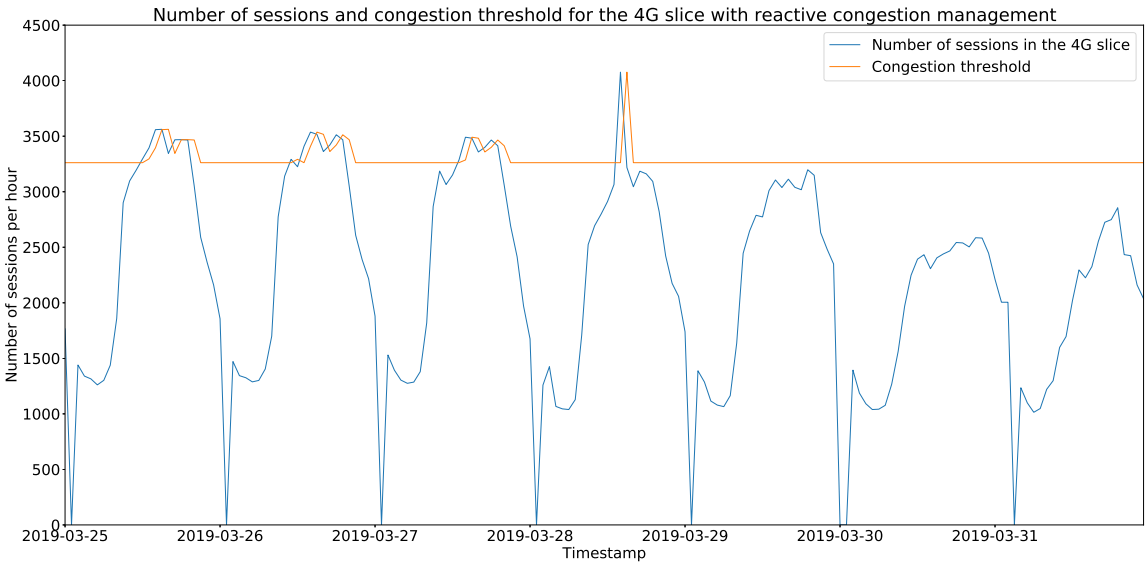

Figure 9.7: Reactive management of the congestion threshold in the 4G slice.

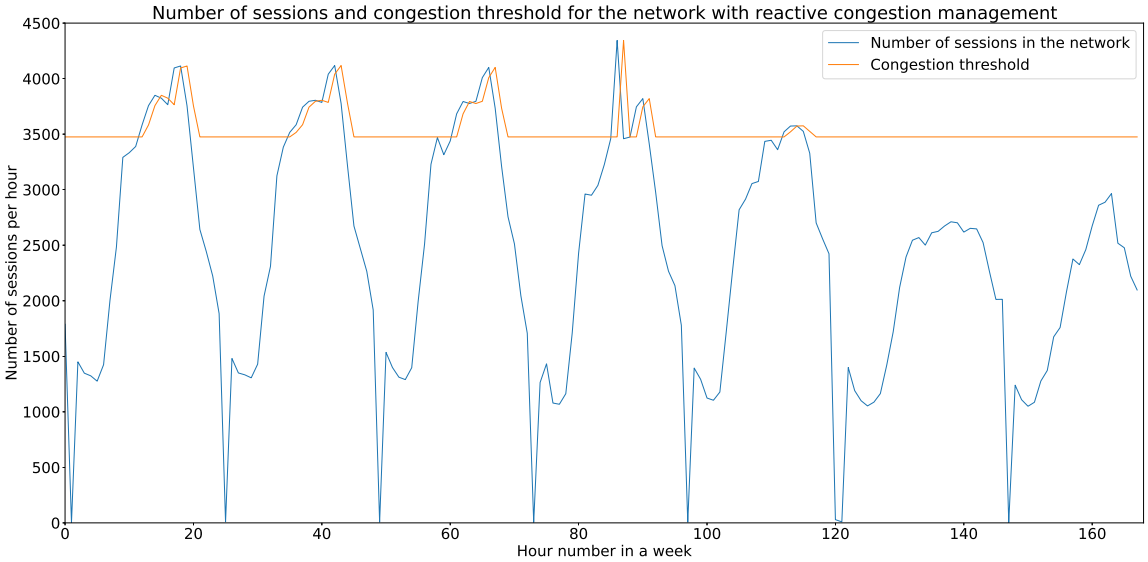

Figure 9.8: Reactive management of the congestion threshold in the network.

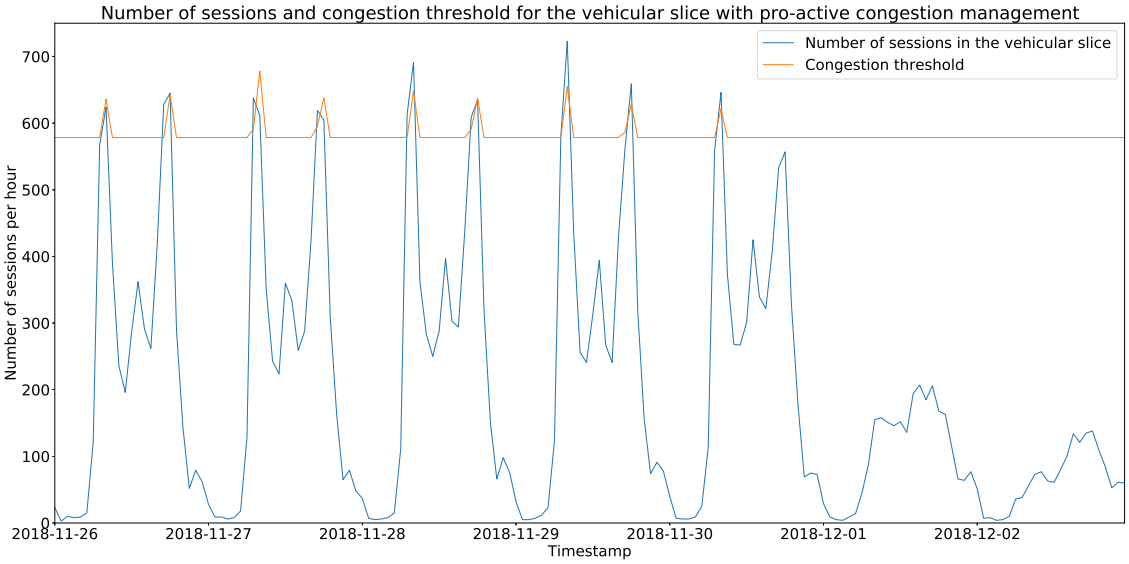

Figure 9.9: Pro-active management of the congestion threshold in the vehicular slice.

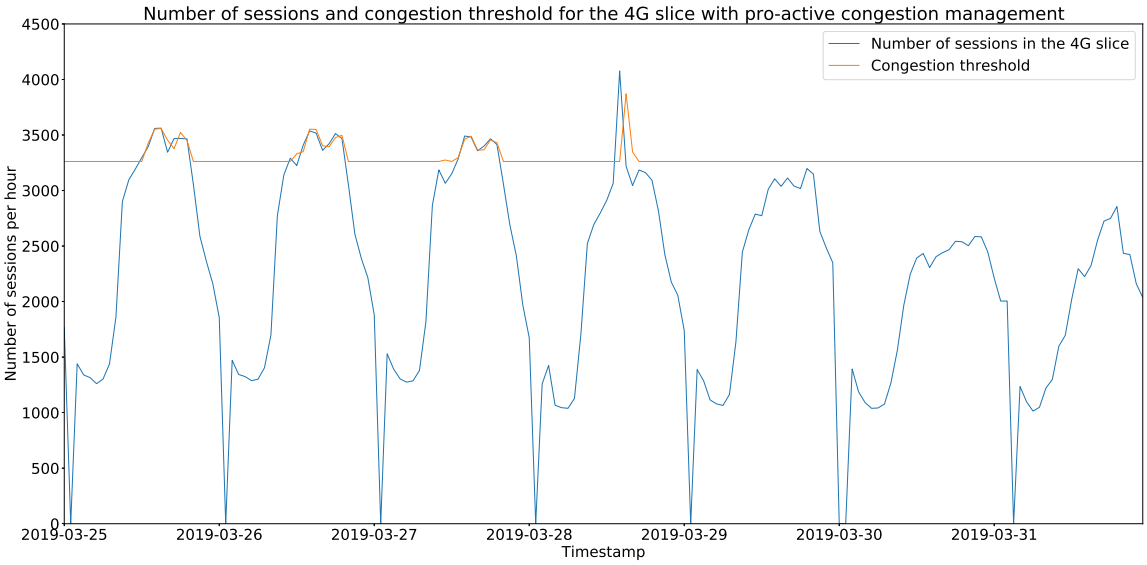

Figure 9.10: Pro-active management of the congestion threshold in the 4G slice.

Finally, the third method uses the forecasts from both slices (the same forecasts that were used in the previous section) to perform pro-active management of the congestion. When the number of sessions predicted in the next time-frame for each slice is above the 80% threshold, the network prematurely adapts its resources for each slice to accommodate the number of sessions predicted, to prevent the congestion in the network. Figures 9.9, 9.10 and 9.11 show the pro-active threshold adaptation in the vehicular slice, in the 4G slice and in the entire network, respectively.

The performance metric for the three methods is the number of sessions above the threshold, which is shown in Table 9.1. The reactive management of the congested threshold reduces the number of congested sessions for both slices, and the pro-active management of the congested threshold reduces it even more, as expected.

An interesting fact is the high number of congested sessions in the entire network when compared

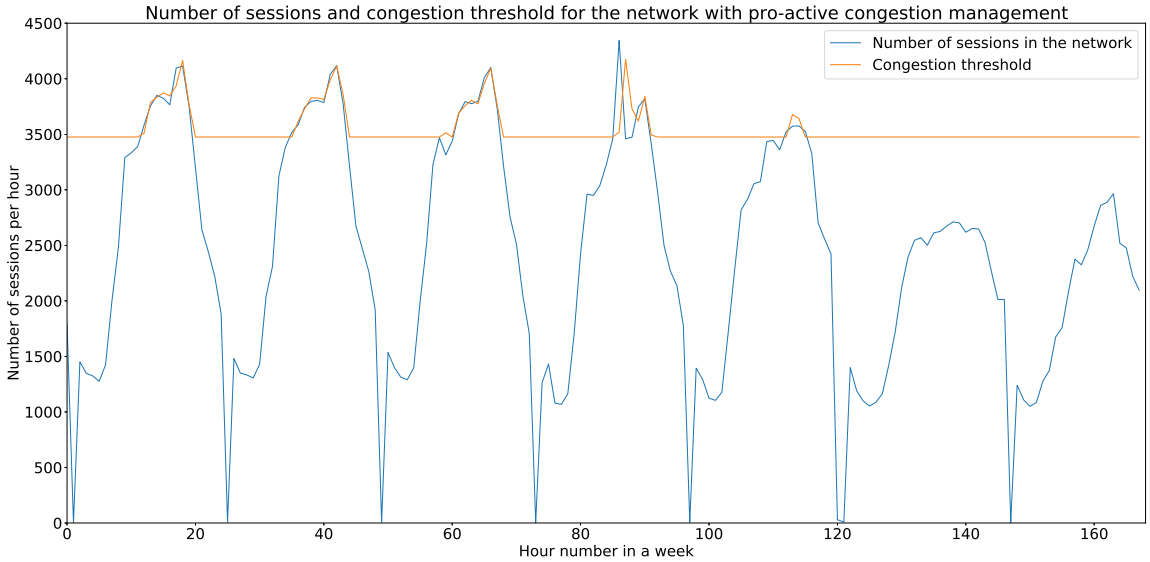

Figure 9.11: Pro-active management of the congestion threshold in the network.

	Vehicular slice	**4G slice**	**Entire Network (aggregated)**
No threshold adaptation	851	4774	10175
Reactive management	674	2033	3342
Pro-active management	343	1188	1522

Table 9.1: Number of congested sessions in both slices, according to the different methods of congestion management used.

with both classes, for the two first methods. That happens due to the aggregated traffic having its mean value closer to the 80% threshold, with more sessions being above the threshold than before, and with lower peaks when compared with the mean value of the data. However, the pro-active management of the network can adapt to it, because the aggregated congestion management has less congested sessions than the sum of the congested sessions in both slices.

These results show that the reactive management of the network with threshold adaptation reduces the congestion in the network. More than that, the results also show that pro-active management reduces even more the congestion in the network, through the utilization of accurate predictions in the network. Other than congestion reducing, the pro-active management in the network also reduces the resources needed for each slice, by predicting accurately what are the resources needed for all slices, allowing for a better resource allocation.

9.3 Conclusion

In this chapter, two scenarios were tested using the forecasts (described in Chapters 4 and 6) and the prediction architecture (described in Chapter 8).

In the first scenario, the goal was to divide a network resource for multiple slices. Various approaches were tested and compared, and it was shown the effectiveness of applying the dynamic threshold algorithm with forecasts, reducing the number of lost sessions per slice at around 5%, when compared with a fixed-threshold policy.

In the second scenario, the objective was to prevent network congestion in multiple slices by giving more resources to the slice when needed. The results showed that it is possible to reduce the congestion

in slices by applying pro-active management of the network with the usage of network predictions.

As a conclusion, in a 5G network, with the use of forecasts made in the prediction architecture, it is possible to improve the network behavior with pro-active management.

10

Conclusion & Future Work

10.1 Conclusions

With the implementation of 5G networks, automated decisions and autonomous action on the network are essential for the management of large 5G operator networks. This dissertation aimed to outline and develop a management system for 5G networks that focused on the forecast of network metrics. The state of the art for management systems for 5G networks was studied in this dissertation as a way to better understand the current trends of management systems and to understand how to integrate an external module in those systems.

Forecasting network metrics

Two networks were used to forecast the number of sessions per hour: a vehicular network of Porto public transports and a national 4G network. Both datasets had a daily and weekly pattern, and they did not presented any long-term increasing or decreasing trend. In the vehicular network, it was possible to understand the working schedule of Porto inhabitants by analyzing the number of sessions per hour in the public transports: the peaks in the number of sessions occur from 6:00 to 8:00 and from 18:00 to 20:00. Although Porto is a city with an increasing number of tourists every year, the majority of the people that use the public transports are Porto inhabitants that follow a working schedule. During the weekends, Porto inhabitants use fewer public transportations than on weekdays.

Several forecasting techniques were applied to both datasets: the persistence algorithm was used to create a baseline result; the statistical algorithm ARIMA was also used, as well as classical machine learning algorithms and deep learning algorithms (feed-forward neural networks, LSTM networks and convolutional neural networks). The tests made varied not only the algorithm, but also the hyper-parameters of the algorithms and the pre-processing parameters applied to the data, such as the number of input lags considered for the machine learning algorithms, the ARIMA order parameters, the differentiation lag applied to the data or the network configuration.

The results showed that the best forecasting models are from the feed-forward and LSTM networks. Furthermore, ensemble algorithms containing both forecasts improve the result. For both networks, the average best number of input lags is 12 or 24, indicating that for both datasets, the forecasting algorithms only need information about the previous 12 or 24 hours to provide an accurate forecast, and the prediction of the number of sessions in the next hour does not depend much on values previous to 24 hours. The ARIMA algorithm was more dependent on the data being stationary, achieving high

error when the data was not differenced. For machine learning algorithms, no differentiation or just one-lag differentiation provided the best results for most cases.

In the machine learning algorithms, three additional features were included to make more accurate forecasts: the day of the week, the hour of the day and the workday (a flag that indicates if it is workday or weekend/holiday). In the tests, the workday was the feature that most improved the results. However, no conclusion could be made about which is the best additional feature: depending on the tests, the results varied. For most tests, the inclusion of additional features improved the forecasts.

Instead of using the previous values as input, it was tested to use statistical features extracted from the time-series to forecast the number of sessions. However, the result was not as good as with the algorithms that used the previous values. One reason for that is the high autocorrelation with the previous values in the data. If the number of sessions did not present any seasonal pattern, the usage of statistical features extracted from the time-series was more likely to improve the results.

Using the vehicular network, two other forecasting scenarios were tested. The first scenario was to forecast accurately higher value observations. It was created new performance metrics and a custom objective function to adapt the prediction goal. While the objective function did not improve the results, the new performance metrics were used to measure the error of the forecasts and obtain the best result throughout the number of trained epochs in the neural networks, improving the forecasts for higher observation values. The second scenario was to forecast accurately on anomalous time periods. The results showed that the best approach was to normalize the input and output data to a similar distribution. In this case, the best results were achieved when it was applied one-lag differentiation to the data.

Real-time Distributed Time-series Prediction Architecture

A real-time distributed time-series prediction architecture was designed and implemented, with the goal of simplifying the prediction of network metrics and the integration in 5G management systems, using a REST API. The architecture also has also other advantages, such as increased modularity, the capabilicity of having multiple predictors of different network metrics to predict independently, online/offline learning and horizontal scaling. This architecture enables the predictors to automatically improve their predictions with time, due to the automatic training with the most recent values, that can be scheduled by the network operator. In this way, the predictors in the architecture are robust to network changes over time. Since the architecture is able to integrate various types of predictors, it can be applied not only to 5G systems, but also to other systems that deal with time-series prediction in real-time.

Root Cause Analysis of Low Network Accessibility

In this work, it was also proposed to use the 4G network data to understand the causes of low accessibility. It was chosen to use a machine learning approach that measures the importance of each feature with an algorithm internal calculation, as opposed to measuring the test error of a model when the features vary. The analysis was made with two different goals: to understand causes of low accessibility in the network and to understand the causes of low accessibility by cell.

The results were different for both tests. For the overall network accessibility, the most important KPIs were the number of handover failures, the CSFB preparation success, the download PDCP data volume in the last transmission time and the cell available time. These KPI are the ones that most influence the low network accessibility in the next hour.

For the analysis by cell, forecasting low network accessibility is harder than for the overall network, with the error achieved being bigger (for the network accessibility model, it was achieved an F1-score

of 86.6%; for the accessibility by cell model, it was achieved an F1-score of 30.79%). The two most important KPIs were the RRC and the E-RAB establishment success ratio. Those KPIs are themselves indicators of low network accessibility, and they are not understood as the cause for network metrics, but as the consequence. These results show that the network accessibility per cell is highly dependent on the network accessibility of that cell in the previous hour. Other important KPIs to the forecast of low network accessibility per cell are the counters related to the number of users and its network utilization, such as the maximum number of users, the average number of connected users or the average number of active users downloading data.

For the network operator, it is essential to understand the causes of low network accessibility, not only to monitor correctly the network, but also to know where to act in the network to increase the network accessibility.

5G network slicing with Network Metrics Forecasts

Finally, using the forecasting models achieved previously, two 5G scenarios were proposed to improve the network slicing. The first scenario was to improve the division of a limited network resource for multiple slices. The proposed algorithm, the dynamic threshold algorithm with the predictions, takes advantage of the predictions for the next hour to adapt the network resources, to ensure that the resources needed for each slice are divided fairly according to the predictions of all slices. In the tests with the number of sessions, this algorithm proves itself to reduce the number of lost sessions in 5%, when compared with the fixed-threshold algorithm.

The second scenario proposed was to prevent network congestion for different slices. In this scenario, the goal was not to split resources for different slices, but to optimize the usage of a resource for slices: each slice would have just enough resources to provide to its users the minimum congestion possible, while saving the most possible resources for other slices or for network emergencies. It was tested three approaches: no congestion management (static resources for all slices), reactive congestion management (the resources for each slice were adapted after a congestion event) and pro-active congestion management (using the forecasts achieved previously, adapt the network resources according to the forecasts of the next hour). With pro-active management of the congestion, it was possible to forecast the congestion and act accordingly in the network to increase the network resources for a slice to avoid congestion.

Using these two scenarios, it was demonstrated that the prediction of network metrics, besides helping the network administrator to understand the future trends of the network, can also be used to perform autonomic improvements in the network.

10.2 Future Work

As future work, there are some elements that can be developed:

- the integration of the prediction architecture in a 5G management system is the next step to prove its performance, flexibility and easy integration;
- the creation of a testbed to use of the prediction architecture in a 5G network in real-time, with online prediction and training, is crucial to demonstrate its benefits in a long-term real-world scenario;
- in the prediction architecture internals, a new module for the creation of out-of-the-box metric predictors can be implemented, to providing a quick start for the network administrators that want to predict a network metric without the need for any programming or machine learning

knowledge. That module could provide some common configurations as options for building the metric predictor;

- a more in-depth analysis can be done about the hyper-parameters used in the forecasting algorithms, with tests with different time-series distributions with various forecasting algorithms;
- the automation of the dataset exploration and forecasting algorithms application to achieve the best model for forecasting network metrics is possible. There is active research in the machine learning automation field (AutoML), and with the correct implementation of dataset exploration, valuable insights about the data can ease the machine learning automation phase, by reducing the tests needed to run;
- a more in-depth analysis can be made to the KPIs that cause low network accessibility in a 4G network. It is important not only to understand what KPIs are the most important for forecasting low accessibility, but also to detect the specific patterns in those KPIs that indicate future low accessibility, to be able to adapt the network resources to prevent it to happen;
- and finally, close the loop with the implementation of changes in the network configuration as a reaction to the prediction approaches, and understand the impact of these changes in the network performance.

Bibliography

[1] T. Saydam and T. Magedanz, "From networks and network management into service and service management", *Journal of Network and Systems Management*, vol. 4, no. 4, pp. 345–348, Dec. 1996. DOI: 10.1007/bf02283158. [Online]. Available: https://doi.org/10.1007/bf02283158.

[2] R. Boutada and X. Jin, *Telecommunication Systems and Technologies - Volume II, Encyclopedia of Life Support Systems*. Eolss Publishers Co. Ltd., 2009, ch. 1.

[3] David Kennedy, "D2.6 Final report on programme progress and KPIs", The 5G Infrastructure Public Private Partnership (5G-PPP), 2017. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2017/10/Euro-5G-D2.6_Final-report-on-programme-progress-and-KPIs.pdf.

[4] A.-M. Bosneag and M. X. Wang, "Intelligent network management mechanisms as a step towards SG", in *2017 8th International Conference on the Network of the Future (NOF)*, IEEE, Nov. 2017. DOI: 10.1109/nof.2017.8251220. [Online]. Available: https://doi.org/10.1109/nof.2017.8251220.

[5] A. Imran and A. Zoha, "Challenges in 5g: How to empower SON with big data for enabling 5g", *IEEE Network*, vol. 28, no. 6, pp. 27–33, Nov. 2014. DOI: 10.1109/mnet.2014.6963801. [Online]. Available: https://doi.org/10.1109/mnet.2014.6963801.

[6] L. Andersson, H. van Helvoort, R. Bonica, D. Romascanu, and S. Mansfield, *Guidelines for the Use of the "OAM" Acronym in the IETF*, RFC 6291 (Best Current Practice), Internet Engineering Task Force, Jun. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6291.txt.

[7] "ITU-T Recommendation X.701 (08/97) - Information technology – Open Systems Interconnection – Systems management overview", ITU-T - Telecommunication Standardization Sector of ITU, Recommendation, 1997.

[8] Cisco, "Introduction to etom - white paper", Tech. Rep., 2009. [Online]. Available: https://www.cisco.com/c/en/us/products/collateral/services/high-availability/white_paper_c11-541448.pdf (visited on 09/23/2018).

[9] "ISO/IEC 20000-1 - Information technology – Service management", International Organization for Standardization, Standard, 2005.

[10] J. Davin, J. Case, M. Fedor, and M. Schoffstall, *Simple Gateway Monitoring Protocol*, RFC 1028 (Historic), Internet Engineering Task Force, Nov. 1987. [Online]. Available: http://www.ietf.org/rfc/rfc1028.txt.

[11] J. Case, M. Fedor, M. Schoffstall, and J. Davin, *Simple Network Management Protocol (SNMP)*, RFC 1157 (Historic), Internet Engineering Task Force, May 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1157.txt.

[12] U. Warrier, L. Besaw, L. LaBarre, and B. Handspicker, *Common Management Information Services and Protocols for the Internet (CMOT and CMIP)*, RFC 1189 (Historic), Internet Engineering Task Force, Oct. 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1189.txt.

[13] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry, *The COPS (Common Open Policy Service) Protocol*, RFC 2748 (Proposed Standard), Updated by RFC 4261, Internet Engineering Task Force, Jan. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2748.txt.

[14] S. Waldbusser, R. Cole, C. Kalbfleisch, and D. Romascanu, *Introduction to the Remote Monitoring (RMON) Family of MIB Modules*, RFC 3577 (Informational), Internet Engineering Task Force, Aug. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3577.txt.

[15] B. Claise, *Cisco Systems NetFlow Services Export Version 9*, RFC 3954 (Informational), Internet Engineering Task Force, Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt.

[16] B. Claise, B. Trammell, and P. Aitken, *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*, RFC 7011 (INTERNET STANDARD), Internet Engineering Task Force, Sep. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc7011.txt.

[17] B. Claise, A. Johnson, and J. Quittek, *Packet Sampling (PSAMP) Protocol Specifications*, RFC 5476 (Proposed Standard), Internet Engineering Task Force, Mar. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5476.txt.

[18] 3GPP, "View on 5G Architecture - 5G Architecture White Paper", 5GPPP Architecture Working Group (5GPPP), 2018. [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2018/01/5G-PPP-5G-Architecture-White-Paper-Jan-2018-v2.0.pdf.

[19] ——, "Study on New Services and Markets Technology Enablers", 3rd Generation Partnership Project (3GPP), TR 22.891, 2015. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/22891.htm.

[20] 5. Americas, "5G Services & Use Cases", 5G Americas, 2017. [Online]. Available: http://www.5gamericas.org/files/3215/1190/8811/5G_Services_and_Use_Cases.pdf.

[21] F. Malandrino and C.-F. Chiasserini, "Present-day verticals and where to find them: A data-driven study on the transition to 5g", in *2018 14th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, IEEE, Feb. 2018. DOI: 10.23919/wons.2018.8311657. [Online]. Available: https://doi.org/10.23919/wons.2018.8311657.

[22] B. Naudts, M. Kind, F.-J. Westphal, S. Verbrugge, D. Colle, and M. Pickavet, "Techno-economic analysis of software defined networking as architecture for the virtualization of a mobile network", in *2012 European Workshop on Software Defined Networking*, IEEE, Oct. 2012. DOI: 10.1109/ewsdn.2012.27. [Online]. Available: https://doi.org/10.1109/ewsdn.2012.27.

[23] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture", *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 2, pp. 5–17, Apr. 1996. DOI: 10.1145/231699.231701. [Online]. Available: https://doi.org/10.1145/231699.231701.

[24] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, "A survey of active network research", *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, Jan. 1997. DOI: 10.1109/35.568214. [Online]. Available: https://doi.org/10.1109/35.568214.

[25] J. van der Merwe, S. Rooney, L. Leslie, and S. Crosby, "The tempest-a practical framework for network programmability", *IEEE Network*, vol. 12, no. 3, pp. 20–28, 1998. DOI: 10.1109/65.690958. [Online]. Available: https://doi.org/10.1109/65.690958.

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks", *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, Mar. 2008. DOI: 10.1145/1355734.1355746. [Online]. Available: https://doi.org/10.1145/1355734.1355746.

[27] O. N. Foundation, "OpenFlow Switch Specification", Open Networking Foundation (ONF), 2015. [Online]. Available: https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[28] ——, "Software-defined Networking: The New Norm for Networks - ONF White Paper", Open Networking Foundation (ONF), 2012. [Online]. Available: http://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[29] A. Gelberger, N. Yemini, and R. Giladi, "Performance analysis of software-defined networking (SDN)", in *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, IEEE, Aug. 2013. DOI: 10.1109/mascots.2013.58. [Online]. Available: https://doi.org/10.1109/mascots.2013.58.

[30] ETSI, "Network Functions Virtualisation – Introductory White Paper", European Telecommunications Standards Institute (ETSI), 2012. [Online]. Available: https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[31] S. Abdelwahab, B. Hamdaoui, M. Guizani, and T. Znati, "Network function virtualization in 5g", *IEEE Communications Magazine*, vol. 54, no. 4, pp. 84–91, Apr. 2016, ISSN: 0163-6804. DOI: 10.1109/MCOM.2016.7452271.

[32] ETSI, "Network Functions Virtualisation – Architectural Framework", European Telecommunications Standards Institute (ETSI), 2013. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf.

[33] J. Ordonez-Lucena, P. Ameigeiras, D. Lopez, J. J. Ramos-Munoz, J. Lorca, and J. Folgueira, "Network slicing for 5g with SDN/NFV: Concepts, architectures, and challenges", *IEEE Communications Magazine*, vol. 55,

no. 5, pp. 80–87, May 2017. DOI: 10.1109/mcom.2017.1600935. [Online]. Available: https://doi.org/10.1109/mcom.2017.1600935.

[34] 5. Americas, "Network Slicing for 5G Networks & Services", 5G Americas, 2016. [Online]. Available: http://www.5gamericas.org/files/3214/7975/0104/5G_Americas_Network_Slicing_11.21_Final.pdf.

[35] S. Mwanje, G. Decarreau, C. Mannweiler, M. Naseer-ul-Islam, and L. C. Schmelz, "Network management automation in 5g: Challenges and opportunities", in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, IEEE, Sep. 2016. DOI: 10.1109/pimrc.2016.7794614. [Online]. Available: https://doi.org/10.1109/pimrc.2016.7794614.

[36] W. Jiang, M. Strufe, and H. D. Schotten, "Intelligent network management for 5g systems: The SELFNET approach", in *2017 European Conference on Networks and Communications (EuCNC)*, IEEE, Jun. 2017. DOI: 10.1109/eucnc.2017.7980672. [Online]. Available: https://doi.org/10.1109/eucnc.2017.7980672.

[37] L. Fallon, J. Keeney, M. McFadden, J. Quilty, and S. van der Meer, "Using the COMPA autonomous architecture for mobile network security", in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, IEEE, May 2017. DOI: 10.23919/inm.2017.7987370. [Online]. Available: https://doi.org/10.23919/inm.2017.7987370.

[38] "Open Network Automation Platform (ONAP) Architecture White Paper", Open Network Autmation Platform (ONAP), 2018. [Online]. Available: https://www.onap.org/wp-content/uploads/sites/20/2018/06/ONAP_CaseSolution_Architecture_0618FNL.pdf.

[39] F. Slim, F. Guillemin, A. Gravey, and Y. Hadjadj-Aoul, "Towards a dynamic adaptive placement of virtual network functions under ONAP", in *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, IEEE, Nov. 2017. DOI: 10.1109/nfv-sdn.2017.8169880. [Online]. Available: https://doi.org/10.1109/nfv-sdn.2017.8169880.

[40] C. Mannweiler, M. Breitbach, H. Droste, I. L. Pavon, I. Ucar, P. Schneider, M. Doll, and J. R. Sanchez, "5g NORMA: System architecture for programmable & multi-tenant 5g mobile networks", in *2017 European Conference on Networks and Communications (EuCNC)*, IEEE, Jun. 2017. DOI: 10.1109/eucnc.2017.7980662. [Online]. Available: https://doi.org/10.1109/eucnc.2017.7980662.

[41] "5G Novel Radio Multiservice adaptive network Architecture (5G NORMA) - Deliverable D3.3 - Final report", 5GPPP Architecture Working Group (5GPPP), 2017. [Online]. Available: http://www.it.uc3m.es/wnl/5gnorma/pdf/5g_norma_d3-3.pdf.

[42] L. Xu, H. Assem, I. G. B. Yahia, T. S. Buda, A. Martin, D. Gallico, M. Biancani, A. Pastor, P. A. Aranda, M. Smirnov, D. Raz, O. Uryupina, A. Mozo, B. Ordozgoiti, M.-I. Corici, P. O'Sullivan, and R. Mullins, "CogNet: A network management architecture featuring cognitive capabilities", in *2016 European Conference on Networks and Communications (EuCNC)*, IEEE, Jun. 2016. DOI: 10.1109/eucnc.2016.7561056. [Online]. Available: https://doi.org/10.1109/eucnc.2016.7561056.

[43] M.-A. Kourtis, M. J. McGrath, G. Gardikis, G. Xilouris, V. Riccobene, P. Papadimitriou, E. Trouva, F. Liberati, M. Trubian, J. Batallx00E9, H. Koumaras, D. Dietrich, A. Ramos, J. F. Riera, J. Bonnet, A. Pietrabissa, A. Ceselli, and A. Petrini, "T-nova: An open-source mano stack for nfv infrastructures", *IEEE Transactions on Network and Service Management*, vol. 14, pp. 586–602, 2017.

[44] M. I. Sanchez, A. Asadi, M. Draxler, R. Gupta, V. Mancuso, A. Morelli, A. de la Oliva, and V. Sciancalepore, "Tackling the increased density of 5g networks: The CROWD approach", in *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, IEEE, May 2015. DOI: 10.1109/vtcspring.2015.7146122. [Online]. Available: https://doi.org/10.1109/vtcspring.2015.7146122.

[45] A. H. Celdrán, M. G. Pérez, F. J. G. Clemente, and G. M. Pérez, "Automatic monitoring management for 5g mobile networks", *Procedia Computer Science*, vol. 110, pp. 328–335, 2017, 14th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2017) / 12th International Conference on Future Networks and Communications (FNC 2017) / Affiliated Workshops, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.06.102. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050917312802.

[46] A. L. Samuel, "Some studies in machine learning using the game of checkers", *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, Jul. 1959, ISSN: 0018-8646. DOI: 10.1147/rd.33.0210.

[47] T. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997, ISBN: 978-0-07-042807-2.

[48] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Neurocomputing: Foundations of research", in, J. A. Anderson and E. Rosenfeld, Eds., Cambridge, MA, USA: MIT Press, 1988, ch. Learning Representations by Back-propagating Errors, pp. 696–699, ISBN: 0-262-01097-6. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104451.

[49] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[50] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994. DOI: 10.1109/72.279181. [Online]. Available: https://doi.org/10.1109/72.279181.

[51] F. Chollet, *Deep Learning with Python*, 1st. Greenwich, CT, USA: Manning Publications Co., 2017, ISBN: 1617294438, 9781617294433.

[52] J. N. K. Rao, G. E. P. Box, and G. M. Jenkins, "Time series analysis forecasting and control", *Econometrica*, vol. 40, no. 5, p. 970, Sep. 1972. DOI: 10.2307/1912100. [Online]. Available: https://doi.org/10.2307/1912100.

[53] J. Huang. (2005). Etom and itil: Should you be bi-lingual as an it outsourcing service provider, [Online]. Available: https://www.bptrends.com/publicationfiles/01-05%20eTOM%20and%20ITIL%20-%20Huang.pdf (visited on 10/05/2018).

[54] J. Moysen and L. Giupponi, "From 4g to 5g: Self-organized network management meets machine learning", *CoRR*, vol. abs/1707.09300, 2017. arXiv: 1707.09300. [Online]. Available: http://arxiv.org/abs/1707.09300.

[55] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (extended version)", *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, 1994. DOI: 10.1109/90.282603. [Online]. Available: https://doi.org/10.1109/90.282603.

[56] A. Azzouni and G. Pujolle, "Neutm: A neural network-based framework for traffic matrix prediction in SDN", *CoRR*, vol. abs/1710.06799, 2017. arXiv: 1710.06799. [Online]. Available: http://arxiv.org/abs/1710.06799.

[57] X. Ding, S. Canu, T. Denoeux, T. Rue, and F. Pernant, "Neural network based models for forecasting", in *in Proceedings of ADT'95*, Wiley and Sons, 1995, pp. 243–252.

[58] H. Feng and Y. Shu, "Study on network traffic prediction techniques", in *Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing, 2005.*, IEEE. DOI: 10.1109/wcnm.2005.1544219. [Online]. Available: https://doi.org/10.1109/wcnm.2005.1544219.

[59] C. W. J. Granger and R. Joyeux, "An introduction to long-memory time-series models and fractional differencing", *Journal of Time Series Analysis*, vol. 1, no. 1, pp. 15–29, 1980. DOI: 10.1111/j.1467-9892.1980.tb00297.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-9892.1980.tb00297.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9892.1980.tb00297.x.

[60] P. Cortez, M. Rio, M. Rocha, and P. Sousa, "Internet traffic forecasting using neural networks", in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, IEEE, 2006. DOI: 10.1109/ijcnn.2006.247142. [Online]. Available: https://doi.org/10.1109/ijcnn.2006.247142.

[61] M. Barabas, G. Boanea, A. B. Rus, V. Dobrota, and J. Domingo-Pascual, "Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition", in *2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*, IEEE, Aug. 2011. DOI: 10.1109/iccp.2011.6047849. [Online]. Available: https://doi.org/10.1109/iccp.2011.6047849.

[62] A. Haar, "Zur theorie der orthogonalen funktionensysteme", *Mathematische Annalen*, vol. 69, no. 3, pp. 331–371, Sep. 1910, ISSN: 1432-1807. DOI: 10.1007/BF01456326. [Online]. Available: https://doi.org/10.1007/BF01456326.

[63] H. Cui, Y. Yao, K. Zhang, F. Sun, and Y. Liu, "Network traffic prediction based on hadoop", in *2014 International Symposium on Wireless Personal Multimedia Communications (WPMC)*, IEEE, Sep. 2014. DOI: 10.1109/wpmc.2014.7014785. [Online]. Available: https://doi.org/10.1109/wpmc.2014.7014785.

[64] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks", GMD - German National Research Institute for Computer Science, GMD Report 148, 2001. [Online]. Available: http://www.faculty.jacobs-university.de/hjaeger/pubs/EchoStatesTechRep.pdf.

[65] G. Feng, "Network traffic prediction based on neural network", in *2015 International Conference on Intelligent Transportation, Big Data and Smart City*, IEEE, Dec. 2015. DOI: 10.1109/icitbs.2015.136. [Online]. Available: https://doi.org/10.1109/icitbs.2015.136.

[66] W. Banzhaf, *Genetic programming : an introduction on the automatic evolution of computer programs and its applications*. San Francisco, Calif. Heidelburg: Morgan Kaufmann Publishers Dpunkt-verlag, 1998, ISBN: 978-1558605107.

[67] Z. Chen, J. Wen, and Y. Geng, "Predicting future traffic using hidden markov models", in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, IEEE, Nov. 2016. DOI: 10.1109/icnp.2016.7785328. [Online]. Available: https://doi.org/10.1109/icnp.2016.7785328.

[68] R. Madan and P. S. Mangipudi, "Predicting computer network traffic: A time series forecasting approach using DWT, ARIMA and RNN", in *2018 Eleventh International Conference on Contemporary Computing (IC3)*, IEEE, Aug. 2018. DOI: 10.1109/ic3.2018.8530608. [Online]. Available: https://doi.org/10.1109/ic3.2018.8530608.

[69] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, "Traffic prediction based on random connectivity in deep learning with long short-term memory", *CoRR*, vol. abs/1711.02833, 2017. arXiv: 1711.02833. [Online]. Available: http://arxiv.org/abs/1711.02833.

[70] W. Liu, A. Hong, L. Ou, W. Ding, and G. Zhang, "Prediction and correction of traffic matrix in an IP backbone network", in *2014 IEEE 33rd International Performance Computing and Communications Conference (IPCCC)*, IEEE, Dec. 2014. DOI: 10.1109/pccc.2014.7017051. [Online]. Available: https://doi.org/10.1109/pccc.2014.7017051.

[71] L. Nie, D. Jiang, L. Guo, S. Yu, and H. Song, "Traffic matrix prediction and estimation based on deep learning for data center networks", in *2016 IEEE Globecom Workshops (GC Wkshps)*, IEEE, Dec. 2016. DOI: 10.1109/glocomw.2016.7849067. [Online]. Available: https://doi.org/10.1109/glocomw.2016.7849067.

[72] R. Vinayakumar, K. P. Soman, and P. Poornachandran, "Applying deep learning approaches for network traffic prediction", in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, Sep. 2017. DOI: 10.1109/icacci.2017.8126198. [Online]. Available: https://doi.org/10.1109/icacci.2017.8126198.

[73] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation", *CoRR*, vol. abs/1406.1078, 2014. arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078.

[74] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units", *CoRR*, vol. abs/1504.00941, 2015. arXiv: 1504.00941. [Online]. Available: http://arxiv.org/abs/1504.00941.

[75] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification", *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1257–1270, Aug. 2015, ISSN: 1063-6692. DOI: 10.1109/TNET.2014.2320577. [Online]. Available: https://doi.org/10.1109/TNET.2014.2320577.

[76] A. Madhukar and C. Williamson, "A longitudinal study of p2p traffic classification", in *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, ser. MASCOTS '06, Washington, DC, USA: IEEE Computer Society, 2006, pp. 179–188, ISBN: 0-7695-2573-3. DOI: 10.1109/MASCOTS.2006.6. [Online]. Available: https://doi.org/10.1109/MASCOTS.2006.6.

[77] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures", in *Proceedings of the 13th International Conference on World Wide Web*, ser. WWW '04, New York, NY, USA: ACM, 2004, pp. 512–521, ISBN: 1-58113-844-X. DOI: 10.1145/988672.988742. [Online]. Available: http://doi.acm.org/10.1145/988672.988742.

[78] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: Myths, caveats, and the best practices", in *Proceedings of the 2008 ACM CoNEXT Conference*, ser. CoNEXT '08, Madrid, Spain: ACM, 2008, 11:1–11:12, ISBN: 978-1-60558-210-8. DOI: 10.1145/1544012.1544023. [Online]. Available: http://doi.acm.org/10.1145/1544012.1544023.

[79] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning", *Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, Oct. 2008, ISSN: 1553-877X. DOI: 10.1109/SURV.2008.080406. [Online]. Available: https://doi.org/10.1109/SURV.2008.080406.

[80] R. Deebalakshmi and V. L. Jyothi, "A survey of classification algorithms for network traffic", in *2016 Second International Conference on Science Technology Engineering and Management (ICONSTEM)*, IEEE, Mar. 2016. DOI: 10.1109/iconstem.2016.7560941. [Online]. Available: https://doi.org/10.1109/iconstem.2016.7560941.

[81] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: Data collection and traffic classification", in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, IEEE, Nov. 2016. DOI: 10.1109/icnp.2016.7785327. [Online]. Available: https://doi.org/10.1109/icnp.2016.7785327.

[82] C.-N. Lu, C.-Y. Huang, Y.-D. Lin, and Y.-C. Lai, "High performance traffic classification based on message size sequence and distribution", *Journal of Network and Computer Applications*, vol. 76, pp. 60–74, 2016, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2016.09.013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S108480451630220X.

[83] P. Ducange, G. Mannara, F. Marcelloni, R. Pecori, and M. Vecchio, "A novel approach for internet traffic classification based on multi-objective evolutionary fuzzy classifiers", in *2017 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, IEEE, Jul. 2017. DOI: 10.1109/fuzz-ieee.2017.8015662. [Online]. Available: https://doi.org/10.1109/fuzz-ieee.2017.8015662.

[84] F. Ertam and E. Avcı, "A new approach for internet traffic classification: Ga-wk-elm", *Measurement*, vol. 95, pp. 135–142, 2017, ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2016.10.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0263224116305498.

[85] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things", *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017. DOI: 10.1109/access.2017.2747560. [Online]. Available: https://doi.org/10.1109/access.2017.2747560.

[86] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques", in *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2004, pp. 205–214. DOI: 10.1007/978-3-540-24668-8_21. [Online]. Available: https://doi.org/10.1007/978-3-540-24668-8_21.

[87] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning", in *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)l*, IEEE, 2005. DOI: 10.1109/lcn.2005.35. [Online]. Available: https://doi.org/10.1109/lcn.2005.35.

[88] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman, "AutoClass: A bayesian classification system", in *Machine Learning Proceedings 1988*, Elsevier, 1988, pp. 54–64. DOI: 10.1016/b978-0-934613-64-4.50011-6. [Online]. Available: https://doi.org/10.1016/b978-0-934613-64-4.50011-6.

[89] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms", in *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data*, ser. MineNet '06, Pisa, Italy: ACM, 2006, pp. 281–286, ISBN: 1-59593-569-X. DOI: 10.1145/1162678.1162679. [Online]. Available: http://doi.acm.org/10.1145/1162678.1162679.

[90] Z. Fan and R. Liu, "Investigation of machine learning based network traffic classification", in *2017 International Symposium on Wireless Communication Systems (ISWCS)*, IEEE, Aug. 2017. DOI: 10.1109/iswcs.2017.8108090. [Online]. Available: https://doi.org/10.1109/iswcs.2017.8108090.

[91] J. Hochst, L. Baumgartner, M. Hollick, and B. Freisleben, "Unsupervised traffic flow classification using a neural autoencoder", in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, IEEE, Oct. 2017. DOI: 10.1109/lcn.2017.57. [Online]. Available: https://doi.org/10.1109/lcn.2017.57.

[92] L. Grimaudo, M. Mellia, E. Baralis, and R. Keralapura, "SeLeCT: Self-learning classifier for internet traffic", *IEEE Transactions on Network and Service Management*, vol. 11, no. 2, pp. 144–157, Jun. 2014. DOI: 10.1109/tnsm.2014.011714.130505. [Online]. Available: https://doi.org/10.1109/tnsm.2014.011714.130505.

[93] S. Zhao, Y. Zhang, and P. Chang, "Network traffic classification using tri-training based on statistical flow characteristics", in *2017 IEEE Trustcom/BigDataSE/ICESS*, IEEE, Aug. 2017. DOI: 10.1109/trustcom/bigdatase/icess.2017.254. [Online]. Available: https://doi.org/10.1109/trustcom/bigdatase/icess.2017.254.

[94] A. F. M. Hani, I. V. Paputungan, and M. F. Hassan, "Support vector regression for service level agreement violation prediction", in *2013 International Conference on Computer, Control, Informatics and Its Applications (IC3INA)*, IEEE, Nov. 2013. DOI: 10.1109/ic3ina.2013.6819192. [Online]. Available: https://doi.org/10.1109/ic3ina.2013.6819192.

[95] J. Ahmed, A. Johnsson, R. Yanggratoke, J. Ardelius, C. Flinta, and R. Stadler, "Predicting SLA violations in real time using online machine learning", *CoRR*, vol. abs/1509.01386, 2015. arXiv: 1509.01386. [Online]. Available: http://arxiv.org/abs/1509.01386.

[96] J. Bendriss, I. G. B. Yahia, and D. Zeghlache, "Forecasting and anticipating SLO breaches in programmable networks", in *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, IEEE, Mar. 2017. DOI: 10.1109/icin.2017.7899402. [Online]. Available: https://doi.org/10.1109/icin.2017.7899402.

[97] P. M. Santos *et al.*, "PortoLivingLab: An IoT-based sensing platform for smart cities", *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 523–532, Apr. 2018. DOI: 10.1109/jiot.2018.2791522. [Online]. Available: https://doi.org/10.1109/jiot.2018.2791522.

[98] A. Stepchenko, J. Chizhov, L. Aleksejeva, and J. Tolujew, "Nonlinear, non-stationary and seasonal time series forecasting using different methods coupled with data preprocessing", *Procedia Computer Science*, vol. 104, pp. 578–585, 2017, ICTE 2016, Riga Technical University, Latvia, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2017.01.175. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187705091730176X.

[99] "VII. note on regression and inheritance in the case of two parents", *Proceedings of the Royal Society of London*, vol. 58, no. 347-352, pp. 240–242, Jan. 1895. DOI: 10.1098/rspl.1895.0041. [Online]. Available: https://doi.org/10.1098/rspl.1895.0041.

[100] W. P. Cleveland and G. C. Tiao, "Decomposition of seasonal time series: A model for the census x-11 program", *Journal of the American Statistical Association*, vol. 71, no. 355, pp. 581–587, 1976. DOI: 10.1080/01621459.1976.10481532. [Online]. Available: https://amstat.tandfonline.com/doi/abs/10.1080/01621459.1976.10481532.

[101] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root", *Journal of the American Statistical Association*, vol. 74, no. 366a, pp. 427–431, Jun. 1979. DOI: 10.1080/01621459.1979.10482531. [Online]. Available: https://doi.org/10.1080/01621459.1979.10482531.

[102] R. F. Carbone and J. S. Armstrong, "Note. evaluation of extrapolative forecasting methods: Results of a survey of academicians and practitioners", 1982.

[103] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[104] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *CoRR*, vol. abs/1412.6980, 2015.

[105] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12, Lake Tahoe, Nevada: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257.

[106] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", *CoRR*, vol. abs/1207.0580, 2012. arXiv: 1207.0580. [Online]. Available: http://arxiv.org/abs/1207.0580.

[107] M. Abadi *et al.*, *Tensorflow: Large-scale machine learning on heterogeneous systems*, 2015. [Online]. Available: https://www.tensorflow.org/.

[108] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks", in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16, Barcelona, Spain: Curran Associates Inc., 2016, pp. 1027–1035, ISBN: 978-1-5108-3881-9. [Online]. Available: http://dl.acm.org/citation.cfm?id=3157096.3157211.

[109] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying lstm to time series predictable through time-window approaches", in *Artificial Neural Networks — ICANN 2001*, G. Dorffner, H. Bischof, and K. Hornik, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 669–676, ISBN: 978-3-540-44668-2.

[110] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", *CoRR*, vol. abs/1502.03167, 2015. arXiv: 1502.03167. [Online]. Available: http://arxiv.org/abs/1502.03167.

[111] K. E. Iverson, *A Programming Language*. New York, NY, USA: John Wiley & Sons, Inc., 1962, ISBN: 0-471430-14-5.

[112] D. Rolnick, A. Veit, S. J. Belongie, and N. Shavit, "Deep learning is robust to massive label noise", *CoRR*, vol. abs/1705.10694, 2017. arXiv: 1705.10694. [Online]. Available: http://arxiv.org/abs/1705.10694.

[113] A. Fisher, C. Rudin, and F. Dominici, *All models are wrong but many are useful: Variable importance for black-box, proprietary, or misspecified prediction models, using model class reliance*, 2018. eprint: arXiv:1801.01489.

[114] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. DOI: 10.1080/14786440109462720.

[115] G. Fedrecheski, L. C. P. Costa, and M. K. Zuffo, "Elixir programming language evaluation for IoT", in *2016 IEEE International Symposium on Consumer Electronics (ISCE)*, IEEE, Sep. 2016. DOI: 10.1109/isce.2016.7797392. [Online]. Available: https://doi.org/10.1109/isce.2016.7797392.

[116] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing", 2011.

[117] S. Yang, "IoT stream processing and analytics in the fog", *IEEE Communications Magazine*, vol. 55, no. 8, pp. 21–27, Aug. 2017. DOI: 10.1109/mcom.2017.1600840. [Online]. Available: https://doi.org/10.1109/mcom.2017.1600840.

[118] P. Ta-Shma *et al.*, "An ingestion and analytics architecture for IoT applied to smart city use cases", *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 765–774, Apr. 2018. DOI: 10.1109/jiot.2017.2722378. [Online]. Available: https://doi.org/10.1109/jiot.2017.2722378.

[119] G. M. D'silva, A. Khan, Gaurav, and S. Bari, "Real-time processing of IoT events with historic data using apache kafka and apache spark with dashing framework", in *RTEICT*, IEEE, May 2017. DOI: 10.1109/rteict.2017.8256910. [Online]. Available: https://doi.org/10.1109/rteict.2017.8256910.

[120] "2018 apache kafka report", Confluent, 2018.

[121] P. Dobbelaere and K. S. Esmaili, "Kafka versus rabbitmq", *CoRR*, vol. abs/1709.00333, 2017. arXiv: 1709.00333. [Online]. Available: http://arxiv.org/abs/1709.00333.

[122] M. Bajer, "Building an IoT data hub with elasticsearch, logstash and kibana", in *FiCloudW*, IEEE, Aug. 2017. DOI: 10.1109/ficloudw.2017.101. [Online]. Available: https://doi.org/10.1109/ficloudw.2017.101.

Appendix A - 4G Network KPIs

In this Appendix, the 4G network KPIs are listed along with their description.

Name	**Description**
Active subscribers (Avg) [#]	Number of active users
Connected active subscribers downloading (Avg) [#]	Number of active users relative to download
Connected active subscribers uploading (Avg) [#]	Number of active users relative to upload
Radio Bearers (Avg) [#]	Average number of Radio Bearers
Cell Availability [seg]	Cell available time, in seconds
Cell Availability [min]	Cell available time, in minutes
Cell Availability Rate [%]	Percentage of the availability of the cell
Cell Planned Unavail Time [seg]	Planned unavailable time of the cell, in seconds
Cell Planned Unavail Time by Neighbours [seg]	Planned unavailable time of the cell neighbors, in seconds
Cell Planned Unavail Time [min]	Planned unavailable time of the cell, in minutes
Cell Unplanned Unavail Time [seg]	Unplanned unavailable time of the cell, in seconds
Cell Unplanned Unavail Time by Neighbours [seg]	Unplanned unavailable time of the cell neighbors, in seconds
Cell Unplanned Unavail Time [min]	Unplanned unavailable time of the cell, in minutes
Connected Subscribers (Avg) [#]	Average number of connected users
CSFB Prep Attempt [#]	Number of CSFB preparation attempts
CSFB Prep Success [#]	Number of CSFB preparation successes
CSFB Prep Success Rate	Success percentage of CSFB preparation
Cell Download Throughput (Avg) [Mbps]	Cell Average Download Throughput
Subscriber download Throughput (Avg) [Mbps]	User Average Download Throughput

PDCP Download Volume	Download PDCP Data Volume in the last Transmission Time Interval
PDCP Download TX Time [Sec]	Download PDCP Transmission Time in the last Transmission Time Interval
PRB Download Available [#]	Number of Physical Resource Blocks available for Download
PRB Download Used (Avg)	Average usage of Physical Resource Blocks for Download
PRB Download Utilization [%]	Percentage utilization of Physical Resource Blocks for Download
ERAB Drop Rate NonGBR [%]	ERAB drop rate for Non Guaranteed Bit Rate traffic
ERAB Drop Rate NonGBR (Reference) [%]	Reference value for the ERAB drop rate for Non Guaranteed Bit Rate traffic
ERAB Setup Attempt NonGBR [#]	Number of ERAB Establishment Attempts for Non Guaranteed Bit Rate traffic
ERAB Setup Failure Rate NonGBR [%]	ERAB Establishment Failure Rate for Non Guaranteed Bit Rate traffic
ERAB Setup Success Ratio	Average ERAB Establishment Success Ratio per hour
ERAB Setup Success NonGBR [#]	Number of ERAB Establishment Successes for Non Guaranteed Bit Rate traffic
ERAB Setup Success Rate NonGBR [%]	ERAB Establishment Success Rate for Non Guaranteed Bit Rate traffic
ERAB Setup Success Rate NonGBR (Reference) [%]	Reference value for the ERAB Establishment Success Rate for Non Guaranteed Bit Rate traffic
ERAB Abnormal Release NonGBR [#]	Number of ERAB Abnormal Releases for Non Guaranteed Bit Rate traffic
ERAB Normal Release NonGBR [#]	Number of ERAB Normal Releases for Non Guaranteed Bit Rate traffic
ERAB Release Success Rate	ERAB Release Success Ratio per hour
Handover Interfreq attempt [#]	Number of Handover Attempts Inter-frequency
Handover Intrafreq Attempt [#]	Number of Handover Attempts Intra-frequency
Handover Failure Rate [%]	Handover failure rate
Handover Interfreq eNode B Failure Rate [%]	Handover inter-eNB Failure Rate
Handover Interfreq eNodeB Success Rate [%]	Handover inter-eNB Success Rate
Handover Intrafreq eNodeB Failure Rate [%]	Handover intra-eNB Failure Rate
Handover Intrafreq eNodeB Success Rate [%]	Handover intra-eNB Success Rate
Handover Interfreq Success [#]	Number of Handover successes inter-frequency
Handover Intrafreq Success [#]	Number of Handover successes intra-frequency
Handover Success Rate [%]	Handover success rate
Handover Success Rate (Reference) [%]	Reference value for the Handover success rate
Active subscribers [#]	Number of maximum active users

Active subscribers download [#]	Number of maximum active users relative to download
Active subscribers (Max) [#]	Number of maximum active users relative to upload
Connected subscribers (Max) [#]	Number of maximum connected users
Measured Time	Cell measured time
PRB Used Average	Average usage of Physical Resource Blocks
RRC Setup Attempt [#]	Number of RRC establishment attempts
RRC Setup seizures DY	Number of RRC Establishment seizures per hour
RRC Setup attempts DY	Number of RRC Establishment attempts per hour
RRC Setup Failure Rate [%]	RRC Establishment failure rate
RRC Setup Success Rate DY	Number of RRC Establishments per hour
RRC Setup Success [#]	Number of RRC Establishment successes
RRC Setup Success Rate [%]	Success Rate of Establishment Success
RRC Setup Success Rate (Reference) [%]	Reference value for the success rate of Establishment Success
Upload Cell Throughput (Avg) [Mbps]	Cell Average Upload Throughput
Upload User Throughput (Avg) [Mbps]	User Average Upload Throughput
PDCP Upload Volume [MB]	Upload PDCP Data Volume in the last Transmission Time Interval
PDCP Upload Volume [MB]	Upload PDCP Data Volume
PDCP Upload Tx Time [Sec]	Upload PDCP Transmission Time in the last Transmission Time Interval
PRB Upload Available [#]	Number of available Physical Resource Blocks for Upload
PRB Upload Used Average	Average utilization of the Physical Resource Blocks for Upload
PRB Upload Utilization [%]	Utilization percentage of the Physical Resource Blocks for Upload