NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

GONÇALO FURTADO MATEUS

BSc in Computer Science Engineering

DATA AND COMPUTER CENTER PREDICTION OF USAGE AND COST

AN INTERPRETABLE MACHINE LEARNING APPROACH

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
November, 2022

NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

DATA AND COMPUTER CENTER PREDICTION OF USAGE AND COST

AN INTERPRETABLE MACHINE LEARNING APPROACH

GONÇALO FURTADO MATEUS

BSc in Computer Science Engineering

Adviser: Cláudia Alexandra Magalhães Soares
Assistant Professor, NOVA University Lisbon

Co-advisers: João Carlos Antunes Leitão
Assistant Professor, NOVA University Lisbon

Antonio José Rodrigues
DSI Director, novobanco

MASTER IN COMPUTER SCIENCE

NOVA University Lisbon
November, 2022

DATA AND COMPUTER CENTER PREDICTION OF USAGE AND COST

Copyright © Gonçalo Furtado Mateus, NOVA School of Science and Technology, NOVA University Lisbon, Novobanco.
The NOVA School of Science and Technology, the NOVA University Lisbon and the Novobanco have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

Acknowledgements

First and foremost, I would like to express my gratitude to my supervisors, Mrs. Claudia Soares and Mr. Antonio Rodrigues, for all the continuous and constructive support they gave me during this dissertation. I also like to express my gratitude to all *novobanco* colleagues for the fantastic experience, support, and insights they gave to me.

Next, I want to thank all my Integrated Master in Computer Science Engineering teachers for all the knowledge and values they transmitted to me during the five years of the course, which were fundamental for developing this work.

To my family, I want to express my gratitude for all their efforts that made it possible for me to reach here and for always supporting and encouraging me never to give up.

Last but not least, I wanted to thank all my friends and colleagues who were and are always present and whom, without their support, I would not have made it. In particular, a special thanks to Lourenço Vasconcelos, Pedro Silva, and Pedro Teixeira for the important change of ideas during this work.

Abstract

In recent years, Cloud computing usage has considerably increased and, nowadays, is the backbone of many emerging applications. However, behind cloud structures, we have physical infrastructures (data centers) in which managing is difficult due to unpredictable utilization patterns. To address the constraints of reactive auto-scaling, data centers are widely adopting predictive cloud resource management mechanisms. However, predictive methods rely on application workloads and are typically pre-optimized for specific patterns, which can cause under/over-provisioning of resources. Accurate workload forecasts are necessary to gain efficiency, save money, and provide clients with better and faster services.

Working with real data from a Portuguese national bank, in this thesis, we propose Ensemble Adaptive Model with Drift detector (EAMDrift). This novel method combines forecasts from multiple individual predictors to achieve a better one, giving weights to each individual model prediction. EAMDrift automatically retrain when needed and identifies the most appropriate models to use at each moment through interpretable mechanisms. However, beyond predicting workloads, we studied the influence of external factors (tweets related to the bank) on data center workloads. As we are working with real data from a bank, it is possible that users can increase or decrease the usage of some applications depending on external factors such as controversies or news about economics. For this study, EAMDrift was projected to allow multiple past covariates.

We evaluated EAMDrift in different workloads and compared the results with several baseline methods models. The experimental evaluation shows that EAMDrift outperforms individual baseline models in 15% to 25%. Compared to the best black-box ensemble model, our model has a slightly higher error (1-3%). Thus, this work suggests that interpretable models are a viable solution for data center workload prediction.

Keywords: Data center management, Interpretable machine learning, Dynamic prediction model, Natural language processing, Feature extraction

Resumo

Nos últimos anos, a computação em nuvem tem tido um aumento considerável e, hoje pode ser vista como a espinha dorsal de muitas aplicações que estão a emergir. Contudo, por detrás das conhecidas nuvens, existem estruturas físicas (centro de dados) nas quais, a gestão tem se revelado uma tarefa bastante difícil devido à imprevisibilidade de utilização dos serviços. Para lidar com as restrições do *auto-scalling* reativo, os mecanismos de gestão dos centros de dados começaram a adotar mecanismos preditivos. No entanto, os métodos preditivos são treinados com base nas cargas de utilização das aplicações e geralmente não estão otimizados para todos os padrões, causando sub/sobre provisionamento dos recursos.

Através da utilização de dados reais do centro de dados de um banco português, neste trabalho, propomos o Ensemble Adaptive Model with Drift detector (EAMDrift). Este novo método combina previsões de vários modelos individuais para obter um melhor resultado. O EAMDrift possui mecanismos interpretáveis que permitem detetar os melhores modelos em cada previsão, bem como detetar momentos para ser retreinado. No entanto, para além de prever a utilização dos recursos do centro de dados, neste trabalho iremos relacionar a influência de fatores externos (tweets relacionados com o banco) com a sua utilização. Sendo estes dados de um banco, é possível que os utilizadores aumentem ou diminuam o uso de algumas aplicações com base em fatores externos (polêmicas ou notícias sobre economia). Para isto, o EAMDrift permite o uso de covariadas passadas.

O modelo proposto neste trabalho foi avaliado em diferentes conjuntos de dados e os resultados foram comparados entre vários modelos de base. O EAMDrift superou todos os modelos de base em 15% a 25%. Quando comparado com o melhor modelo que também combina várias previsões mas de forma não interpretável, o nosso modelo obteve uma erro ligeiramente maior (1-3%). Assim, este trabalho sugere que modelos interpretáveis podem ser uma solução viável para a gestão dos centros de dados.

Palavras-chave: Gestão de centro de dados, Modelos de machine learning interpretáveis, Modelos de previsão dinamica, Processamento de linguagem natural, Extração de *features*

Contents

List of Figures ix

List of Tables xii

Acronyms xv

1 Overview: The growth of cloud computing 1
- 1.1 Context and Motivation . . . 1
- 1.2 Objectives . . . 2
- 1.3 Organization of the document . . . 3

2 State of the art 5
- 2.1 Data centers and machine learning . . . 5
  - 2.1.1 Data Centers overview . . . 5
  - 2.1.2 *Novobanco* data center structure . . . 6
  - 2.1.3 Challenges and power of machine learning . . . 7
  - 2.1.4 Related work . . . 8
- 2.2 Baseline machine learning models . . . 14
- 2.3 Interpretable methods for machine learning models . . . 21
  - 2.3.1 Why to provide explanations . . . 21
  - 2.3.2 How to achieve interpretability . . . 21
  - 2.3.3 RuleFit . . . 22
  - 2.3.4 Drift detection . . . 24

3 *Novobanco* data center 25
- 3.1 Data Overview . . . 25
- 3.2 Direct channels dataset . . . 26
  - 3.2.1 Data source, format, and pre-processing steps . . . 26
  - 3.2.2 Exploratory data analysis (EDA) . . . 28

4 Twitter data 35

4.1 Why is Twitter important . . . 35
4.2 Tweets Extraction . . . 35
4.3 Tweets from all media channels . . . 36
4.3.1 Data preprocessing and cleaning . . . 36
4.3.2 Exploratory data analysis (EDA) . . . 37
4.4 Tweets related to *novobanco* . . . 40
4.4.1 Filtering Tweets from mass media search . . . 41
4.4.2 Search tweets directly with keywords . . . 43

5 Interpreting the Twitter Data **46**
5.1 Natural Language Processing (NLP) . . . 46
5.2 Pre-processing of the dataset . . . 46
5.3 Lexicon-Based Sentiment analysis . . . 48
5.3.1 SentiLex-PT . . . 48
5.3.2 EMOTAIX.PT . . . 50
5.3.3 Validation of results . . . 51
5.4 Topic modelling . . . 55
5.4.1 DMM with Gibbs Sampling . . . 55
5.4.2 Coherence Metric . . . 57
5.4.3 Experiments . . . 57

6 Data center baseline analysis **64**
6.1 One server, different patterns . . . 64
6.2 Baseline Models study . . . 65
6.2.1 Results . . . 68
6.2.2 Covariates study . . . 71
6.3 Servers usage versus Twitter analysis . . . 71

7 EAMDrift: Methodology and Results **74**
7.1 Proposed model architecture . . . 74
7.2 Implementation of EAMDrift . . . 81
7.3 Model optimization . . . 82
7.3.1 Results . . . 85
7.4 Simulation scenario and model interpretability . . . 87
7.5 Validation of results . . . 91

8 Conclusion **94**
8.1 Main Contributions . . . 95
8.2 Limitations and future work . . . 96

Bibliography **97**

Appendices

A Datasets specification **104**

Annexes

I Questionnaire view **108**

II Other Metrics Plots **112**

II.1 RAM Utilization 112

II.2 Round trip average utilization 114

II.3 Write wait time 117

II.4 Google Cluster Trace 119

List of Figures

1.1 Thesis overview diagram 3

2.1 *Novobanco* Data Center (DC) structure. 6
2.2 DC workload versus capacity along time. Inaccurate workload forecasting results in under and over-provisioning of resources. (Image based on [12]) 8
2.3 Temporal Fusion Transformer model architecture [46]. 20
2.4 One decision tree generated in RuleFit algorithm. [51]. 23

3.1 *Novobanco* DC structure with highlight in the servers used in this work. . . 26
3.2 Round Robin Database (RRD) file schema. 26
3.3 Heatmap with correlated features degree. 28
3.4 Correlated features diagram. (Red corresponds to features positively correlated, and blue corresponds to features negatively correlated) 29
3.5 Write wait time metric over time. 30
3.6 CPU Average over servers and layers. 30
3.7 Cpu usage layers 31
3.8 Absolute CPU utilization of servers (excluding the ones that serve as backup). 32
3.9 CPU utilization of data center machines by day of month 32
3.10 CPU utilization of data center machines by day of week 33
3.11 CPU utilization of data center machines by hours 33
3.12 CPU utilization of data center machines by months 34
3.13 CPU utilization vs. Ram utilization. 34

4.1 Distribution of tweets of the main languages between 2014 and 2021. Data extracted from mass media tweets search. 37
4.2 Tweets count per year between 2014 and 2021. 37
4.3 Relation between the number of tweets per year, in percentage. 38
4.4 Tweets count per year between 2014 and 2021. 38
4.5 Relation between the number of news retweets and replies, grouping by year. 39
4.6 Relation between the number of news retweets and likes, grouping by year. 40

4.7 Relation between tweet's metrics (likes, retweets, and replies) over the years. 40
4.8 Number of tweets per year between 2014 and 2021. 41
4.9 Relation between the number of tweets per year, in percentage. 42
4.10 Tweets count per year between 2014 and 2021. 42
4.11 Number of tweets per year between 2014 and 2021. 43
4.12 Number of tweets in percentage of the main languages between 2014 and 2021. Data extracted from twitter keyword search. 44
4.13 Relation between the number of tweets per year, in percentage. 44
4.14 Tweets count per year between 2014 and 2021. 45

5.1 General steps use to pre-process texts before analyzing them. 47
5.2 Results of the lexicon-based sentiment analysis with SentiLex-PT dictionary. 49
5.3 Word cloud for each type of sentiment. 50
5.4 Evolution of sentiments present in tweets using EMOTAIX.PT lexicon. . . . 50
5.5 Results of the manual classification agreement. 53
5.6 Sentiment analysis using our ensemble model 55
5.7 Occurrences of main words in tweets related to novobanco. 58
5.8 Relation between the starting number of topic with the final number of topics and coherence. 59
5.9 Relation between the number of iterations with the final number of topics and coherence. 59
5.10 Relation between alpha and beta values with the final number of topics and coherence. 60
5.11 Number of documents per topic found in our final model. 61
5.12 Trend of the main topics and sentiment (with 7 days moving average) over time. 63

6.1 Patterns present in one single server. Different patterns (highlighted in grey) are found over time for each server. 64
6.2 Examples of workload patterns selected to observe model performance. . . 65
6.3 Illustration of the 30 days cross-validation procedures applied for testing the models. 66
6.4 Comparison of the Mean absolute percentage error (MAPE) values for all models and patterns. 70
6.5 Dashboard to compare the average server's usage with tweets related to *novobanco*. 73

7.1 Model architecture 75
7.2 Demonstration about how are historical data used to create the training set. 77
7.3 Examples of detected drifts using two different methods. 79
7.4 Time to train each model. 82
7.5 Mean error depending on the metric used. 83
7.6 Accuracy and validation error of testing different windows sizes for CPU. . 83

7.7 Accuracy and validation error of testing different windows sizes for covariates. 84
7.8 Testing different model configurations. 84
7.9 Influence of covariates in our proposed model. 85
7.10 Proposed method vs. baseline models in three different workloads. 86
7.11 Predictions in a simulations scenario of the workload average of the first layer. 87
7.12 Detailed analysis of the highlighted part in Figure 7.11. 88
7.13 Predictions in a simulations scenario of the workload average of the second layer. 89
7.14 Detailed analysis of the highlighted part in Figure 7.13. 90

I.1 Flow diagram of the internal structure of the developed application. 109
I.2 First page of our questionnaire. 110
I.3 Page with one question of our questionnaire. 110
I.4 Last page of our questionnaire. 111

II.1 RAM usage layers 112
II.2 Absolute RAM utilization of servers (excluding the ones that serve as backup). 112
II.3 Absolute RAM utilization of servers (excluding the ones that serve as backup). 113
II.4 Round trip average usage layers 114
II.5 Absolute Round trip average utilization of servers (excluding the ones that serve as backup). 114
II.6 Round trip average of data center machines by day of month 115
II.7 Round trip average of data center machines by day of week 115
II.8 Round trip average of data center machines by hours 115
II.9 Round trip average of data center machines by months 116
II.10 Write wait time of data center machines by day of month 117
II.11 Write wait time of data center machines by day of week 117
II.12 Write wait time of data center machines by hours 118
II.13 Write wait time of data center machines by months 118
II.14 Absolute CPU usage of Google Cluster Trace workload in hours. 119

List of Tables

2.1 Mean Absolute Percentage Error of the three best predictors depending on workload pattern. (Table from [25]) 10
2.2 Comparison of the resource estimation error of the four experiments using different models and datasets. (Table from [5]) 12
2.3 Comparison of techniques described in Related work. 13
2.4 Comparison of different baseline models. 14
2.5 Comparison of different interpretable models. (Table from [50]) 22

3.1 Dataset overview ranges (Data was taken on 9 August 2022). 27
3.2 Data center dataset overview. 28

5.1 Effect of the pre-process steps in text. 47
5.2 Comparison of techniques used to create manual text annotations. (P=Positive, N=Negative, W=Weak, S=Strong, n=Neutral) 51
5.3 Answers provided to the respondents to classify tweets. 52
5.4 Comparison of approaches used for Sentiment analysis. 54
5.5 Statistics about tweets related to novobanco. 58
5.6 Final topics found with the best model after removing topics with low frequency (less than 4). 62

6.1 Best model parameters for each pattern. 69
6.2 Mean Absolute Percentage Error of the best predictor depending on workload pattern. (- and + refer to workloads with low and high variation, respectively.) 70
6.3 Mean Absolute Percentage Error of each model individually. 71
6.4 Influence of covariates in the transformer model predictions. (- and + refer to workloads with low and high variation, respectively.) 72

7.1 Errors of the best baseline model and our proposed model for each workload pattern. 86
7.2 Rules with the highest coefficient for the first layer workload. 89

7.3 Rules with the highest coefficient created by the ensemble model for the second layer. 90
7.4 Validation of our proposed model. 92
7.5 Errors of baseline models and our proposed model for each workload pattern. 93

A.1 Media accounts that were used to retrieve data. (Information present in the table was last consulted in September 2022) 104
A.2 Features description of Twitter Dataset. Reasons descriptions for some features to have been discarded are presented in table A.4. 105
A.3 Description of metrics motorized in the novobanco data center. All the metrics are given in numerical values (with the respective scale) in a time series format. (*Reasons are presented in Table A.4) 106
A.4 Reasons for some features been discarded during data cleaning process . . 107

Acronyms

ACF Auto Correlation function 17, 67
ADWIN Adaptive Windowing 24, 79
AIC Akaike Information Criterion 17, 67
AR Autoregressive 10, 17
ARIMA Autoregressive integrated moving average 9, 10, 13, 16, 17, 67
ARMA Autoregressive and Moving Average 10
AWS Amazon Web Services 5

BIC BIC Bayesian Information Criterion 17, 67
BRDES Brown Double Exponential Smoothing 10

CDF Cumulative distribution function 24

DC Data Center ix, 5–9, 11–13, 26
DCN Data Center Network 7, 8
DMM Dirichlet Multinomial Mixture vii, 55, 56, 58, 60
DNN Deep Neural NetworK 11, 13

EAMDrift Ensemble Adaptive Model with Drift detector iv, v, 3, 4, 74, 75, 80–85, 87, 89, 91–96
EDA Exploratory data analysis vi, vii, 25, 28, 36, 37
ELU Exponential Linear Unit 20
EN Elastic Net 12

G-SVM Gaussian Support Vector Machine 10
GBT Gradient boosting tree 10, 13
GCT Google Cluster Trace 91, 92
GLU Gated Linear Units 20
GRN Gated Residual Network 20

GRU Gated Recurrent Unit 9, 13

I Integrated 17
IaaS Infrastructure as a Service 5

KR Krigin 13
KS Kolmogorov-Smirnov 24
KSWIN Kolmogorov-Smirnov Windowing method 24, 79, 80

L-SVM Linear Support Vector Machine 10
LASSO Least absolute shrinkage and selection operator 12
LDA Latent Dirichlet Allocation 55
LR Linear regression 10, 12, 13
LSTM Long-short Term Memory 9, 13, 14, 18, 67–70, 81, 89, 91–93

MA Moving Average 17
MAE Mean absolute error 80, 82
MAPE Mean absolute percentage error x, 66, 69–72, 80, 82, 85, 86, 92, 93
MOE Mixture of Experts 10
MSE Mean Squared Error 66–69, 77, 80, 82, 85, 86, 92, 93

NB Gaussian Naive Bayes 13
NLP Natural Language Processing vii, 3, 19, 46, 47, 95
NNLS Non-negative least square 12

Onprem On-premises data center 5–7

PaaS Plataform as a Service 5
PACF Partial Auto Correlation function 17, 67

RAM Random-access memory 71, 72
RDF Random Decision Forest 10, 91, 92
RFE Recursive Feature Selection 71
RNN Recurrent Neural Network 18
RR Ridge regression 12
RRD Round Robin Database ix, 26, 27
RTA Round trip average 71, 72

SaaS Software as a Service 5
SARIMA Seasonal Autoregressive Integrated Moving Average 14, 16, 17, 66–70, 81, 88–90, 92, 93

SVM Support Vector Machine 10, 12, 13, 91, 92

TFT Temporal Fusion Transformer ix, 14, 19–21, 66–71, 81

WWT Write wait time 71, 72

1

Overview: The growth of cloud computing

In this Chapter, we present an overview of this dissertation, highlighting the context and motivation of this work. It is also presented a description of the scope of this document.

1.1 Context and Motivation

With the continuous growth of technology, computer system resources, such as storage, have migrated to the cloud. In the last five years, it is estimated that the number of organizations using cloud services increased from 48% to 84% [1]. This is explained by the technological advancements, scalability, 24/7 on-demand availability, flexibility, and pay-as-you-go billing mode that cloud infrastructure offers [2]. On the other hand, the SARS-CoV-2 pandemic also has increased the use of cloud computing, showing its importance in connecting and sharing for people located in different physical places [3].

Behind clouds, we have physical infrastructures distributed over different locations, being that in each location, we have a data center. A data center is a facility that houses computer systems and related components such as storage, switches, and routers that work together to ensure excellent performance for programs that use them. Data centers typically include environmental controls (such as air conditioning to remove the heat generated by the servers and other components), various security measures, and redundant or backup power supplies.

However, this increased usage of cloud computing has created some problems in data center networks related to resource management. Unpredictable resource utilization patterns on the network make it difficult to plan and split operations between machines, which leads to situations where for the number of job requests in the network, the number of resources allocated to those jobs could be more optimal. When too few resources are allocated (under-provisioning), some resources must be activated on-demand, leading to performance problems. Likewise, when too many resources are allocated (over-provisioning), this will make some resources to be running idle [4]. Efficiently managing

data center resources is vital to maximize performance and obtain better operational costs [5].

In cloud computing, elasticity is defined by Herbst *et al.* [6] as *"the degree to which a system can adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible"*. Nowadays, different solutions try to achieve elasticity and allocate the best resources at each time, and this can be done in a reactive, proactive, or predictive scaling [7]. A reactive scaling monitors the application and adjusts its capacity to ensure optimum performance at a low cost. By name, reactive scaling means that it reacts to changes in the application, either in terms of resource failure or usage calls. An example of reactive scaling is the *Autoscaling* mechanism. It is one of the common approaches used by cloud providers (such as AWS) that dynamically adjusts data center resources based on differing workloads (e.g., job requests, CPU usage) and some user-defined criteria.

On the other hand, proactive scaling (or scheduled scaling) allows scaling applications resources based on expected future traffic. For example, if we expect the workload to be lower on weekends, we can set a rule to downgrade resources in that time. Finally, predictive scaling uses artificial intelligence to forecast the future workload to allocate the best resources at every moment.

Data center management urges automation to gain efficiency and provide better services to clients. Improving this process will allow for better resource allocation, faster services, and significant cost savings regarding resources and energy. Currently, the most used mechanism is reactive, as it is the one that provides more guarantees for the performance of the applications. Data center networks are critical areas, and the large-scale growth makes management a challenging task to be done by simple reactive or proactive methods.

1.2 Objectives

In this thesis, in collaboration with *novobanco*, we addressed the problem of using low-level network and computation metrics as feature signals to be used by a machine learning model to predict global usage of the data center. We used a real dataset from the *novobanco* data center to predict workloads.

However, more than predicting metrics, we had the objective of studying the influence of human context. For example, in the case of *novobanco*, users can increase or decrease the usage of some applications depending on external factors. Depending on controversies or news about economics and the bank itself, more or fewer operations can be done. As we worked with real data, in this thesis, we also tried to study if there was some relation between economic/bank news and resource utilization. For this, we used publications from Twitter because it is the most used real-time platform where people can freely express their opinions and ideas and in which social media channels publish

news to reach people quickly and easily. With the tweets extracted, we applied some Natural Language Processing (NLP) methods (sentiment analysis and topic modeling) to study them. The results of this analysis were posteriorly applied to our data center predictive model as covariates to study the relations. Covariates are variables given to machine learning models that can alter the relationship between the independent and the dependent variables in a statistical model. In other words, these variables provide more information and help the model predict the target outcome.

Although, creating a general predictive model that uses other covariates (in this case, Twitter) does not give us much information about the possible relations. Instead, it tells us if covariates influence the outcome positively or negatively. As a result, more than having a model that uses human context information to predict the data center usage, we proposed a novel interpretable machine learning method (EAMDrift) that provides some relations and explanations for the predictions—having interpretability, not just increases the understanding of the relations between data center usage and human context, but also allows human decision-makers to have a high level of confidence in forecasts.

In sum, the objective of this thesis is demonstrated in Figure 1.1. We used two datasets (one with data center metrics and the other with tweets), and our main objective will be to predict data center usage. Moreover, we provided some interpretable mechanisms in our proposed model that allowed us to study the real influence of covariates (tweets) in our predictions more deeply.

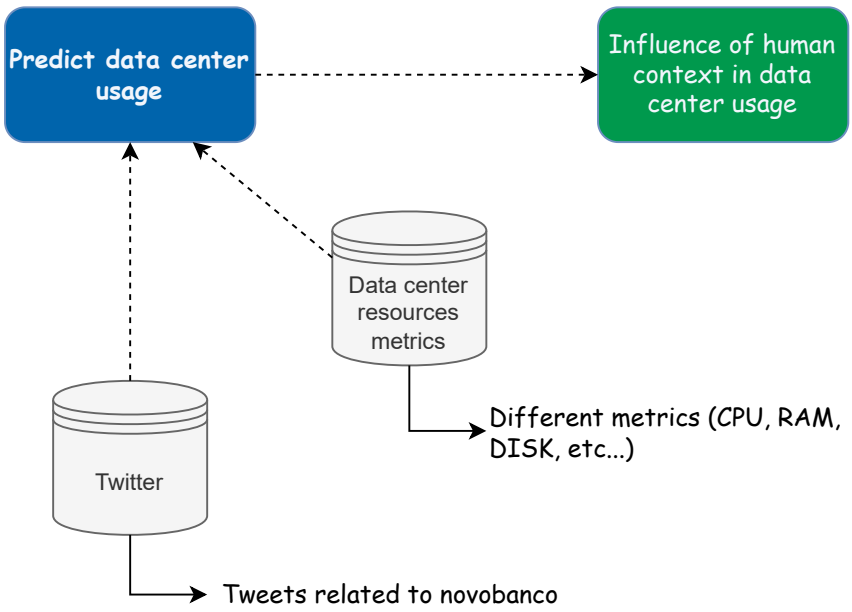


Figure 1.1: Thesis overview diagram

1.3 Organization of the document

The remaining of this thesis is organized as follows:

1. Chapter 2 (State of the art) summarizes the required knowledge needed to understand the subsequent chapters in this dissertation. In the first part of the Chapter, we present some background to frame our work and related works. The second part is dedicated to interpretable and non-interpretable models, explaining their importance to our application.

2. Chapter 3 (*Novobanco* data center) aims to explain the dataset of *novobanco* data center used in this thesis. This Chapter presents an initial overview of this dataset, its format, and how it was obtained. Next, we analyze their features, excluding correlated and unnecessary features for our final objective.

3. Chapter 4 (Twitter data) describes the Twitter dataset that is used in this thesis. This Chapter explains why Twitter was chosen as our dataset source to study human context and analyzes the dataset.

4. Chapter 5 (Interpreting the Twitter Data) outlines two different approaches, Sentiment analysis and Topic modeling, used to interpret the content of the tweets dataset. We start by presenting how we prepared the text of tweets to be analyzed, and next, the experiments and the results obtained with the two methods used.

5. Chapter 6 (Data center baseline analysis) provides a baseline analysis over the servers presented in our dataset, providing some essential baseline insights for the model that was developed. Furthermore, it is also studied some relations between *novobanco* data center servers usage and human context (i.e., tweets related to *novobanco*).

6. Chapter 7 (EAMDrift: Methodology and Results) presents the proposed model of this thesis (EAMDrift). We explain the model architecture, followed by some tests and results. In the end, we compare our proposed model to the baseline models used in this work and validate the results in a different dataset.

2

State of the art

In this Chapter, we summarize the required knowledge needed to understand the subsequent chapters in this dissertation. We address some of the works related to our own, providing their context and motivation for why it is relevant.

2.1 Data centers and machine learning

2.1.1 Data Centers overview

Data Center (DC) is a physical facility that houses critical computing resources as data or applications [8]. They comprise several components such as servers, switches, and routers that work together to ensure a high level of performance to applications and programs that are using it [9]. DC is a crucial point in cloud computing as it enables users to access resources and process programs on-demand [10]. Also, DC needs to be scalable and efficient to provide a fast and secure connection between all components, to deal with the continuous evolution of the network, and the ever-growing cloud computing needs [11]. Companies regularly use them to centralize information in one point in a way that can be accessed from everywhere.

Two DC solutions are typically adopted by the big companies: On-premises data center (Onprem) and cloud solutions. Onprem refers to private DC houses in companies' facilities and maintained by themselves. In contrast, cloud computing is defined as services provided by third-party service providers. There are different types of services provided by cloud computing, but the more used and known are Infrastructure as a Service (IaaS), Plataform as a Service (PaaS), and Software as a Service (SaaS) solutions. Cloud solutions bring advantages such as better energy costs, scalability, high levels of security, and predictable costs. With the growth of this type of computation, many cloud solutions have been appearing provided by big companies such as Amazon (AWS), Google (Google Cloud), Microsoft (Azure), or IBM (IBM Cloud). However, the usage of this kind of cloud solution brings some disadvantages, mainly for countries more peripheral as

Portugal, because usually, the DC physical structures are located in Europe center regions. That is why some companies do not have their resources totally in the cloud and additionally use private data centers housed in companies' facilities and maintained by themselves (Onprem). Besides, this approach allows not to be dependent on either cloud providers or Onprem data centers.

2.1.2 *Novobanco* data center structure

In the specific case of *novobanco*, their DC is organized as Figure 2.1 shows. *Novobanco* follows the practices of having either Onprem and cloud data centers, which are used for different operations. Knowing that *novobanco* is a bank, it is required to have a high level of security and be careful in all operations that need to access the private resources of the bank so that it does not happen that the bank's customers are harmed. The following explanation will present the general overview of *novobanco* data center more related to the direct channels, as these are the ones that deal with most of the critical and private information of the bank and, therefore, the most important to optimize.

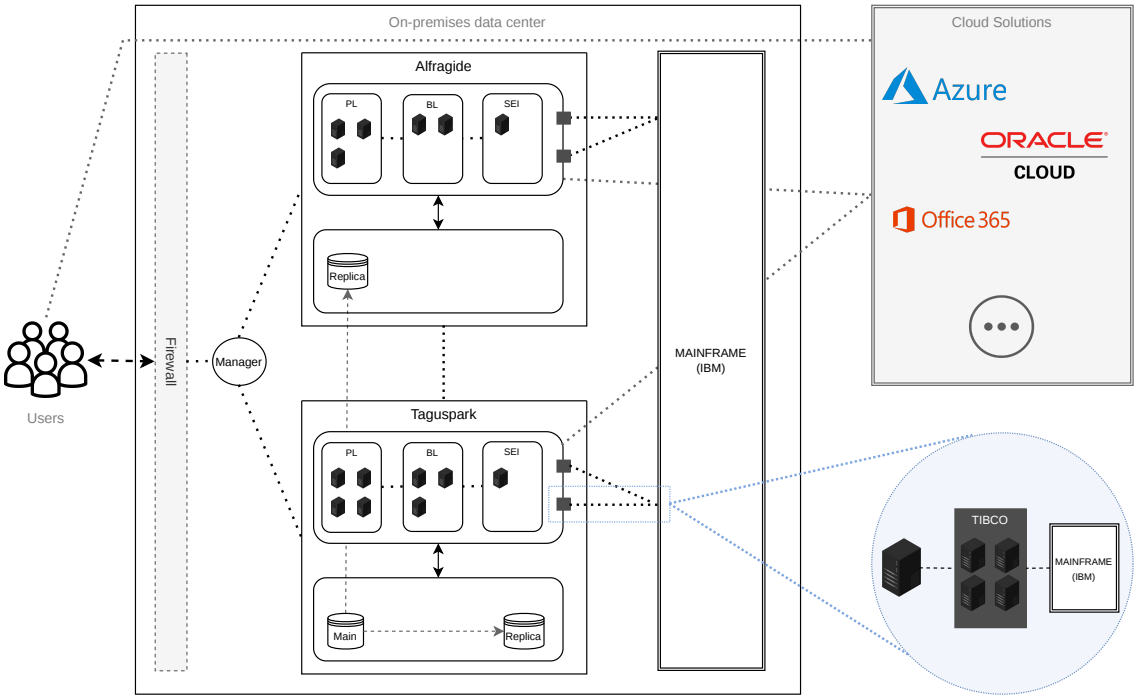


Figure 2.1: *Novobanco* DC structure.

Focusing on Onprem data centers, there are two data centers managed and maintained by *novobanco* (Taguspark and Alfragide) and an additional one (Mainframe) that IBM maintains. The data center located in Taguspark is the main one and is always active, and the one located in Alfragide works as a backup data center and is not always active. Nevertheless, the database of the Alfragide data center is always active to provide a secure and updated backup of data. The internal structure of both data centers maintained by *novobanco* is the same and is organized in layers (PL, BL, SEI, BD). The first layer works

mainly as the front-end layer, and as the layers go on, the operations made in each of them require more processing skills and security. To guarantee security, firewalls exist before entering the data center and between all layers (PL, BL, SEI, BD). Note that there are different servers for different operations, and Figure 2.1 just represents the overall architecture of *novobanco* data center.

Although this architecture already guarantees a high level of security, private information such as clients' accounts is not located in either Taguspark or Alfragide data center. Instead, this information is located in a data center owned by *novobanco* but maintained by IBM (Mainframe). As can be noticed by looking at Figure 2.1, the information located in Mainframe can just be accessed through the Onprem Taguspark and Alfragide data centers using a set of servers called *TIBCO*. This *TIBCO* servers process the information to the maximum so that the Mainframe requests are simple and fast, without extensive operations processing. Moreover, *TIBCO* servers add an additional layer of security and work as an API to facilitate all the connections with the Mainframe. An example is when a user wants to log in to the *novobanco* app, he has to pass through several layers and arrive at the *TIBCO* servers, where it will be processing what is needed to contact Mainframe with a request-reply request, and return if the login was successfully or not. The same logic applies to the bank account balance and other operations.

With the growth of cloud computing, *novobanco* has also started migrating some applications to the cloud. However, even though some applications can be accessed directly to the cloud by users when a cloud application needs to access private information located in Mainframe, it needs to make a request to the Onprem data centers. After the request passes through the multiple Firewalls, the information is returned to the cloud application.

Finally, it is also possible to analyze that before the user's request arrives at the Onprem data centers, there is an *Manager*. This *Manager* aims to coordinate operations and, in case the Main data center (Taguspark) is overloaded, distribute the requests amount to either Taguspark or Alfragide data centers, always maintaining the coherence in operations.

In sum, we can notice that the internal structure has many Firewalls, and to access private information many steps need to be guaranteed. This makes it difficult for a cyberattack to succeed, as there are many stages that cybercriminals need to pass unnoticed.

2.1.3 Challenges and power of machine learning

Data center networks are growing fast, and managing Data Center Network (DCN) is already a tedious process. This process will become tough to improve only by humans due to random patterns/behaviors in future networks. DC urges the need for automation, which can be reached with machine learning models for automated management and optimization of the network. These models could automatically optimize data center machines and spread the work between them, avoiding having data center resources

under or over-provisioning, as depicted in Figure 2.2. A good resource estimation model will lead to better planning, schedule, and management of DC, which saves money. In the specific case of *Novobanco*, and as will be possible to observe later in Chapter 3, what happens, as a rule, is to have a provisioning mechanism superior to what is necessary (over-provisioning) so that servers never fails. Thereby, the case of under-provisioning will be rare, and it is not expected to be found very often.

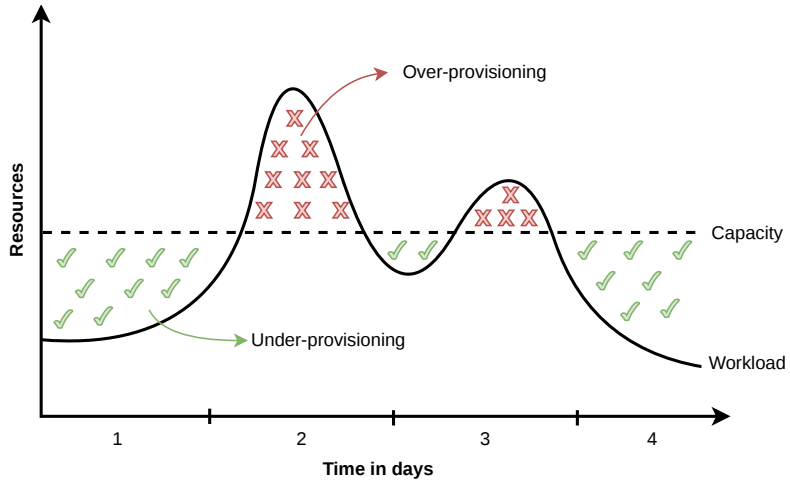


Figure 2.2: DC workload versus capacity along time. Inaccurate workload forecasting results in under and over-provisioning of resources. (Image based on [12])

However, machine learning algorithms in DC are more complex in this field. Firstly, few real datasets can be used, and one of the reasons is that companies cannot and do not want to expose their client's data and data center architecture mechanisms. Secondly, the traffic that passes in DCN is a mix of mice and elephant flows, but most traffic is mice flows sent before any optimization takes place. Because of this, to avoid models creating more latency on the network, it is necessary to choose machine learning model architectures that can process data and make decisions in order of microseconds. Another challenge in machine learning is that the data needed to train a machine learning model should be kept from the network because it is too risky to let data leave its host because of privacy [12]. Studies show that some machine learning models in some points of the network are an easy target for security and privacy threats because data could be easily poisoned [13].

2.1.4 Related work

With the growth of machine learning, researchers have been developing machine learning methods to help manage DC. Several areas could help improve DC, but along this subsection, more importance will be given to critical areas that leverage machine learning and are relevant to our work (Workload forecasting and Traffic classifications). In Section 2.1.4.4, we will also briefly talk about other approaches that have been made to optimize DCN using machine learning models.

2.1.4.1 Single model approaches

Diverse approaches have been made to predict DC workloads. Models have been designed based on statistical and machine learning domains, generally relying on regression, time-series, machine learning, and recurrent neural networks [14, 15].

One of the most common approaches to DC resources forecasting is using time series models. Time series models use a sequence of data points over an interval to predict future outcomes [16]. Fang *et al.* used Autoregressive integrated moving average (ARIMA) to estimate CPU utilization [17]. Calheiros *et al.* also used ARIMA to predict the arrival rate of workload [18]. However, time-series approaches failed to make good predictions in unpredictable workload patterns because it is challenging to fit this kind of model in these situations [16, 19].

Jayakumar *et al.* developed the LoadDynamics, a machine learning model based on Long-short Term Memory (LSTM) [4]. LSTM models were used because they can efficiently learn long-term dependencies. With the correct hyperparameters, these models can detect and predict various patterns in a workload [20]. The workload input is split into intervals, and each LSTM cell receives as input a workload interval to train this model. The authors used Bayesian Optimization to find the best hyperparameters for the model. BO is an iterative process that, in each iteration, searches for the best hyperparameters using a stochastic process, namely a Gaussian Process. After a fixed number of iterations, the best model is chosen and used as the final predictor.

The idea of Khan *et al.* was to create a model that could predict not only one DC resource but all resources that significantly affect energy consumption, such as disk operations, CPU, memory, and network transmission rate [21]. They investigated several machines and deep learning algorithms and found that Gated Recurrent Unit (GRU) model had lower error values for all the estimated resources. This model allows predicting energy state in future points because the abovementioned resources significantly impact energy consumption.

Workload prediction depends a lot on the type of flows in the network. Classifying the types of flows allows better workload predictions because of the influence that large flows, which require a high transfer rate, have on the workload. Zhu *et al.* proposed SmartTrans, a system that uses deep learning methods for traffic classification (that affects the schedule in the network) and flow size rank prediction (that affects the flow completion time). [22] Estrada-Solano *et al.* also proposed a model to predict the type of flows in the network [23]. The model comprises an analyzer that analyzes and classifies the outgoing package and a learner that trains a model to help the analyzer classify the packages. In this case, the authors are dedicated to predicting elephant flows of great magnitude.

2.1.4.2 Multi-model approaches

Studies showed that a single model is not sufficient to lead with unpredictable patterns and behaviors that the network flow has [24–26]. A study conducted with 21 widely-used

prediction algorithms in four different workloads showed that the top predictors vary considerably for different workload patterns, as illustrated in Table 2.1. For different workload patterns, different models yield better estimations.

Table 2.1: Mean Absolute Percentage Error of the three best predictors depending on workload pattern. (Table from [25])

Increasing Workload		**On and Off Workload**		**Cyclic Bursty Workload**		**Random Workload**	
L-SVM	28%	**G-SVM**	22%	**ARIMA**	38%	**G-SVM**	45%
AR	29%	**ARMA**	30%	**BRDES**	41%	**LR**	46%
ARMA	30%	**L-SVM**	44%	**L-SVM**	43%	**L-SVM**	46%
Average	**51%**	**Average**	**69%**	**Average**	**75%**	**Average**	**52%**

Understanding this problem, Iqbal *et al.* proposed a novel method that adaptively and automatically identifies the most appropriate model to be used in some specific scenarios [24]. They used classical machine learning methods such as Linear regression (LR), Support Vector Machine (SVM), Gradient boosting tree (GBT), et al., to predict workload.

The proposed novel method is divided into two parts. In the first part ("Adaptive Prediction Method Learning"), the system receives historical resource utilization data divided into sliding windows as input. The system trains different machine learning models for each sliding window and predicts which model performs better. The selected features of the observed sliding window and the best prediction method are stored in the training data. In the second part, the training data created in this method's first part is inserted as input. The system prepares a Random Decision Forest (RDF) to predict the best model for a specific sliding window with training data. Once the RDF is trained, the system can predict the data center resource utilization in real-time for each sliding window, using the best model, considering the current window features. This system works by simply switching which model to use along time.

Liu *et al.* also proposed an adaptive prediction approach that categorizes the workloads into different classes based on workload features. The proposed method dynamically selects LR for slow-changing workloads and SVM for fast-changing workloads to estimate the CPU utilization of machines [26].

Differently, from the two models mentioned above, Kim *et al.* proposed a method called CloudInsight, that estimates future workload for the data centers by dynamically determining the weight of each predictor [25]. CloudInsight is inspired by the Mixture of Experts (MOE) problems [27]. The MOE's key insight is that a group decision made by all local experts is often preferable to a single expert's decision [28]. However, unlike the general MOE approach that relies on a simple linear combination of all local experts, CloudInsight creates an ensemble model with different weights for each local predictor. During the training process, CloudInsight creates and adjusts an output vector ($n \times 1$)

with each predictor's weight based on their accuracy for the current workload. CloudInsight uses this vector to create the ensemble model based on the following weighted average equation:

$$\text{Ensemble model} = \frac{\sum_{i=1}^{n} w_i p_i}{\sum_{i=1}^{n} w_i}$$

where w_i is the predictor i weight and p_i is the prediction from predictor i. CloudInsight gives higher weights to potentially more accurate predictors and lower weights to potentially less accurate predictors.

2.1.4.3 Importance of window size

Developed works also noticed the significant improvement that window size selection has in the workload prediction of DC. Workload changes are highly volatile, and fixed-size observation windows cannot always capture the local trends in the data. Studies show that if, at every estimation step, an exhaustive search to find the best observation window size is done, it is possible to significantly reduce the prediction estimation error [5, 19, 24].

Dalmazo *et al.* proposed a system to estimate network traffic using statistical techniques. This approach adjusts the window size based on the variance of the current, and previous sliding windows [29].

Baig *et al.* created a system to find the best window size for an observation [5]. In the first part of this system (Training Data Set Preparation), the system receives as input historical resource utilization logs of a data center split in sliding windows with a fixed size. For each sliding window, the system does a linear search to find the ideal length of that window that minimizes the prediction error when predicting the next interval's resource consumption. The system builds training data with the data from each sliding window and the identified corresponding optimal window size. Once all sliding windows are analyzed, the system enters the second part (Resource Utilization Prediction). Here, with the training data from the previous step, a Deep Neural NetworK (DNN) is trained to estimate the best window size for a given sliding window data. After training, the DNN model predicts the best window size for the current sliding window observations.

Using different machine learning models and comparing the *Baig*, *Dalmazo*, and fixed-window-size (FixedW) method revealed that the system proposed by *Baig* could improve the prediction accuracy from 16% to 54% compared to other models. (A comparison of these three methods made in [5] is presented in Table 2.1).

2.1.4.4 Other works

Beyond workload forecast and classification, machine learning is being applied in other DC problems to optimize and automate networks.

Table 2.2: Comparison of the resource estimation error of the four experiments using different models and datasets. (Table from [5])

Data set	Experiment	LR	SVM	EN	LASSO	RR	NNLS	Average
Alibaba	FixedW	0.61	0.57	0.53	0.56	0.61	0.68	0.59
	Dalmazo	0.83	0.91	0.82	0.82	0.83	1.00	0.87
	Baig	**0.40**	**0.38**	**0.36**	**0.39**	**0.37**	**0.41**	**0.39**
Materna	FixedW	0.55	0.54	0.50	0.53	0.55	0.58	0.54
	Dalmazo	0.61	0.75	0.60	0.60	0.61	0.65	0.64
	Baig	**0.49**	**0.49**	**0.46**	**0.47**	**0.48**	**0.42**	**0.47**
Bitbrains	FixedW	0.23	0.32	0.19	0.22	0.21	0.24	0.24
	Dalmazo	0.56	1.00	0.55	0.55	0.56	0.64	0.64
	Baig	**0.20**	**0.22**	**0.17**	**0.20**	**0.19**	**0.20**	**0.20**

Chen *et al.* proposed a model to make traffic optimization decisions locally (i.e., application server), named *AuTO* [30]. This model uses deep reinforcement learning to optimize traffic of the long flows, adjusting the sending rate at end-hosts and avoiding congestion in the receiver queue. Topology management, which consists in finding good link connections in the network to provide good connectivity, is also a big problem in DC that requires automation. Salman *et al.* proposed a model that uses Deep Reinforcement Learning, and according to the network state, selects the links to activate [31]. Different kinds of failures, such as server, link, and switch, can happen in DC. Arzani *et al.* created a system based on random forest models to identify causes of failures and anomalies in the network [32]. With the growth of DC, it is also essential to have security in mind. Tongaonkar *et al.* proposed SANTaClass, an unsupervised machine learning model that uses network traffic to generate protocol signatures [33]. This model can also identify encrypted communications and distinguish between VPN and non-VPN networks.

2.1.4.5 Conclusions

Several different approaches have been made to try to predict workload in DC. In this Section (2.1.4), we started by presenting some single models that predict workload. Even though these models present promising results, studies showed that a single model is not enough to adapt to all unpredictable network flow changes. Adaptive models can address variable-nature workloads and adapt to the stochastic changes of workloads. Experimental results prove that adaptive methods that choose the best model for each sliding window outperform all other models that use a single model to predict. However, a multi-model strategy brings the overhead problem to the discussion. On average, these approaches are computationally expensive and take more time to train and predict. Knowing that DC decisions have to be made fast, we must not disregard the single model approaches as these can lose a small amount of accuracy but gain in computation and time.

Table 2.3: Comparison of techniques described in Related work.

Work	Observed Metric	Algorithms	Objective	Periodically re-trains
Fang [17]	CPU	ARIMA	Workload prediction	X
Calheiros [18]	Job requests	ARIMA	Workload prediction	X
Jayakumar [4]	Job arrival rate	LSTM	Workload prediction based on long-term dependencies	X
Khan [21]	Disk operations, CPU, Memory, Network	GRU	Predict DC resources that affects the energy consumption	X
Zhu [22]	Packets transmitted	Deep Learning	Flow type Prediction	X
Estrada-Solano [23]	Packets transmitted	Incremental learning algorithms	Large (elephant) flows Prediction	YES
Iqbal [24]	CPU	LR, SVM, KR, GBT, NB	Workload prediction by selecting the best predictor for the observed sliding window	X
Liu [26]	CPU	LR, SVM	Workload prediction by using (LR) for slow-changing workloads and (SVM) for fast-changing workloads	X
Kim [25]	Job arrival rate	21 prediction algorithms	Workload prediction by assigning an importance weight to each predictor	X
Dalmazo [29]	Network traffic	Statistical techniques	Adjusts window size based on the variance of the current and previous sliding windows	X
Baig [5]	CPU	DNN	Predicts the ideal length for the current observed sliding window	X

We also have to consider that machine learning needs continuous monitoring to learn new patterns over time, which can create latency in the network [24]. Excepting the Estrada-Solano *et al.* method, none of the works presented consider this aspect.

In the major part of these models, authors use CPU resource estimations, but models can be adapted to predict other resources. Studies show that CPU consumes 58% of the total host's energy, making it one of the most critical resources to be optimized [34]. Our research did not find any proposed system that uses interpretable models to explain its predictions (this will be discussed in subsequent Section 2.3). Table 2.3 summarizes and compares the main and more relevant works presented in this Section.

2.2 Baseline machine learning models

Six models were chosen to obtain a good baseline accuracy for our data center dataset: Exponential Smoothing, Seasonal Autoregressive Integrated Moving Average (SARIMA), Prophet, Long-short Term Memory, Transformer, and Temporal Fusion Transformer (TFT). We chose models of different characteristics and natures, from more uncomplicated to complex ones, to have a set that could effectively compare the different models in our dataset. This will be important to compare the performance of different models and to determine whether the model are performing well or not.

In Table 2.4, we compare the features of the baseline models chosen for this work. The univariate and multivariate refer to the number of variables used to model the data. A univariate model is a model that uses a single variable to describe the time series data. This model type is used when there is only one variable of interest, and the goal is to model the relationship between this variable and time. A multivariate model, on the other hand, is a model that uses multiple variables to describe the time series data. Multivariate models are used when multiple variables are believed to be related to each other and the response variable, and the goal is to model the relationships between all of these variables. In general, multivariate time series models are more complex. They can provide more detailed insights into the relationships between the variables, but they are also more challenging to fit and interpret. Multiple-series training refers to training a model on multiple time series simultaneously rather than training a separate model for each time series individually. Past covariates refer to variables observed at some point but no longer available at the current time. Future covariates refer to variables that are not observed at the current time point but are expected to be available in the future and are believed to be related to the response variable of interest. Static covariates are variables that are not time-dependent and do not change throughout the time series [35]. Next, we will discuss each model more deeply.

Table 2.4: Comparison of different baseline models.

Algorithm	Univariate	Multivariate	Multiple-series training	Past covariates	Future covariates	Static covariates
Exponential Smoothing	X					
Prophet	X				X	
SARIMA	X				X	
LSTM	X	X	X		X	
Transformer	X	X	X	X		
TFT	X	X	X	X	X	X

Exponential Smoothing

Exponential smoothing is a simple and effective method for forecasting time series data, especially when there is a slight trend or seasonality [36]. It is beneficial for forecasting data that is subject to change slowly over time, and this method aims to produce a smoothed version of a given time series. It does this by using a weighted average of past data points, with the weighting decreasing exponentially as the data gets older. The method produces a forecast for the next period based on the smoothed data. There are several variations of exponential smoothing, including simple exponential smoothing, double exponential smoothing (also known as Holt's method), and triple exponential smoothing (also known as Holt-Winters method) [37]. In simple exponential smoothing, the forecast for the next period is based solely on the most recent data point. In contrast, in double exponential smoothing, the forecast also considers the trend of the data.

Moreover, in the Holt-Winters or triple exponential smoothing method (the one that was used in this work), the forecast takes into account both the trend (b_t) and the seasonality (s_t) of the data, and the seasonal component can be either additive or multiplicative. The additive method is more suitable when the seasonal fluctuations remain approximately the same along the time series. In contrast, the multiplicative method is more appropriate when the seasonal variations change in proportion to the level of the series. The forecast of the Holt-Winters (triple exponential smoothing) method is calculated using the equation F_{t+1} (shown in 2.1[1]), where the level, trend, and seasonal component equations are used to forecast the following time series value [38]. In equation 2.1, $l(t)$ is the level component, y_t is the observed value at time t, s is the period of the seasonality, and α, β, and γ are the smoothing coefficients for the level, trend, and seasonality components, respectively.

$$
\begin{aligned}
F_{t+1} &= l_t + b_t + s_t \\
l_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \\
b_t &= \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \\
s_t &= \gamma(y_t - l_t) + (1-\gamma)s_{t-m}
\end{aligned}
\tag{2.1}
$$

Prophet

The Prophet model [39] is a powerful tool for forecasting time series data. Developed by Facebook, it is designed to be flexible and easy to use while providing good forecast accuracy. One of the main features of the Prophet model is its ability to detect and incorporate trends and seasonality in the data automatically. It also allows the user to specify additional regressors or external variables that may impact the time series. The Prophet model is implemented in Python and R and is available as an open-source package. The Prophet model brings some advantages compared to other models (for

[1] Note that this equation refers to the additive form of Holt-Winters method as this was the one that achieved best results in our work

example, Exponential Smoothing) as it automatically selects the appropriate model for the time series based on the data. Hence, there is no worry about tuning its hyperparameters. Prophet can also handle multiple seasonalities (daily, weekly, and yearly seasonality), outliers, and missing data. To fit a Prophet model, we need to provide a Data Frame with a "ds" column for the date and a "y" column for the variable we want to forecast.

Overall, the Prophet model decomposes a time series into trend, seasonality, and noise components and then uses machine learning techniques to make predictions. This model has some similarities with Exponential smoothing as the forecasts can also be described as the sum of three main components, following equation 2.2, where: $y(t)$ is the predicted value at time t, $g(t)$ is the trend component at time t, $s(t)$ is the seasonal component at time t, $h(t)$ is the holidays component at time t, and $\epsilon(t)$ is the error term at time t.

$$y(t) = g(t) + s(t) + h(t) + \epsilon(t) \tag{2.2}$$

The trend component ($g(t)$) represents the long-term underlying pattern in the time series data. It can be modeled using a piecewise linear, consisting of several straight-line segments between *changepoints*. Although, Prophet can also be modeled using a logistic curve equation if the data carrying capacity, growth rate, and offset vary over time and change at each *changepoint*. These *changepoints* are a Prophet model innovation and can be defined as instances in which there is a significant change in data behavior. The model automatically determines them based on the data properties, but the user can also manually tune them.

The seasonality component ($s(t)$) is modeled using the Fourier series, a mathematical function representing periodic signals as a combination of sine and cosine functions. This component allows Prophet to capture more complex patterns and model the cyclical patterns, or seasonality, present in the data. Seasonality can be added to the model either in an additive or multiplicative way. The default value for this is additive, meaning that the seasonality effect will be added to the trend component of the model. Conversely, a multiplicative seasonality means that the seasonality effect will be multiplied by the trend component.

The holidays component ($h(t)$) allows the model to account for events that significantly impact the time series. Including a holiday component requires the holiday name and the holiday range of days.

Lastly, Prophet also has an error term component ($\epsilon(t)$) that accounts for any remaining uncertainty or randomness in the data that were not explained by the trend, seasonality, or holiday components. It can be seen as the "noise" component and is incorporated through a likelihood function.

SARIMA

The third model selected for this work was Seasonal Autoregressive Integrated Moving Average (SARIMA), and can be seen as an extension of the ARIMA model. Like ARIMA,

SARIMA is a linear time series model that considers Autoregressive (AR), Integrated (I), and Moving Average (MA) components of the data but has an advantage compared to ARIMA that is a component regarding seasonality. So, this model becomes particularly useful for modeling time series data that exhibit seasonal patterns. In SARIMA, the AR component models the relationship between the current value of the time series and its past values, the I component models make the time series stationary, and the MA component models the relationship between the current value of the time series and the errors or residuals from past forecasts. Lastly, the seasonal component models the periodic patterns in the data, such as monthly or quarterly patterns.

SARIMA model can be defined as $SARIMA(p,d,q)(P,D,Q)s$, and to fit the model, we need to specify the values of the hyperparameters, i.e., the non-seasonal order (p,d,q) and the seasonal order $(P,D,Q)s$. p is the order of the AR component, d is the order of the integrated component, q is the order of the MA component, and s is the period of the seasonal component. P, D, Q means the same as p, d, q, but for the seasonal order. Finding the best hyperparameters in SARIMA is not easy as they are seven parameters, and each can have several values. To facilitate this, we can plot the Auto Correlation function (ACF) and Partial Auto Correlation function (PACF), which will provide important insights about the values for the p and q components [40]. Posterior to this analysis, to find the optimal combination of parameters, it is also recommended to use a grid search to fine-tune the model and find the best combination of parameters that gives the lowest Akaike Information Criterion (AIC) and BIC Bayesian Information Criterion (BIC) [41]. AIC and BIC are based on the likelihood function and estimate the probability of the observed data given the model. A lower AIC or BIC indicates a better model.

The AIC takes into account the model's complexity and the fit of the model to the data. Consequently, the lowest the AIC value, the best the model is fitting to the data. AIC is defined in Equation 2.3, where k is the number of parameters in the model, and $\widehat{L}$ is the maximum value of the likelihood function for the model.

$$AIC = 2k - 2\log(\widehat{L}) \tag{2.3}$$

The BIC is much closer to the AIC as it also considers the model's complexity and how well the model fits data but has a more significant penalty term. BIC can be formally defined by Equation 2.4, where k is the number of parameters in the model, n is the number of samples, and $\widehat{L}$ is the maximum value of the likelihood function for the model. Observing the Equations of AIC and BIC, it is possible to notice that the only difference is in the first term. While in AIC, this only depends on k, in BIC, it also depends on n. As n has to be positive, as the number of n, the $\ln(n)$ term also increases.

$$BIC = \ln(n)k - 2\log(\widehat{L}) \tag{2.4}$$

Long-short Term Memory

Long-short Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) widely used for modeling sequential data, such as time series, natural language, and speech. LSTM was proposed by Sepp Hochreiter and Jürgen Schmidhuber [42] to overcome some problems of RNN. LSTM has an internal memory that allows them to store and access information over a longer period, which makes LSTM better at dealing with long-term data dependencies. Moreover, LSTM have multiple gate layers to control the information that enters and leaves the cell, enabling the network to recall or forget information selectively. The internal memory and multiple gates are not present in simpler RNN architectures. However, LSTM can be more difficult to train than RNN as it is more complex, but it brings advantages mainly in tasks that require the model to remember and use long-term dependencies.

The architecture of an LSTM network can be seen as a series of cells that are connected sequentially (like RNN), i.e., each cell takes an input and produces an output, being that the state of the previous cell is passed to the next one in the sequence. Internally, to regulate the flow of information into and out of the cell, each LSTM cell has an input gate layer, an output gate layer, and a forget gate layer.

The forget gate layer aims to choose which information from the cell state should be discarded. For this, a sigmoid function gets input and a hidden state (i.e., a vector that stores information from the current and previous time steps) from the previous cell and output values between 0 and 1, where 0 means "completely get rid of this" and 1 means "completely keep this" [43]. An advantage of this is that the values will be between 0 and 1, which will gradually reset the states and not immediately reset to 0.

The input gate layer aims to control how much new information will be stored in the cell. It gets an input and a hidden state from the previous cell. It decides which information to keep by multiplying the output of a sigmoid function with a hyperbolic tangent function. The sigmoid function will output values between 0 and 1, where 0 means low importance and 1 means high importance. The hyperbolic tangent function will guarantee that the values will remain between -1 and 1, preventing some values from rising to a point where the others appear insignificant.

The last layer is the output gate layer, which is responsible for choosing which information leaves the cell. This decision is based on the multiplication of a hyperbolic tangent function with the actual cell state with a sigmoid function that decides which parts of the cell state will be returned. The cell's output is determined by the current input and the information stored in the internal memory. Moreover, this output is used to modify the internal memory and generate the output for the subsequent time step.

Transformer

The Transformer model is a neural network model introduced in [44] by Vaswani *et al.* It is designed to process sequential data and is an encoder-decoder architecture. The

encoder maps a sequence of symbol representations (input) to a sequence of continuous representations. The decoder will generate an output that transforms this sequence of continuous representations into a new sequence of symbols, generating one symbol at a time. The model uses the previously generated symbols as additional input to create the following ones at each phase. One of the key innovations of the Transformer model is the replacement of the commonly used recurrent layers in encoder-decoder architectures by a self-attention mechanism. This allows the model to focus and give more importance to relevant parts of the input, making it possible to capture long-range relationships in the input sequence more effectively. Another innovation in the Transformer model is the multi-headed attention feature, which allows the model to attend to multiple parts of the input sequence simultaneously. Overall, the Transformer model has proven to be very effective for a variety of NLP tasks (machine translation, language modeling, and summary) and has also been used in other domains, such as computer vision and speech recognition [45].

The architecture of a Transformer can be described as a stack of fully connected self-attention and point-wise layers, either in the encoder or decoder part. Each encoder is composed of a stack of identical layers. Each layer has two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. The multi-head self-attention sub-layer is responsible for computing the relationships between different elements of the input sequence using self-attention mechanisms. It takes the input sequence and produces an output representation that encodes the relationships between the elements. The position-wise fully connected feed-forward network is a fully connected neural network that processes the output of the self-attention sub-layer and produces a final output representation for the input sequence. Together, the sub-layers of the encoder can capture complex relationships between elements in the input sequence of symbols and produce a compact representation of it.

The decoder part is very similar to the encoder and comprises a stack of identical layers. However, in addition to the two sub-layers presented in the encoder, the decoder has an additional one that will perform multi-head attention over the encoder output. Moreover, to guarantee that the predictions for a given position in the output sequence only depend on the previously generated output, in the decoder stack, the self-attention sub-layers are modified to prevent positions from attending to positions that come after them. Additionally, the output embeddings are shifted from one position to further.

Temporal Fusion Transformer

The Temporal Fusion Transformer (TFT) model [46] is a variant of the transformer model designed to improve interpretability and performance in several tasks. Like the Transformer, the TFT is a powerful sequence-to-sequence model that uses self-attention mechanisms. However, TFT implements a new type of self-attention layers as they are interpretable. Besides interpretability, TFT also supports more types of covariates compared

to Transformer (Table 2.4) and multi-horizon forecasting that consists of the generation of multiple predictions for different time horizons.

The architecture of TFT model is shown in Figure 2.3 and is based on a central component proposed by the authors, named Gated Residual Network (GRN). This component was motivated by the difficulty that models have in anticipating the required non-linear processing and tries to help the network understand which inputs are simple and require additional modeling. GRN takes a primary input, and an optional context vector and uses two dense layers and two activation layers (Exponential Linear Unit (ELU) [47] and Gated Linear Units (GLU) [48]), and the output of this functions will be passed through a normalization layer. However, GRN also have a residual connection that allows the network to skip some inputs entirely.

The TFT model starts by selecting the essential variables through a network that uses GRN components and a softmax function to give weights to the features. These components will allow the model to classify features as useful or noisy. In the next step, TFT uses LSTM Encoder Decoder Layers to produce context-aware embeddings. The encoder receives the known inputs (past covariates), and the decoder receives the unknowable future inputs (future covariates). Regarding the static covariates, the model uses specific encoders that encode the static context vectors to condition temporal dynamics.

The interpretable self-attention mechanisms are applied in the last part of the TFT model. These mechanisms explicitly encode the order of the input and output sequences, making it easier to understand how the model uses the input sequence to generate the output sequence, allowing users to check the variable's importance, discover the persistent temporal patterns, and identify the significant events.

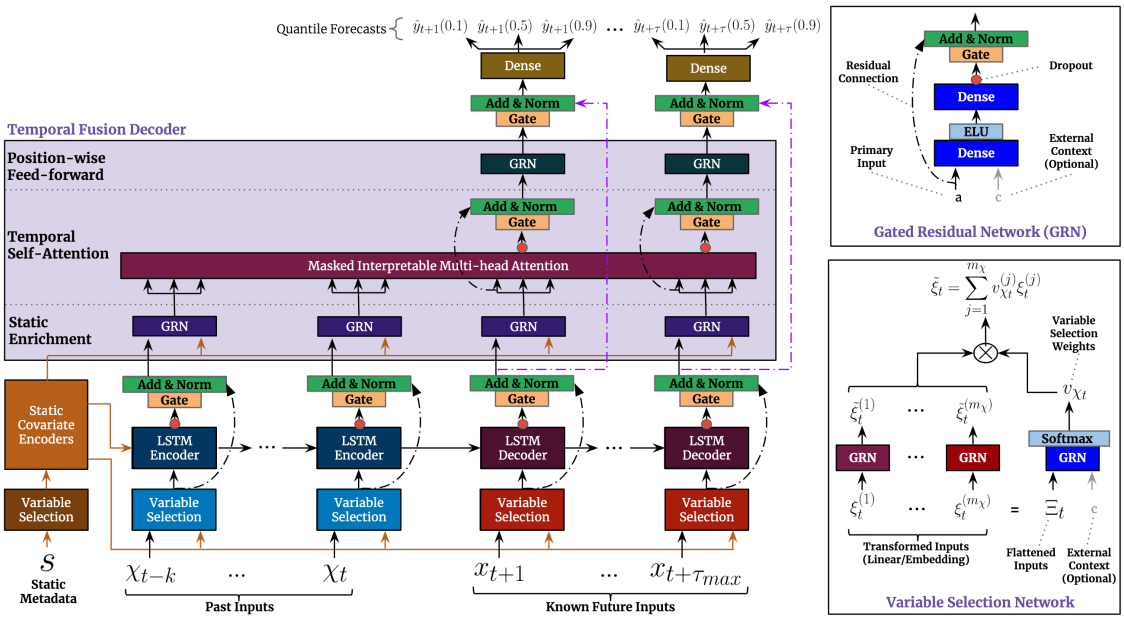


Figure 2.3: Temporal Fusion Transformer model architecture [46].

2.3 Interpretable methods for machine learning models

2.3.1 Why to provide explanations

It is essential to explain predictions more than just present predictions because, in many situations, having a correct prediction is not enough to solve the original problem. In essence, an explanation is a "bridge" between people and a decision-maker that allows humans to understand the decisions [49]. For example, in a model that predicts a drug for a patient, it is essential to know why the model predicted that because it is necessary to provide explanations so the patient can trust and accept the decision. Explanations can be made by adding interpretability to our machine-learning problems. The greater the degree of interpretability of a model, the easier it is to understand and accept certain decisions/predictions. Previously, we already had an insight about the TFT model, but in this Section we will talk more deeply about how we can achieve interpretability and some other interpretable models.

However, interpretability is only sometimes needed in all machine problems because models are used in a low-risk environment in some situations, making predictions mistakes not significant. Moreover, interpretability also loses importance when the problem is well studied or if interpretability enables people to manipulate the system.

2.3.2 How to achieve interpretability

Interpretability can be done in several ways, and due to that, we can classify it using different criteria. For example, it is possible to apply interpretability methods that analyze the model after training (post hoc) or by analyzing the machine learning model itself (intrinsic). Intrinsic interpretability relates to models, such as short decision trees, that are considered interpretable due to their basic structure [50].

Besides that, we can interpret machine learning models by analyzing feature input and output pairs after training (model agnostic) or by analyzing internal model parameters and interpretation tools (model specific). Interpretability is also possible to be analyzed in a global way that tries to explain how the model works using features and all the learned components (weights and other parameters), or in a local way that consists of using a single or a group of predictions to analyze the model.

In the end, interpretability can be achieved in several ways. The most important is choosing methods that provide reasonable explanations for the problem being studied.

In Table 2.5, we present some interpretable models with their main characteristics. To better understand this table, a model is considered linear if the relationships between features are modeled linearly. A model is monotone if an increase in feature values invariably leads to a decrease or to an increase in the target outcome of the model. Interactions refer to models that can automatically detect interactions between features. Finally, different models handle different tasks. Models can handle classification, regression, or both

of these tasks. However, in the end, the most important thing is to choose models that suit the project objective.

Table 2.5: Comparison of different interpretable models. (Table from [50])

Algorithm	**Linear**	**Monotone**	**Interaction**	**Task**
Linear regression	Yes	Yes	No	Regression
Logistic regression	No	Yes	No	Classification
Decision trees	No	Some	Yes	Regression, Classification
RuleFit	Yes	No	Yes	Regression, Classification

2.3.3 RuleFit

For this work, the interpretable model chosen was RuleFit, developed by Friedman and Popescu [51], and allows finding feature interactions automatically, maintaining interpretability simple. It is a tree ensemble algorithm and supports regression and classification tasks, and one of the significant advantages of this algorithm is that it is intrinsically interpretable. The idea behind RuleFit is to create a sparse linear model based on the original features and new features created by the model where interactions in the data are captured. These new features are called rules, which are binary *IF-THEN* statements based on the dataset features. An example of a rule can be seen next:

IF cpu_utilization < 50 *AND* number_negative_tweets > 20
THEN 1 ELSE 0

As RuleFit is a tree ensemble algorithm, these rules are generated from multiple decision trees where each tree can generate many rules, as is depicted in Figure 2.4. A simple tree with three leaf nodes can generate four rules (identified with r_1, r_2, r_3 and r_4), meaning that every path generates a new rule, not only the paths ending in the leaf nodes. Each rule is binary, meaning it will return 1 for points that meet all conditions and 0 for the others, and the chain of each binary rule that comes from each tree node can be done by adding a AND operator. Along the trees generated by RuleFit, the rules can vary from simple to more complex ones (a lot of *AND* in the expression). Extensive trees can generate more complex rules and capture more interactions among the dataset variables but can also increase the difficulty of interpreting. Ultimately, the rules are straightforward to understand and explain to people unfamiliar with the model or machine learning.

To generate the rules, RuleFit can use any tree ensemble algorithm. However, as the *rulefit* package uses Gradient Boosting, we decided to follow the same strategy. Gradient boosting fits an ensemble of decision trees to predict a target variable Y using a set of

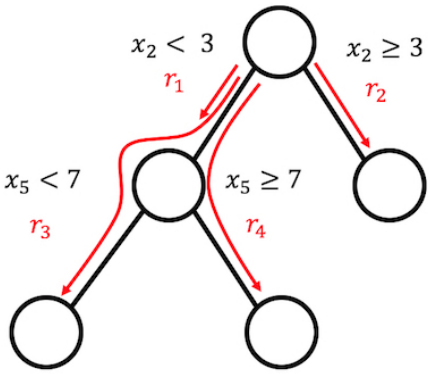


Figure 2.4: One decision tree generated in RuleFit algorithm. [51].

original features X. The basic idea behind gradient boosting is to train weak models iteratively, each trying to correct the mistakes of the previous model. During each iteration, a new model is trained to predict the residual error (difference between the predicted and the true output) of the previous model [52]. The new model is added to the ensemble of models, and the prediction of the ensemble is updated accordingly. This process is repeated until the ensemble reaches a satisfactory performance level or a pre-defined number of iterations is reached. The final prediction is the weighted sum of the predictions of all the models in the ensemble. In the end, RuleFit will create the rules from all ensemble trees.

This previous process generated many different rules, creating new features (each rule becomes a new feature). Although, some of the rules may not be informative and should be discarded. In the second step of the RuleFit, a sparse linear model will be trained using a $LASSO$ penalization that will have the function to force some rule weights to zero. This process will generate the weights for all rules.

Mathematically, RuleFit can be described following the Equation 2.5, where $\hat{f}(x)$ is the prediction function, K and p represents the number of rules and the number of features, respectively. Finally, $\hat{\alpha}$ and $\hat{\beta}$ are the weight vector for the rules and original features, respectively.

$$\hat{f}(x) = \hat{\beta}_0 + \sum_{k=1}^{K} \hat{\alpha}_k r_k(x) + \sum_{j=1}^{p} \hat{\beta}_j l_j(x_j) \tag{2.5}$$

The weights of each of the RuleFit features ($\hat{\alpha}$ and $\hat{\beta}$) can be described following Equation 2.6. The first term of this equation represents the loss function that can be any (L), and the second term corresponds to the $LASSO$ regression penalization.

$$(\{\hat{\alpha}\}_1^K, \{\hat{\beta}\}_0^p) = \quad argmin_{\{\hat{\alpha}\}_1^K, \{\hat{\beta}\}_0^p} \sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)})) + \lambda \cdot \left(\sum_{k=1}^{K} |\alpha_k| + \sum_{j=1}^{p} |\beta_j| \right) \tag{2.6}$$

Interpretation in this algorithm is simple, as we can observe the Rules by order of importance (how much a feature contributes to the prediction) and support (how many

points belongs to a rule). Support is an essential feature as it can explain some interactions in the dataset that were not detected by the naked eye.

2.3.4 Drift detection

Interpretability in machine learning is directly associated with explaining the predictions, but there are other ways to bring explanations to our models. One of them is drifts that aim to sound an alarm when changes in the data are detected. These alarms can warn the user or indicate that the model needs to be retrained/adjusted [53].

In machine learning, there are different types of drifts, such as concept drifts, covariates shifts, or prior probability shifts [54]. For this work, we will mainly focus on concept drifts, which is a drift that occurs when the relationships between the input and output variables change. Regarding concept drifts, they might appear in different ways, such as incrementally (where the data distribution is changing slowly), abruptly (where the data distribution changes suddenly), or in a recurring/systematic way (where the data changes with some frequency) [55]. Concerning our work, in cases where the workload trend suffers a change, a concept drift would trigger.

Several approaches for detecting concept drifts have been proposed. For this work, we selected two methods named Adaptive Windowing (ADWIN), and Kolmogorov-Smirnov Windowing method (KSWIN).

ADWIN is a method that divides data into overlapping windows and monitors the performance in each window. If the data distribution decreases significantly in some windows, it may indicate that a drift occurred. ADWIN uses a significant value δ that can be defined manually to check if two sub-windows correspond to the same distribution.

KSWIN is a concept drift detection method based on the Kolmogorov-Smirnov (KS) statistical test. KS is used to compare the distribution of a part of data to a reference distribution. It is based on the maximum difference between some sample empirical Cumulative distribution function (CDF) and the reference CDF. The user can define this difference through a parameter α. CDF contains the probability of a random variable X, with a given probability distribution, being found at a value less than or equal to x [56].

3

Novobanco data center

This Chapter aims to explain the dataset of novobanco data center used in this thesis. We start by presenting an initial overview of this dataset, how it was obtained and its format, followed by an exploratory data analysis of the dataset feature.

3.1 Data Overview

Recalling the diagram 2.1 explained in Chapter 2, we could observe that there are a lot of servers and channels in *novobanco* data centers. As it is difficult to investigate all servers individually, we had to restrict our research to some servers with significant importance for the problem we are dealing with. Consequently, we continued our investigation with a set of servers located in Taguspark and used as direct channels. It was decided not to use data from the Alfragide data center, as this is not the main one and is just used as a backup. Figure 3.1 highlights the localization of the servers used. These servers were chosen because they are used for processing user requests (bank balance, credits, transfers, and others) and are the ones with major importance in the bank. Recalling the structure of this data center, there are four main layers. The first layer (PL) deals mainly with the front-end part, the second layer (BL) processes most of the operations needed, the third layer (SEI) processes mainly enterprise operations, and the last layer is the database layer. Considering this and as will be further justified, we will use the first and second layers as these are the ones that can be optimized and the ones that process the most important information. Contrarily, the third layer does not process so many processes, and the last layer is the databases, and it is not correct to turn on and turn off databases as this could create problems related to database reliability and consistency.

During the following sections, we will first present how we extract the server's metrics from the *novobanco* data center, followed by an Exploratory data analysis (EDA), overall metrics. As CPU is one of the most critical metrics for the objective of our work, it will have a more considerable relevance in this Chapter.

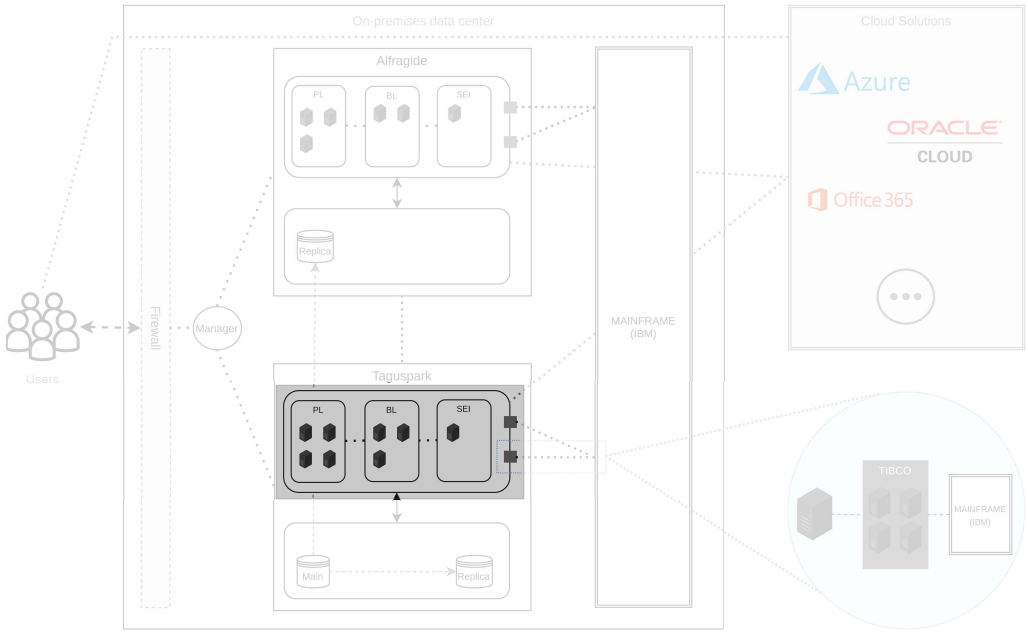


Figure 3.1: *Novobanco* DC structure with highlight in the servers used in this work.

3.2 Direct channels dataset

3.2.1 Data source, format, and pre-processing steps

The metrics monitoring mechanism implemented in *novobanco* servers is `Nagios`[1]. `Nagios` is a monitoring application of all mission-critical infrastructure elements such as services, operating systems, network protocols, systems metrics, and network infrastructure. It is a system developed in `Nagios` but can run in other `Unix-like` environments. In the specific case of server monitoring in *novobanco*, each server has a file for each metric (CPU, RAM, read and write operations, and others) and can be exported in an RRD file format.

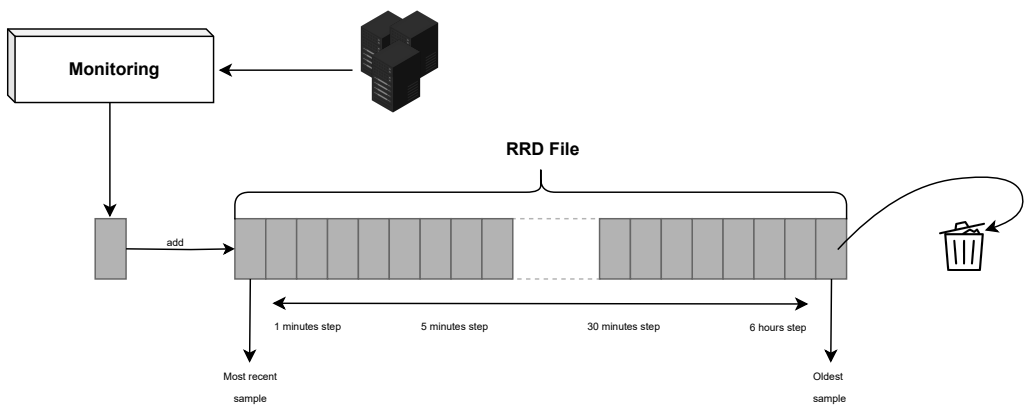


Figure 3.2: RRD file schema.

[1] https://www.nagios.org/

RRD is a format commonly used to handle time series data related to metrics monitoring, and the data is kept in a database with circular buffers. Thus, the time range is constant over time (in the case of *novobanco*, the last record is from 1460 days ago, counting at the time it is observed). Figure 3.2 illustrate how data is stored in this kind of files. As can be observed, there is a monitoring node that assumes a time-variable range called step (in case of *novobanco* this step is of one minute), and in each step, it saves the metric value and erases the last sample in the RRD file. However, despite adding a new sample to the file every minute, the step is not one minute in all of RRD file. Instead, as the time goes far away, the step increases firstly to 5 minutes, after 30 minutes, and finally to 6 hours. In sum, a new sample is added every minute. As time goes away, it starts to compress samples into more extensive time ranges (note that these intervals can be changed in the setup phase, and the available time ranges and the number of records available can be defined manually). Table 3.1 Shows the number of days available for every time range. As observed, the number of available days increases as the range increases. For the rest of this work, we will restrict to the 30 minutes and 6 hours ranges as the 1 and 5 minutes ranges do not have sufficient days to observe trends and seasonality, and so on to have power models. Also, the day the data were taken was 9 August 2022, so the beginning range of days will vary depending on the time step in observation. For example, for the time step of 30 minutes, we will have records from 11 May to 9 August 2022, but for the time step of 6 hours, we will have records from 10 August 2018 to 9 August 2022.

Table 3.1: Dataset overview ranges (Data was taken on 9 August 2022).

Time Step	Number of days	Starting day	Number of records
1 minute	Last 2 days	07-08-2022	2880
5 minute	Last 10 days	30-07-2022	2880
30 minute	Last 90 days	11-05-2022	4320
6 hours	Last 1460 days	10-08-2018	5840

The problem with files in RRD format is are not human readable and cannot be understood with standard and commonly used editors. Because of this, we had to make use of `rrdtool`[2], a tool made for handling, managing, and visualizing time-series RRD files over `Linux`, and was later adapted for the PHP and `Node.js` languages. Over the different methods that `rrdtool` provides, we made use of `dump` that convert the content of an RRD file into a human-readable XML format. With the files in XML format, we create a python script that reads and organized the content of each file in an csv file to be easy to work and explore in the subsequent phases.

[2] https://oss.oetiker.ch/rrdtool/index.en.html

3.2.2 Exploratory data analysis (EDA)

Several metrics are being monitored for each server, and as we deal with multiple servers, we have multiple files to organize and study. Table 3.2 does an overview of the dataset, and in Table A.3 is provided a more detailed description of features, their type, and which features were discarded (with the respective reason in Table A.4). Note that for all the graphs presented during this section, the data has a time step of 6 hours, except for the one involving the hours (for that, it was used a step of 30 minutes).

Table 3.2: Data center dataset overview.

Number of servers	Number of layers	Number of metrics per server (processed data)	Obtained on
21	4	29 (11)	9 August 2022

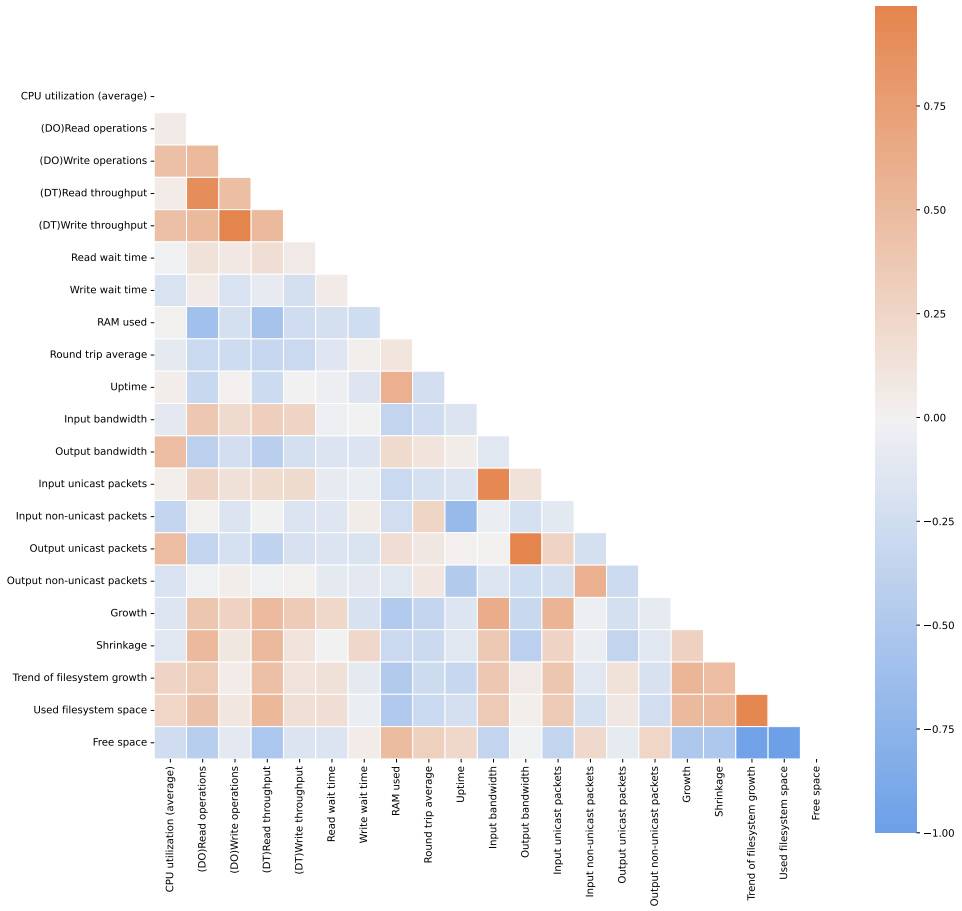


Figure 3.3: Heatmap with correlated features degree.

Observing that the amount of features available is high, we started by exploring correlations between features to reduce the feature space. For this, we calculate the correlation of all features available for every server individually and calculate the average of individual servers to understand the correlations common to all servers (note that the features available in each server are the same). Figure 3.3 shows the main positive and negative

correlated features. Observing and studying this graph, we decided to discard features with a correlation bigger or smaller than 0,65. As they are correlated, these features will not bring additional information to our study and will increase the complexity of this study. Diagram of Figure 3.4 shows the features that correlate positively (in red) and negatively (in blue) with a correlation of more or less than 0.7, respectively. Also, from the correlated features, the features that were maintained are highlighted. From the 16 correlated features, ten were discarded, reducing the total number of features from 21 to 11.

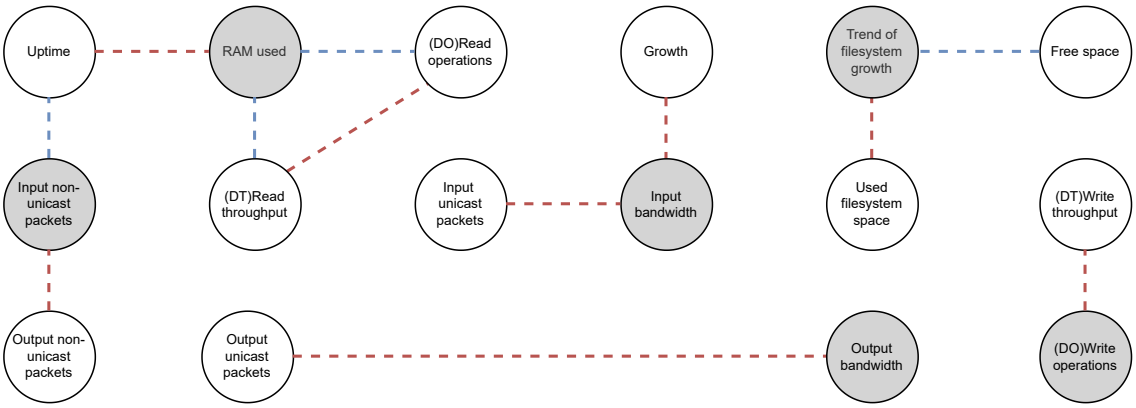
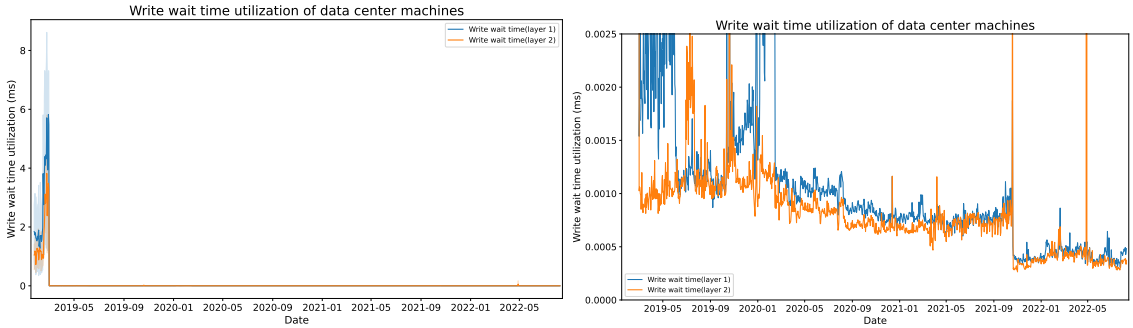


Figure 3.4: Correlated features diagram. (Red corresponds to features positively correlated, and blue corresponds to features negatively correlated)

Moreover, we decided to do a more internal analysis of the remaining features. With the remaining ten features, we investigated if there was missing data. Regarding the first layer, there was no missing data in any server concerning the 30 minutes and 6 hours step data. Regarding the second layer, in the 30 minutes step data, there were no missing values in any server feature. In contrast, in the 6 hours step data, we found that between 10 August 2018 and 23 January 2019, in six of the ten features maintained, there were more than 80% of null values in almost all servers. This suggests that during that time, there was a failure in monitoring data, or simply the monitoring was turned off. Because of this, and to have matching records in all 6 hours step data of both layers, we discarded all the values between 10 August 2018 and 23 January 2019 of both layers, which resulted in the loss of 560 rows. After this, we did some basic statistics and visualizations to understand the dispersion of these features. With this, we notice that features "Write operations", "Read wait time", and "Write wait time"were producing strange values as they are significantly changing in the beginning and stabilize near zero since 2019-02-28 (an example of this is presented in Figure 3.5(a)). Studying these events internally, we notice that metrics have such small values from a certain point that we cannot notice the changes. If we observed with a small y-axis range, we could notice that the values are not exactly zero and vary over time (Figure 3.5(b)). This fact happened on all servers and is related to servers' updates.



(a) Write wait time metric with normal range. (b) Write wait time metric with a small y-axis range.

Figure 3.5: Write wait time metric over time.

The following analysis consisted in analyzing the servers through the different layers of *novobanco* data center. Figure 3.6 shows this analysis. As we can see, the CPU utilization varies for different layers, which indicates that each layer should be studied individually to have better and more accurate results. Moreover, it is possible to observe that in the first and second layers, there is a server that, in the major part of the time very near to 0. These two servers act as backup servers in case one server fails or the requests increase significantly, and the other servers cannot handle that amount of requests. The third layer is composed of just one server, so optimizing this layer is impossible. For the database layer, there is the primary database, and the other two act as a backup of the primary database, so the number of operations made in the ones that are not the primary will be lower. With this analysis, we decided for the subsequent phases to focus mainly on analyzing the first and second layers as these are the ones with more interest to study.

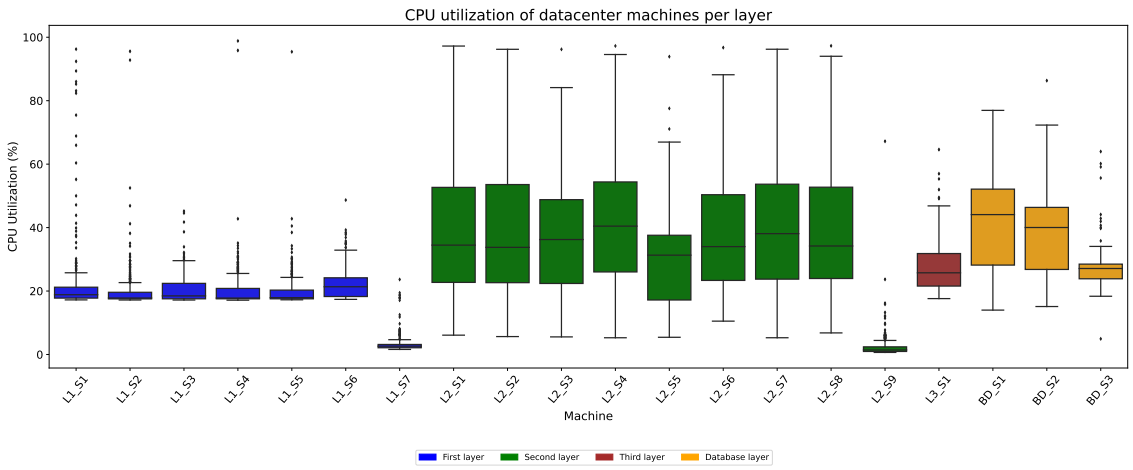
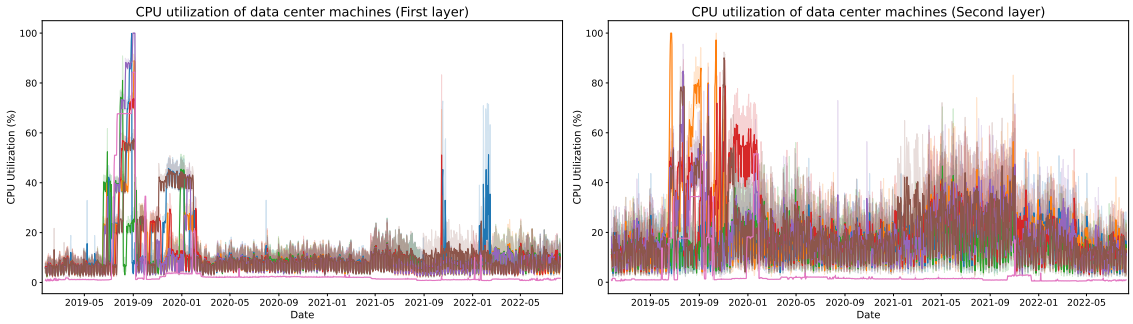


Figure 3.6: CPU Average over servers and layers.

Analyzing the CPU utilization over time of the first layer and second layer (Figure 3.7), the difference in patterns can be noticed. As the first layer is mainly used for the front end, the processing in this layer is not too big, but instead, in the second layer, where

many operations are processed, we could notice a very high CPU usage. Also, as expected from the Figure 3.6 analysis, we have a server with almost no usage and just an increase in cases where the usage increases in other servers. It is also interesting to observe that in the first layer (3.7(a)), there was an increase at the end of 2021 and the beginning of 2022 in one of the servers, and around April 2022, three servers had a peak of utilization. Analyzing this graph together with the previous one, and knowing that there is a server that acts as a backup in the first and second layer, we decided to discard these two servers (L1_S7 and L2_S9) of optimization because independent of any optimization that we want to do, knowing that we are dealing with direct channels of a bank, it will always be necessary to have a server to act immediately in case of any failure.



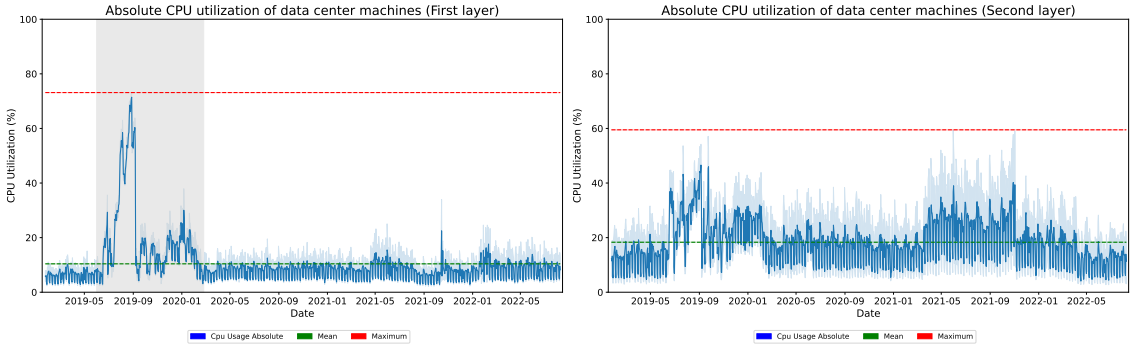
(a) CPU utilization of the servers of the first layer. (b) CPU utilization of the servers of the second layer.

Figure 3.7: Cpu usage layers

The following observation (present in Figure 3.8) was to analyze the CPU utilization in absolute terms of the first and second layer (excluding the servers that serve as backup). With these graphs, we can observe that in the first layer, the CPU utilization never was superior to 53%, and the mean value was around 21%, while in the second layer, it arrived at values near 98% with a mean of around 40%. Also, Figure 3.8(a) emphasizes the peak of April 2022 as the bigger one of the two noticed in the previous Figure (3.7), and this peak also corresponds to the time the CPU has the more prominent absolute peak in the second layer. The final remarks of this graph reinforce that while there is not a considerable variance in the first layer and the CPU used is not very high, the second layer is the opposite.

With a better understanding of the general usage and variance of servers of each layer, we decided to investigate trends related to the hour of the day, the month, or the day of the week. Behind this is that users will probably have time zones that use more frequently and other time zones that use more infrequent applications of *novobanco*. In the graphs that will be explained next, each line corresponds to a trend of an individual server, as in charged dark blue color, it is presented the mean of all servers.

Starting by observing the CPU utilization over the month days (Figure 3.9), we can see that in both layers, there is a slight increase in the beginning and ending days of

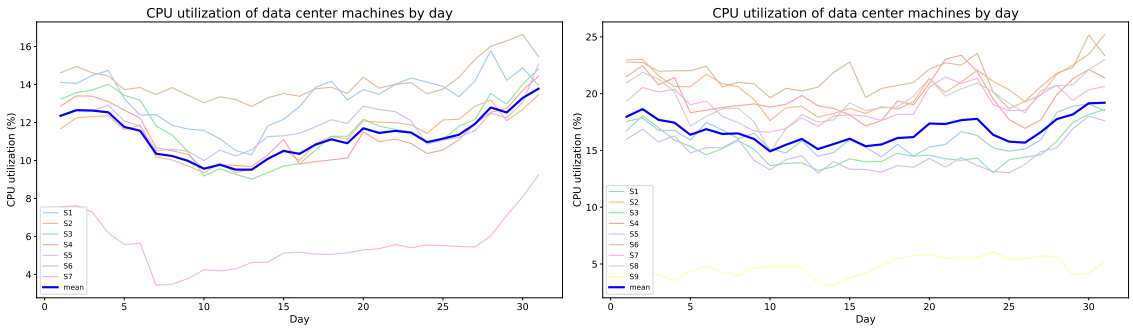


(a) Absolute CPU utilization of the first layer.

(b) Absolute CPU utilization of the second layer.

Figure 3.8: Absolute CPU utilization of servers (excluding the ones that serve as backup).

the month. This can be related to days when people receive their salary (in Portugal is generally at the end of each month), have to pail their house bills, or organize their finances for the next month.



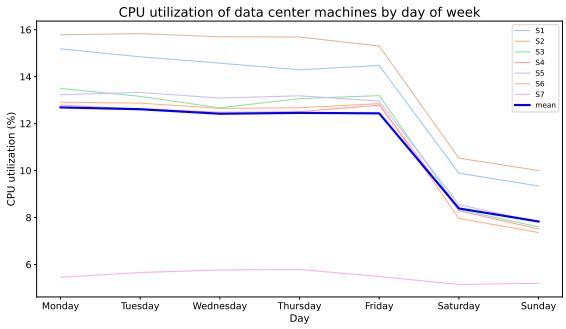
(a) CPU utilization of the first layer by day of the month.

(b) CPU utilization of the second layer by day of the month.

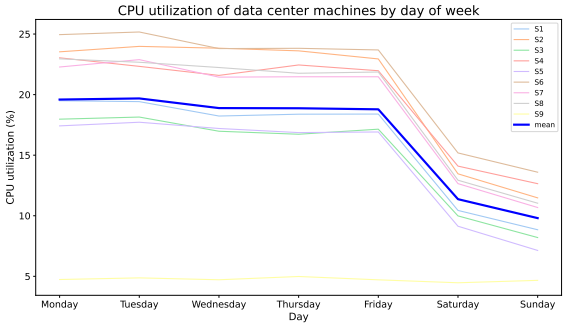
Figure 3.9: CPU utilization of data center machines by day of month

Regarding CPU utilization by day of the week (Figure 3.10), we could notice that in both layers, there is constant CPU usage during business days. Contrarily, on the weekends, there is a significant decrease in CPU utilization. This was a curious observation as we expect that these servers' usage could increase as some people like to organize their economies on weekends, but these graphs did not show this.

After analyzing the CPU utilization by day of the week, we analyzed the CPU utilization by the hours of the day in the first and second layer, depicted in Figure 3.11(a), 3.11(b) respectively. Analyzing these graphs, we could analyze some clear trends in data. Regarding both graphs, it is possible to observe a growth in usage in the morning and starts to decrease in the late afternoon. During this time, there is a slight decrease in usage around 1-2 pm, which may be correlated with the usual lunchtime. Moreover, in the first layer, two curious peaks happen over all servers at daybreak (around 3 and 7 am),



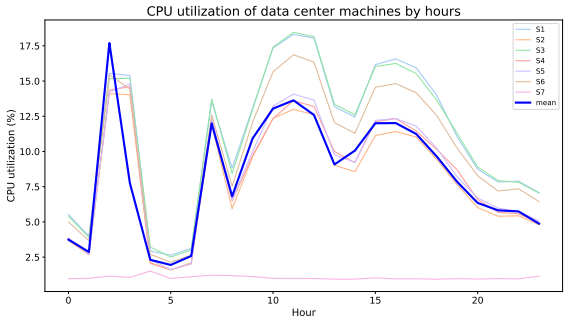
(a) CPU utilization of the first layer by day of the week.



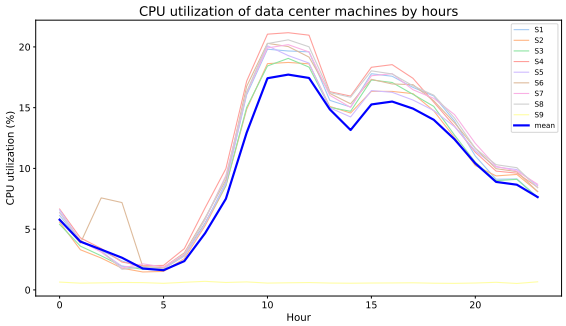
(b) CPU utilization of the second layer by day of the week.

Figure 3.10: CPU utilization of data center machines by day of week

and we did not find any apparent reason.



(a) CPU utilization of the first layer by hours.

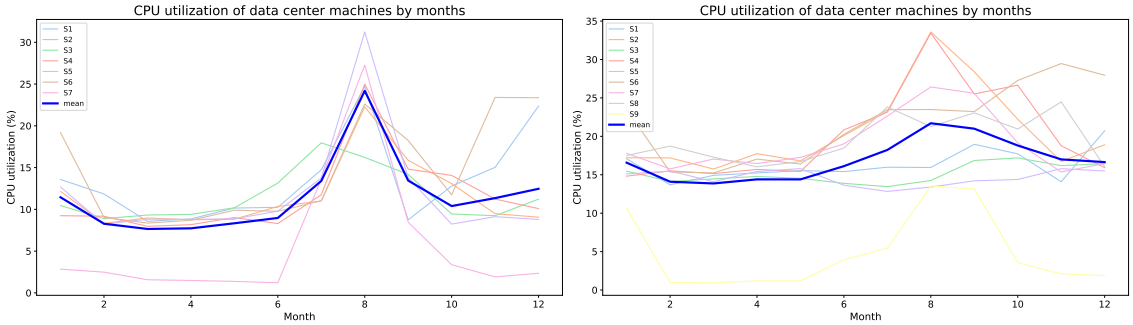


(b) CPU utilization of the second layer by hours.

Figure 3.11: CPU utilization of data center machines by hours

Some interesting trends happen regarding CPU utilization by month (Figure 3.12). It can be observed that August is where there is a peak in both layers, which can be explained because it is most people's vacation month, so more operations and bank balance inquiries are made. Moreover, a peak could be observed in the final and beginning months of the year (December-January). This probably happens because of the festive seasons like Christmas and the new year, and it is known that this season involves a considerable cost of money.

The last analysis we provide in this Chapter is between CPU and RAM because they can be seen as the two most essential components of a computer. Figure 3.13 presents this relation, where blue points are regarding the first layer and orange points to the second layer. We can observe that no strong correlations exist, as RAM has a range of points, and for every CPU value, RAM values are strict to that range. Moreover, We have a large set of points when the CPU value is less than 20%, but as the CPU increases, the number of points decreases for both layers. It is also interesting to note that in the first layer, the region with more significant values for RAM is the zone with lower CPU values, meaning



(a) CPU utilization of first layer by months.

(b) CPU utilization of second layer by months.

Figure 3.12: CPU utilization of data center machines by months

that the first layer did some operations that did not require many processing resources but many memory resources.

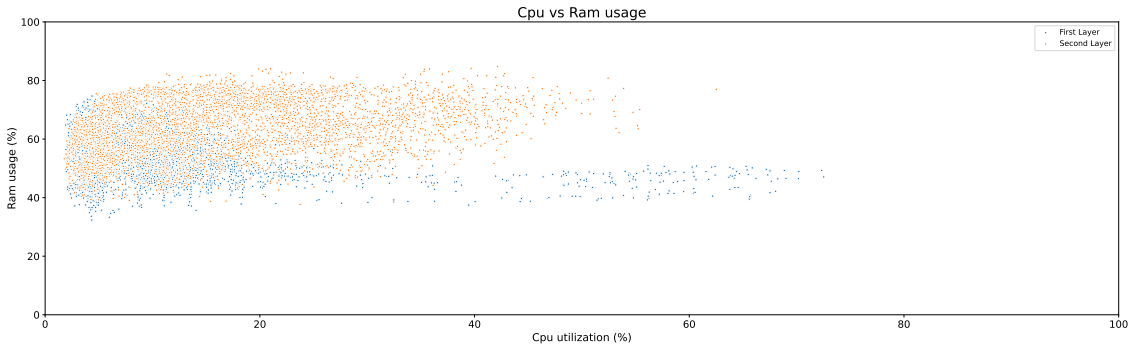


Figure 3.13: CPU utilization vs. Ram utilization.

In short, during the analysis in this section, we strict a lot to CPU usage despite having many other features available. We did this because the CPU is the most crucial resource to predict and is the most related to server usage. With this analysis, we first concluded to discard optimization in the third and database layers as in the case of the third layer, there is only a server, so it cannot be optimized. In the case of the database layer, it is also tricky because replicas need to be synchronized. Turning on and off some databases is not viable because it would create data consistency problems. That is why we restrict to the first and second layers, and in both these layers, we also observe that each layer has a server with a meager CPU consumption that serves only for emergency cases (as a backup). It is required to have these servers always there in case of an unexpected failure of some other server. To complement this analysis, in Annex II, we present some other Figures to help understand other relevant metrics.

4

Twitter data

This chapter aims to explain the Twitter dataset used in this thesis to understand the relation of tweets and novobanco data center resources utilization. We present how we obtained this data, describing and analyzing it.

4.1 Why is Twitter important

In today's society, social media started to be part of regular daily routine tasks. It is a valuable communication tool that can reach everyone, regardless of where people are. There are different kinds of social media, but they are generally used to interact, have fun, and access news and information [57].

Twitter is a real-time platform where people can freely express their opinions and ideas. Besides that, social media channels publish all the news on this platform to reach people quickly and easily. Data from Twitter is used to conduct several studies because Twitter is used by more than 320 million active users, which provides enough data on almost any significant trend or shift (for example, rise and fall of stock market prices) [58].

4.2 Tweets Extraction

Twitter's Developer Platform enables programmatic access to Twitter through *Twitter API*[1]. Through a set of programmatic endpoints, *Twitter API* allows developers to find and retrieve tweets, user information, trends, and so on.

To use *Twitter API* we created a Python program that accesses this API using authentication keys and tokens obtained from an *Twitter API* application. With these keys and tokens, header access to the API is created. The request is built through a function that receives the header and creates a request with the query we want to get and the remaining query parameters. *Twitter API* just allow making 300 requests per 15-minute window,

[1] https://developer.twitter.com/en/docs/twitter-api

and in each request, it is just possible to get 100 tweets. To deal with this limit, we insert a loop in our program to keep making requests until there are no more tweets to retrieve but using the Python function `sleep()` to delay the time between requests. *Twitter API* returns tweets in `JSON` format, that we convert to `CSV` format after search. This results in a `CSV` file with 17 features (detailed in Table A.2).

4.3 Tweets from all media channels

With the Python program prepared, we created a list with the leading Portuguese media accounts to extract tweets from those accounts. We defined the search since the beginning of the year 2014 because *novobanco* was created in that year, and in this way, we could investigate tweets since the creation of the bank. We collected data from 16 media sources, which resulted in 3 175 369 different tweets. Table A.1 list the media sources used presenting some characteristics of them (number of publications and number of followers).

4.3.1 Data preprocessing and cleaning

In this section, we will present the preprocessing and cleaning process to ensure that the dataset is well prepared to be analyzed. We started by checking if there were duplicate tweets in the dataset. We used the unique id that each tweet has, which resulted in zero repeated tweets. We explored data using features individually to understand whether errors, missing values, or special values exist. We also did some basic statistics, using some `pandas`[2] functions like `value_counts()`, `info()`, `unique()`, `describe()`, and performed calculations such as variance, mean, and standard deviation to correctly understand and interpret each feature. We conclude that the data in this dataset was well structured, and all empty/missing camps were filled with `NaN` values.

We noticed that the feature with the date was in `String` format, so we transformed this feature into a python `DateTime`[3] object, to make it easier to manipulate dates and times during Exploratory data analysis (EDA). Analyzing tweets language, we discover that 94,4% of tweets are in Portuguese, 3,3% in Spanish, and 2,3% in other languages, depicted in Figure 4.1. However, when we investigated tweets individually to understand why Portuguese media accounts have tweets in other languages, we found out that some tweets' text has some expression or word, not in Portuguese. In these cases, the automatic algorithm of Twitter that classifies languages is fooled and makes some mistakes. Because of that, we did not exclude the tweets that Twitter did not classify as not being Portuguese.

Some of the features of this dataset were discarded during the data cleaning process. Table A.2 presents which features were discarded explaining the reason in Table A.4 .

[2] https://pandas.pydata.org/docs/index.html
[3] https://docs.python.org/3/library/datetime.html

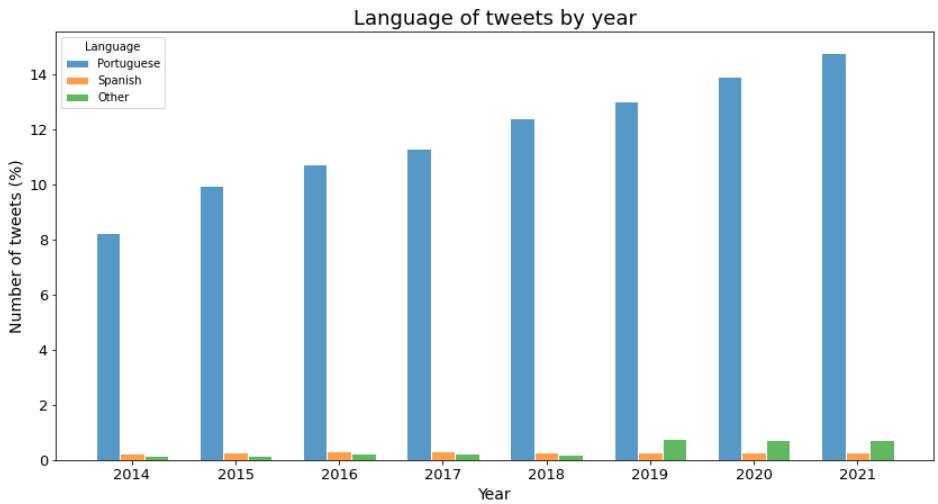


Figure 4.1: Distribution of tweets of the main languages between 2014 and 2021. Data extracted from mass media tweets search.

4.3.2 Exploratory data analysis (EDA)

Exploratory data analysis (EDA) is an approach to analyzing data, where it is possible to discover patterns, identify errors in data, detect and understand patterns in the data, and find interesting relations between variables. EDA uses visualizations and statistical graphics to better and easier understand the dataset. To help us in EDA plots, we made use of `Matplotlib`[4] and `Seaborn`[5] libraries.

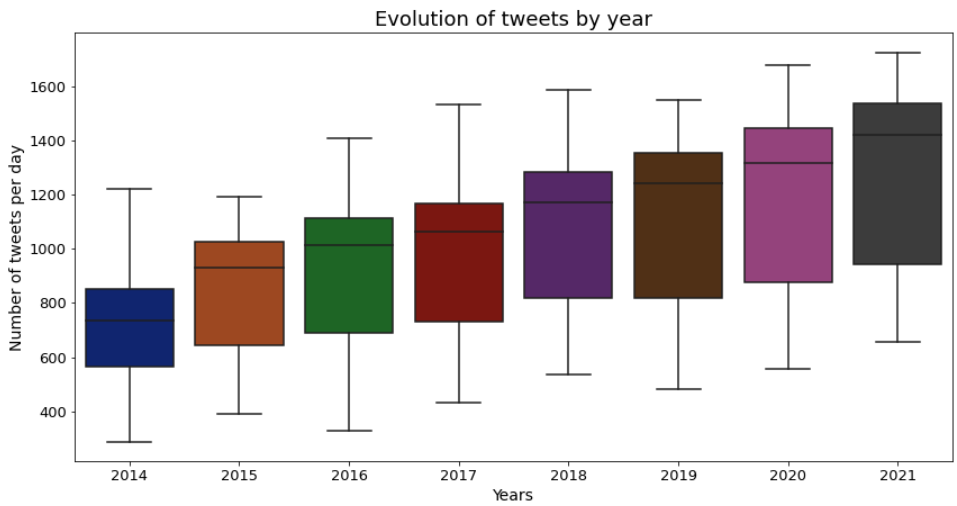
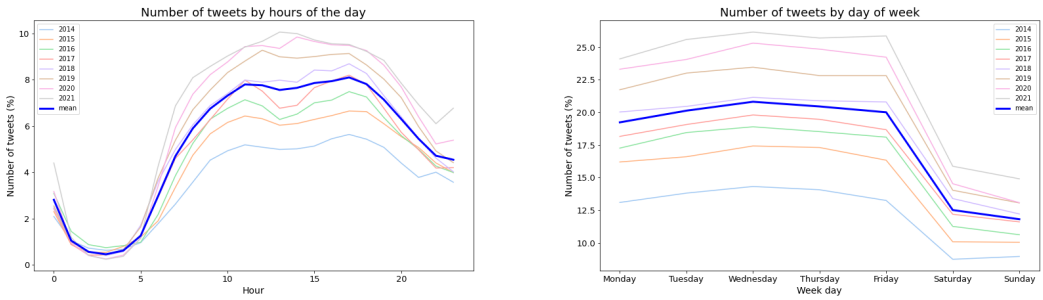


Figure 4.2: Tweets count per year between 2014 and 2021.

[4] https://matplotlib.org/
[5] https://seaborn.pydata.org/

Analyzing the tweets over the years showed that the number of tweets per year has been increasing since 2014, as depicted in Figure 4.2. Besides, we could observe that, on average, the number of tweets per day has also been increasing since 2014.

After analyzing the tweets over the years, we analyzed the evolution of tweets per hour of the day, days of the week, and months of the year, depicted in Figure 4.3(a), 4.3(b) and 4.4 respectively. These graphs allow us to understand if there are any correlations of tweets with the time they are published. In these graphs, we added a line with mean percentage because it helps analyze the graph as a whole, having a better estimation of how the percentage of tweets fluctuates along all years, without being biased by years with more or fewer tweets. These graphs showed us that tweets from mass media accounts are mainly published between 8 am and 7 pm from Monday to Friday. This can be explained because these hours correspond to the Portuguese working hours. Referring to Figure 4.3(a), we can notice that around 1-2 pm exists a slight decrease in the number of tweets released, which can be because of lunchtime. As Twitter has a feature to schedule publications hours, we cannot be sure if the reasons presented are correct, although the probability is high.



(a) Relation of tweets with hours of the day.

(b) Relation of tweets with days of the week.

Figure 4.3: Relation between the number of tweets per year, in percentage.

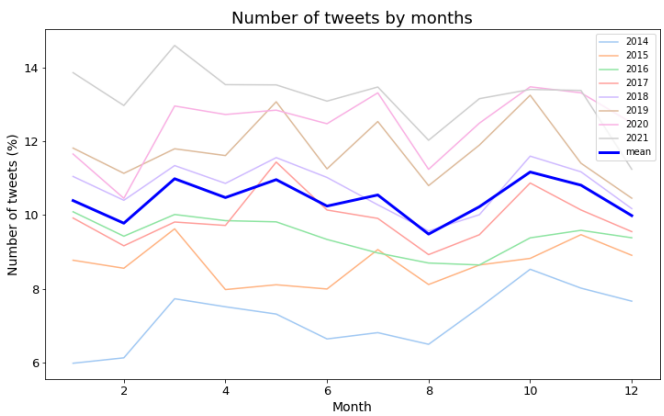


Figure 4.4: Tweets count per year between 2014 and 2021.

Contrarily, concerning Figure 4.4, we conclude that any significant correlations exist between the number of tweets and the month of the year. Just in August, on average, can be noticed a very slight decrease in the number of tweets that can be explained because this is the vacation month in Portugal.

This dataset also has a feature called `public_metrics` that has information about the number of tweet replies, retweets, and likes. Figure 4.5 shows the relation between news retweets and replies, with color change depending on the year. In this graph, we exclude some outliers to understand better the relation of these two variables in the more concentrated zone. By looking at this graph, we can see that both variables do not have a clear correlation. The more concentrated zone shows that most tweets in this dataset have less than 80 replies and less than 180 retweets. Also, by looking at colors, the number of replies increases with the year (tweets from recent years have more replies). For the number of retweets, any clear conclusion can be taken from the color change of this graph.

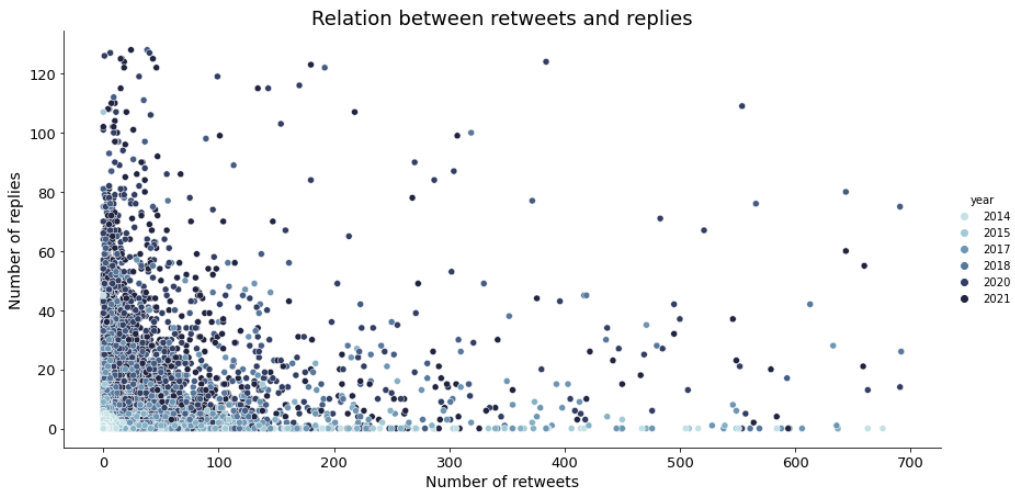


Figure 4.5: Relation between the number of news retweets and replies, grouping by year.

Another relation to be studied is between the number of retweets and the number of likes. Figure 4.6 shows this relation, having a color change depending on the tweet year. As in the previous graph, some outliers were excluded in this one. This graph shows that the oldest tweets have significantly fewer likes than the most recent ones. When the number of retweets increases, it is possible to notice a slight correlation with the number of likes, which means that if the number of retweets increases, the number of likes will increase as well.

Contrarily to what we thought and having in account the Figures 4.5 and 4.6, there is no clear relations between number of likes, retweets and replies. The only small correlation is between the number of tweets and the number of likes. Moreover, to better understand the variations of likes, retweets, and replies along the years, Figure 4.7 presents a graph that has the number of each variable individually along years. This graph support what was said before about the most recent tweets having more likes than the older ones (green line). Another interesting point is the tweet's replies line (orange) that is increasing

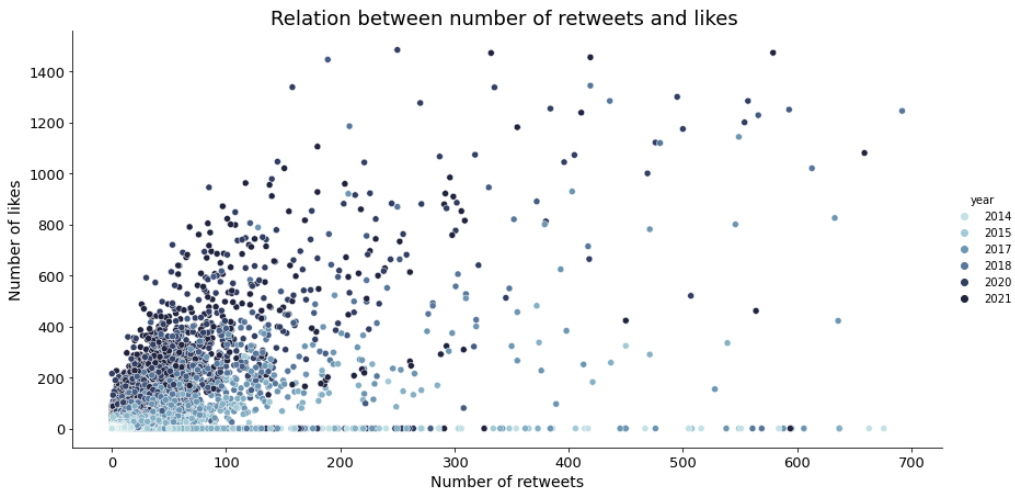


Figure 4.6: Relation between the number of news retweets and likes, grouping by year.

over the years but not so much compared to the increase in the number of likes. The same for the number of retweets (blue line), as this one has some variations in 2014 and 2017. Looking at these three tweets metrics (likes, retweets, and replies), it is clear that the metric gaining more prominence over the years is the tweet's likes. Like is one of the most straightforward metrics used in Twitter, explaining this increase.

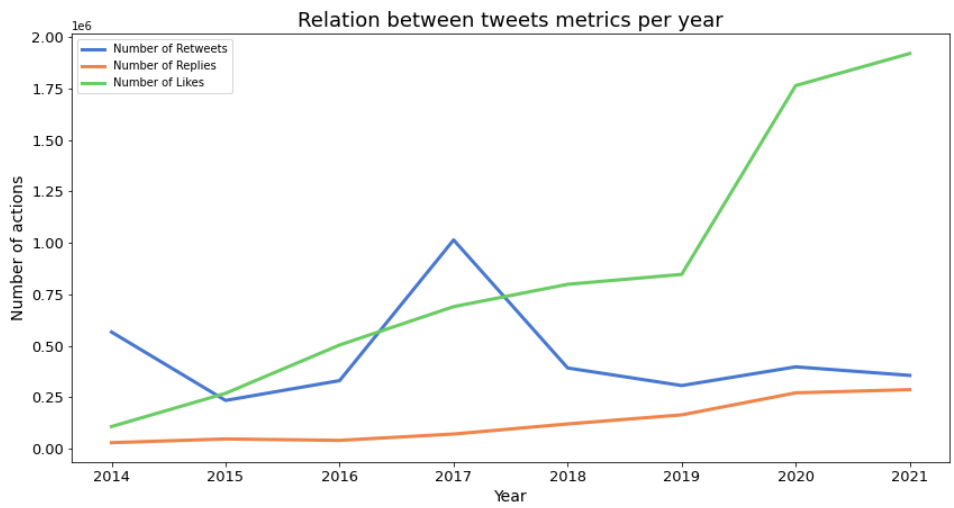


Figure 4.7: Relation between tweet's metrics (likes, retweets, and replies) over the years.

4.4 Tweets related to *novobanco*

It is possible to use different ways to obtain tweets related to *novobanco*. The first one, and one that has more reliable information, is to filter the tweets obtained by the mass media search presented in the last Section and filter tweets based on the content. Another possibility is to directly to a search query through *Twitter API* with keywords related to

novobanco. In these cases, we could select one or more keywords, and the search query returns all tweets that contain at least one of those keywords. Both of these ways will be presented in the following sections, auxiliary by some exploratory data analysis to understand each method better.

For both methods, it is necessary to select the keywords to get *novobanco* related tweets. From our search, we found out that the bank is manily refered in twitter as *"Novo Banco"*, *"novobanco"* and *"novobancopt"*. Since tweets with *"novobancopt"* word are already obtained with *"novobanco"* word, we selected *"Novo Banco"* and *"novobanco"* as our final search keywords.

4.4.1 Filtering Tweets from mass media search

Our first objective is to filter the data from mass media search presented in the last Section (4.3) to get *novobanco* related tweets, based on the keywords mentioned above. Filter tweets by keywords result in 14 650 different tweets representing around 0.004% of tweets in the initial dataset.

Analyzing the amount of media news related to *novobanco* per year, as depicted in Figure 4.8, we can see that the *novobanco* has been more referenced by the Portuguese media in some years than others. This bar chart shows that the number of tweets is not constant over the years, but 2014, 2018, and 2019 were years where the number of news related to the bank was smaller. *novobanco* was founded in August of 2014, so the lowest number of news in this year can be explained by that. During 2018 and 2019, we did not find any apparent reason for the number to be inferior, but during this time *novobanco* appeared to be in a stable situation. However, posterior years increase can be explained by the enormous losses that the bank presented and some polemics related to the offer of remuneration and bonuses for its managers despite the enormous losses that the bank was having [59]. To analyze twitter patterns, we created similar graphs to the ones shown

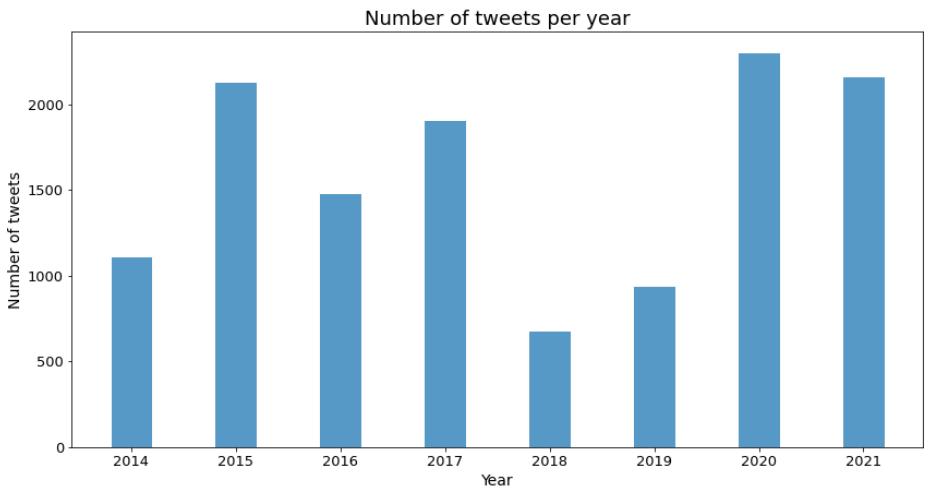
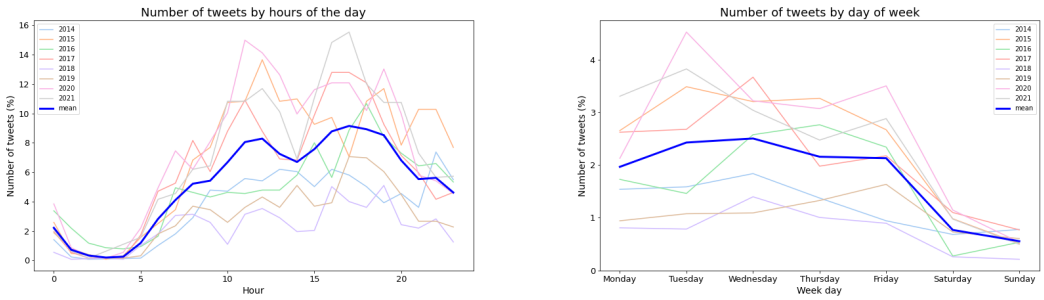


Figure 4.8: Number of tweets per year between 2014 and 2021.

before to study the evolution of tweets per hour of the day, days of the week, and months of the year, depicted in Figure 4.9(a), 4.9(b), and 4.10 respectively. For Figure 4.9(a) and 4.9(b), we can see that the mean line pattern is similar to the pattern obtained with the all media dataset (Figure 4.3). However, when observing the individual lines of each year, we can notice considerable variations between years that did not happen before. With the whole mass media dataset, the number of tweets was following a growing linear trend over the years, but when we filtered, we saw that the number of tweets related to *novobanco* changed from year to year. As it happened before, Figure 4.10 shows that does not exist a clear correlation between months and the number of tweets. We noticed that number of tweets increased in 2014, 2015, and 2020 during August and September, and on May 2021.

Another aspect of interest is the line corresponding to 2014 in this graph that only starts having tweets from July. This is because *novobanco* was founded just in August of 2014, and probably until the official foundation, few people knew about this new bank. This also proves the reason why in 2014, the number of news related to *novobanco* was lower than in other years.



(a) Relation of tweets with hours of the day.

(b) Relation of tweets with days of the week.

Figure 4.9: Relation between the number of tweets per year, in percentage.

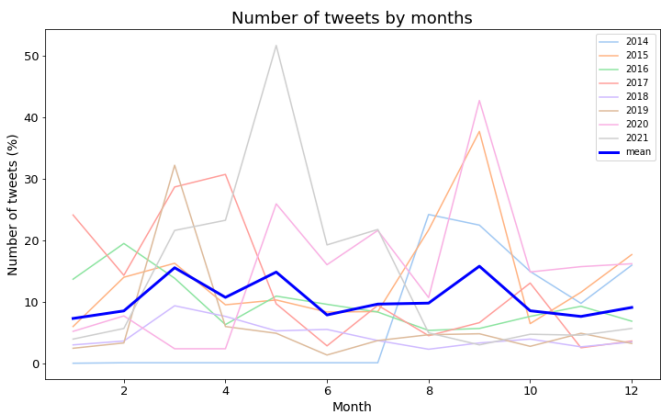


Figure 4.10: Tweets count per year between 2014 and 2021.

4.4.2 Search tweets directly with keywords

The other possibility to retrieve tweets from *Twitter API* is searching directly in Twitter for tweets that contain one or more keywords. This search resulted in 158 959 tweets from different sources (verified and non-verified ones). In this search, more than the news found in the last section, tweets from all users will be in this dataset. Because of this, tweets can have both reliable and non-reliable content. Besides that, this search results in multiple tweets that use slang expressions to address the bank. Although, we should not discard tweets from this search because what people write, even if they use slang, can influence the bank resources usage.

By analyzing the number of tweets of this search per year, we can conclude that there is no clear pattern evidence (Figure 4.11). Moreover, comparing the number of tweets from this search with the number of released news related to *novobanco* (presented before in Figure 4.8) showed that similar patterns were formed, which reveals that in times where there is more released news from some media sources about the bank, there are also more tweets in all Twitter network.

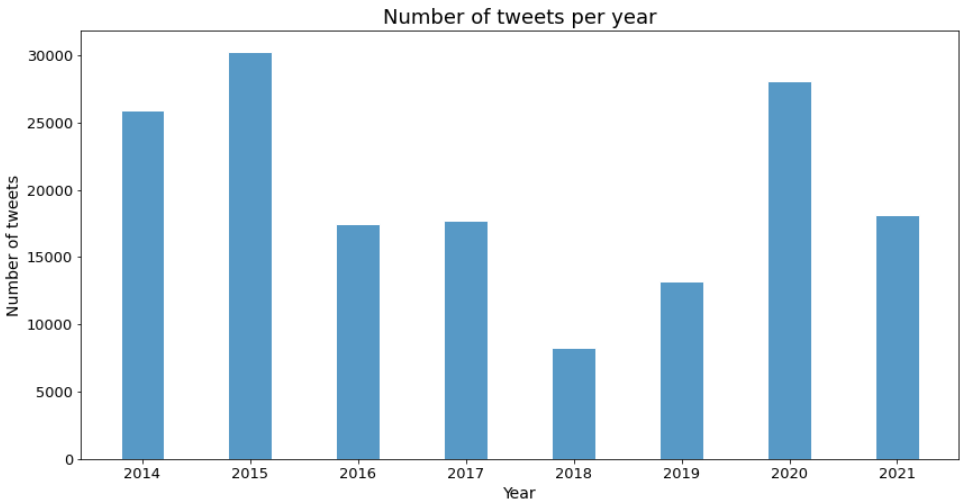


Figure 4.11: Number of tweets per year between 2014 and 2021.

This search resulted in only 75% of tweets in the Portuguese language. Figure 4.12 shows the number of tweets by languages over the years. Compared to graph 4.1, we can observe that this search significantly increased tweets of foreign origin. This happened because tweets from all world that contain the keywords inserted are retrieved.

An analysis of the evolution of tweets per hour of the day, days of the week, and months of the year is illustrated in Figure 4.13(a), 4.13(b), and 4.14, respectively. Figure 4.13(a) and 4.13(b) show that tweets are published mainly during working hours and days. However, it is possible to notice that in the 2014 year, this pattern was not followed because Figure 4.9(b) shows an increasing number of tweets during the weekend.

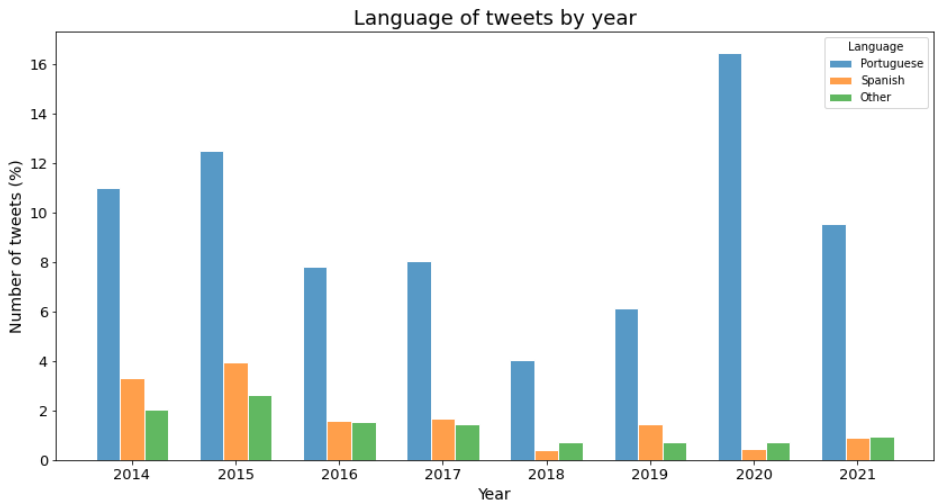
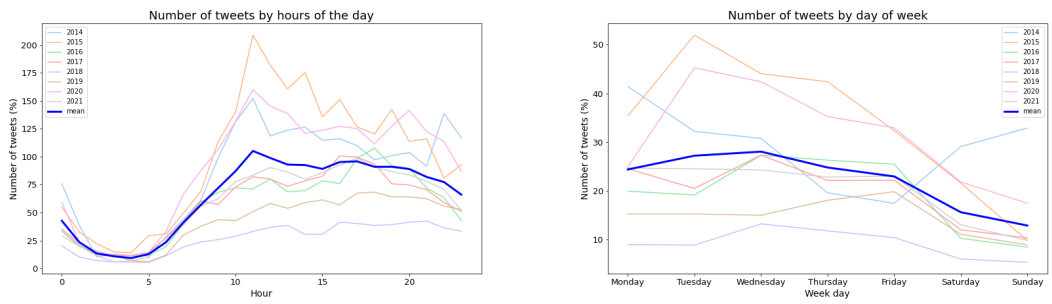


Figure 4.12: Number of tweets in percentage of the main languages between 2014 and 2021. Data extracted from twitter keyword search.



(a) Relation of tweets with hours of the day.

(b) Relation of tweets with days of the week.

Figure 4.13: Relation between the number of tweets per year, in percentage.

Looking at Figure 4.14, we can see a constant volume of tweets throughout all the months, except for August and September in 2014, 2015, and 2020. As mentioned before, the bank was founded in August of 2014, which decreased the total tweets number that year. The 2015 and 2020 years increase can be because some financial reports are published during September. This also happened in the dataset with media news filtered by keywords (Figure 4.10).

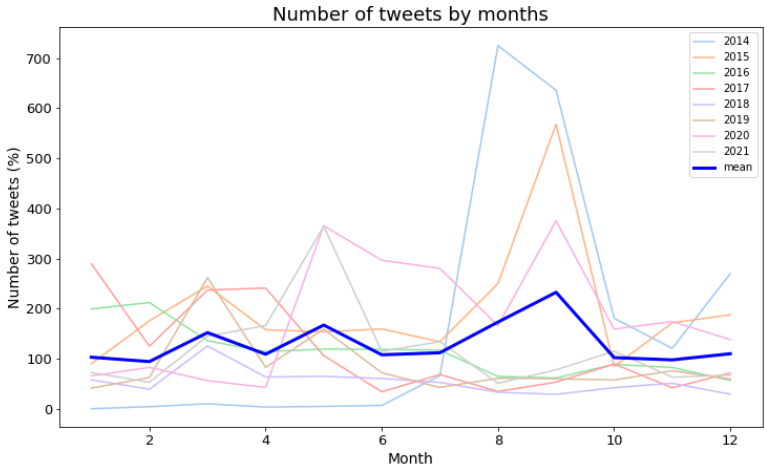


Figure 4.14: Tweets count per year between 2014 and 2021.

5

Interpreting the Twitter Data

This chapter outlines two different approaches, Sentiment analysis and Topic modeling, used to interpret the content of tweets related to novobanco. We start by presenting how we prepared the text of tweets to be analyzed, followed by the experiments with the methods used.

5.1 Natural Language Processing (NLP)

The previous chapter explained how we obtained tweets and presented some data exploration to understand the potential of their features. Although the insights this already gives us, it is also essential to look at the content of each tweet and understand if it has a positive or negative sentiment and what is the main topic present in them. For example, if we have 100 tweets and 90 of those tweets are positive, this can influence the model differently if those 90 tweets are negative.

Since this kind of data is typically too vast for a simple user to analyze manually, it is necessary to find ways to automate this process [60]. Computational linguistics, or simply natural language processing, is the subfield of computer science concerned with making a computer capable of learning, understanding, and generating human language content [61]. In this chapter, we used two natural language processing techniques, Sentiment analysis and Topic modeling. Concerning this analysis, it has just been used tweets related to *novobanco* from the mass media search (presented in 4.4.1). Using this type of tweet allows reliable and trustable information that can provide better conclusions and, therefore, better results for the models. Tweets from all users were not used in this chapter because they can sometimes negatively influence the results as some are baseless opinions and have many slang expressions.

5.2 Pre-processing of the dataset

Before we start analyzing the content of tweets, some pre-process steps need to be done. As shown in Figure 5.1, there are five main steps: *Cleaning the text*, *Tokenization*, *Reducing*

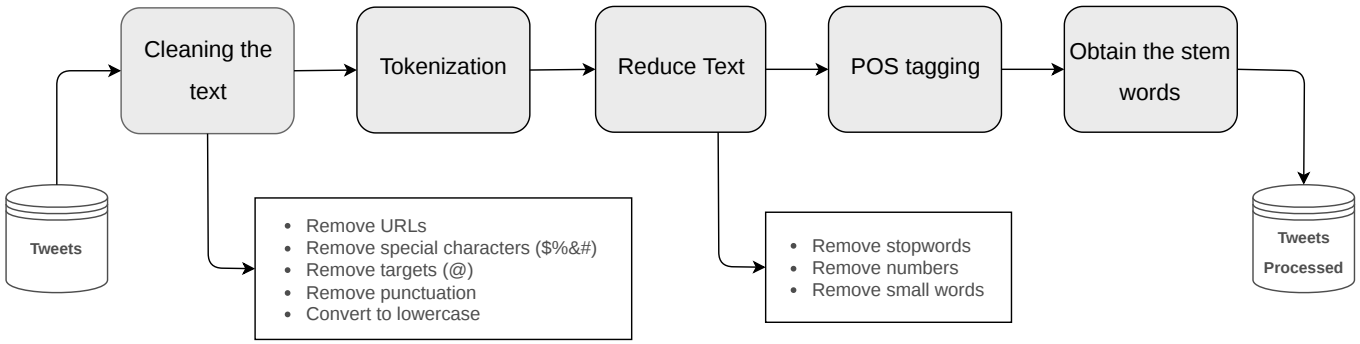


Figure 5.1: General steps use to pre-process texts before analyzing them.

text, POS tagging and Obtaining the stem words [62]. The first step removes all the special characters, punctuation, URLs, and mentions present in the text and, in the end, converts the letters to lowercase. Generally, in this first step, it is common to remove all accents. However, for our sentiment analysis, since we are dealing with Portuguese texts and all the Portuguese libraries of words have accents, we decided not to remove them to avoid creating any conflict. Contrarily, for Topic modeling, all accents were removed. After cleaning the text, we do the *Tokenization* step. This step consists of breaking the text into small chunks called Tokens and can be performed at sentences or word level. Tokenization helps interpret the meaning of the text by analyzing the word sequence [63]. For this, it was used the `word_tokenize` function provided by the `NLTK` [64] library. After Tokenization, we start the *Reduce Text* step that removes words that generally do not have much influence in NLP, such as stopwords, numbers, and words with less than two letters. We use `LX-Stopwords` [65] to remove the stopwords, a Portuguese dictionary composed of 2631 words created by the Natural Language and Speech Group of the University of Lisbon.

The last two steps consist of studying the words more grammatically and were done using *SentiLex-PT* [66]. In lack of words cases in this dictionary, we used an NLP package made for different languages named `Stanza` [67]. *Part-of-speech (POS) tagging* will associate a label indicating the grammatical category it belongs to, and the last step will convert each word to its stem form. Obtaining the stem words can be done using stemming or lemmatization. However, in this work, we decide to use lemmatization as this uses vocabulary and morphological analysis to identify the inflected forms of words (lemmas), and studies show that it can provide better results compared to stemming [68–70]. Figure 5.1 resumes the effect of each pre-processing step presented in this section.

Table 5.1: Effect of the pre-process steps in text.

Text cleaned	**Tokenization**	**Reduce Text**	**POS tagging**	**Lemmatization**
manifestantes entram na sede do novo banco	[manifestantes, entram, na, sede, do, novo, banco]	[manifestantes, entram, sede, novo, banco]	[[manifestantes, NOUN], [entram, VERB], [sede, NOUN], [novo, ADJ], [banco, NOUN]]	[manifestante, entrar, sede, novo, banco]

5.3 Lexicon-Based Sentiment analysis

Sentiment analysis can be defined as the process of classifying opinions, emotions, or attitudes from text, speech, or other sources [60]. Sentiment analysis allows us to understand the general opinion about something at a particular time. More concrete for this work, this analysis will provide essential information about the general satisfaction or not about *novobanco* at each time.

There are two main techniques for doing sentiment analysis: Machine Learning Approaches and Lexicon-Based Approaches. Machine Learning approaches create models that can classify text. In contrast, Lexicon-Based Approaches use dictionaries with sentiment scores and sum the sentiment of each word to understand if the words of a particular tweet are mainly positive, negative, or neutral [60]. Despite sentiment analysis already being a very developed process for the English language, studies dealing with the European Portuguese language and other languages are scarce. Because of this, several studies use Lexicon-Based Approaches or approaches that involve translating the original data to English and then using an English sentiment analysis tool. On the other hand, translation errors and language-specific information can have a significant impact on the final result because some context and specific semantics of each language are lost [71]. Another problem that needs to be present when doing sentiment analysis in languages other than English is the lack of suitable datasets to train and validate the models.

Considering all these problems and trying not to mask the specific syntax terms of the Portuguese language, our baseline approach will be lexicon-based, even though this type of approach cannot deal with domain and context-specific orientations.

5.3.1 SentiLex-PT

To do the Lexicon-Based Sentiment analysis, we start by using the *SentiLex-PT* [66] dictionary, one of the complete dictionaries for the Portuguese language because it was conceived taking into account the different syntactic-semantic contexts in which words can occur. *SentiLex-PT* is a dictionary conceived specifically for opinion and sentiment analysis for texts in European Portuguese, and is composed of 7,014 lemmas and 82,347 inflected forms. Each entry of this dictionary corresponds to one word/expression with information about the grammar category (adjective, noun, verb, or idiomatic expression), the sentiment attribute (1, -1, 0), and the sentiment assignment method (some of the words were labeled by software designed specifically for this purpose).

To classify tweets, we start by counting the number of positive ($nPos$), negative ($nNeg$), and neutral ($nNeu$) words present in each tweet regarding the words and expressions present in the *SentiLex-PT* dictionary. After this, to calculate the overall sentiment ($f(t)$) of each tweet (t), we decided in cases where $nNeu > nPos + nNeg$ to classify tweets as neutral, whereas in other cases we multiply the number of words of each sentiment by its correspondent polarity and sum all of those values ($nPos \cdot 1 + nNeu \cdot 0 + nNeg \cdot -1$).

Equation 5.1 resumes this method in a simplified way.

$$f(t) = \begin{cases} nPos - nNeg & \text{for} \quad nPos + nNeg \geq nNeu \\ 0 & \text{for} \quad nPos + nNeg < nNeu \end{cases} \tag{5.1}$$

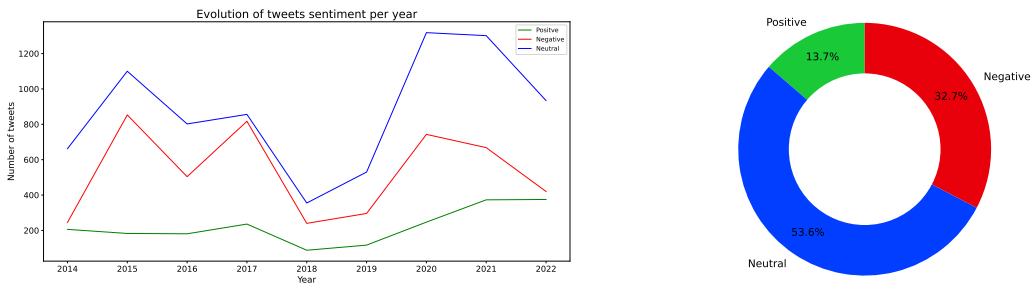
Tweets (t) with $f(t) > 0$ will be classified as positive, $f(t) < 0$ will be classified as negative and $f(t) = 0$ will be classified as neutral.

Knowing that this method fails when giving us the degree of the sentiment of each tweet taking into account the extension of all tweet text, we used another method that follows the equation 5.2. This method computes the ratio of the result of the equation 5.1 and the total number of words present in it ($count(t)$), returning values between -1 and 1. Values close to -1 indicate that the text is highly negative, and values close to 1 indicate that the text is highly positive. Applying this method in our dataset, we obtained values between -0.7 and 0.9.

$$f2(t) = \frac{f(t)}{count(t)} \tag{5.2}$$

The advantage of this method is that it can provide some insights into how positive and negative a tweet is. Despite this, the number of positive, negative, and neutral tweets will remain the same as Equation 5.1 because what will define the polarity is the function $f(t)$. However, this method will bring further benefits in section 5.3.3.

Applying these Equations (5.1 and 5.2), from a total of 14650 tweets, we end up with 2006 positive tweets, 4786 negative tweets, and 7858 neutral tweets (Figure 5.2(b)). In Figure 5.2(a), we observe that the number of positive tweets stays low over the years, instead of negative and neutral tweets that change over the years. It is also interesting to observe that in 2017 the difference between positive and negative tweets was the smallest compared to other years, meaning that it probably is the year when more negative news related to *novobanco* appeared. Figure 5.3 shows a word cloud with the main words present in each sentiment classification.



(a) Evolution of tweets sentiment per year.

(b) Percentage of tweets per sentiment.

Figure 5.2: Results of the lexicon-based sentiment analysis with SentiLex-PT dictionary.

(a) Positive. (b) Neutral. (c) Negative.

Figure 5.3: Word cloud for each type of sentiment.

5.3.2 EMOTAIX.PT

Another dictionary that can be used to do the Lexicon-Based Sentiment analysis is the *EMOTAIX.PT* [72]. This dictionary was initially created by Piolat and Bannour [73], but it was later adapted to European Portuguese. *EMOTAIX* is a dictionary created to organize words into different categories according to their emotional load. It is composed of 3992 words where each word can be part of a Positive, Negative, or Neutral category and is classified into three hierarchical levels: global level (with six topics), middle level (with 18 topics), and specific level (with 50 topics). For example, the global level of one positive word can be Benevolence, Well-being, or Composure, and for a negative word, it could be either Malevolence, Ill-being, or Anxiety. From a total of 14650 tweets, using the

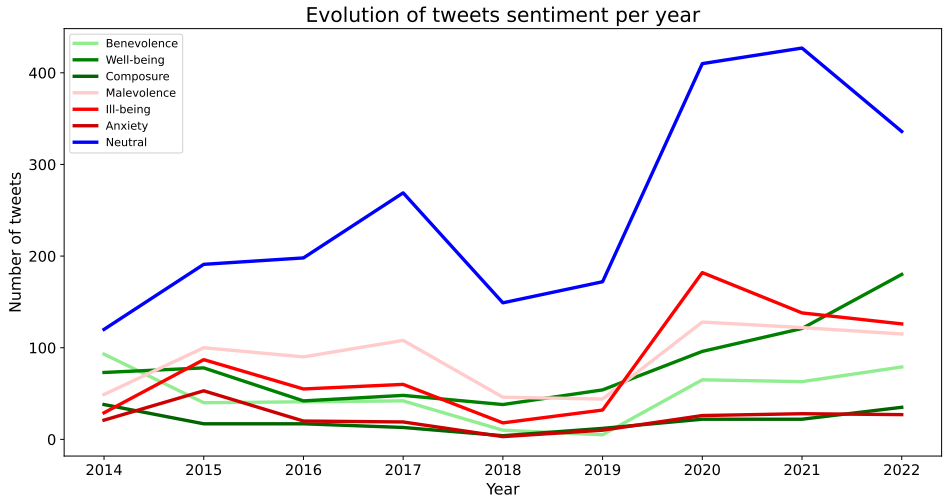


Figure 5.4: Evolution of sentiments present in tweets using EMOTAIX.PT lexicon.

Equations presented in the previous section (5.1 and 5.2), we end up with 1161 positive tweets, 1530 negative tweets, and 11959 neutral tweets. Thus, it is easily concluded that this dictionary cannot classify so many words as *SentiLex-PT* despite the detailed classification for each word that it provides. From all the levels that this lexicon has available, it was just made use of the global one in this work. Figure 5.4 shows the

evolution of each category present at the global level over the years (green lines stand for positive categories and red lines for negative categories). It is possible to see that this lexicon found much more neutral words when compared to the other topics and does not have a clear trend. We can observe that in 2020 exists a small peak in all categories, similar to what happened in the search with the *SentiLex-PT* dictionary.

5.3.3 Validation of results

One of the problems of lexicon-based sentiment analysis is the negation of words like "is not perfect". For this sentence, Lexicon-based approaches probably will classify this sentiment as positive because of the word "perfect"despite not being so. In order to understand if our Lexicon-based classification is classifying well, we decided to manually label some tweets to compute the accuracy of our approaches.

Although this process seems easy, some precautions must be taken because of human's natural lack of attention and errors, which can lead to classification mistakes. So, asking multiple people to annotate the same content is usually a good idea to ensure the best possible result. Table 5.2 resumes some of the work done to create manual data annotations for texts. Shamma *et al.* [74] and Saif *et al.* [75] classifies three times each tweet and tweets with unanimous classifications are discarded. On the other hand, in Rosenthal *et al.* [76] approach, five different persons classify each tweet and, in cases where the results are ambiguous, to avoid discard results, the average is calculated and mapped to the nearest integer value. However, this is a problematic way to describe an ambiguous tweet. For instance, if a controversial tweet gets two negative labels, two positive labels, and one neutral label, it would be labeled neutral. However, this tweet would certainly not be the same as a tweet with a unanimous agreement on a neutral label [77]. Regarding the type of labels that the methods used, we can see that while Shamma *et al.* and Saif *et al.* use a simple label schema, Rosenthal *et al.* uses a more detailed schema.

Table 5.2: Comparison of techniques used to create manual text annotations. (P=Positive, N=Negative, W=Weak, S=Strong, n=Neutral)

Name	Discarted (%)	Coverage	Labels
Shamma *et al.* [74]	33.6%	3x	P, N, Mixed and Other
Saif *et al.* [75]	26.5%	3x	P, N, 0, Mixed and Other
Rosenthal *et al.* [76]	0%	5x	SP, SN, WP, WN and 0
Our approach	5.2%	5x	P, N, n with detailed descriptions

For this work, we follow mainly the guidelines presented in Rosenthal *et al.* [76]. We start by randomly selecting 250 tweets related to *novobanco* from the mass media search and give them to 30 different participants with ages comprised between 21 and 61. The tweets were distributed to each participant so that each participant classified 50

tweets, and five different persons classified each tweet. There was no contact between participants, and some hidden tests for quality control were created. For our hidden test, we selected some tweets with evident sentiment to compare with the participant's answers. If the answers did not match, it probably was a questionnaire with random answers so it will be discarded. The hidden tests resulted in 5 discarded questionnaires. The big difference between our approach and Rosenthal *et al.* is the labeling schema used for classification. In our approach, we create three labels, but to each label is created a description, presented in Table 5.3, because simple schemes create doubt in respondents. The descriptions for each label were based on a questionnaire presented in [78].

Moreover, knowing that the way Rosenthal *et al.* deals with unanimous classifications is not the best, in our approach, we decided to discard tweets without a minimum of 50% of votes in one category. For example, tweets with two positive votes, two negative votes, and one neutral vote were discarded. Furthermore, to obtain the participant's answers, we created a `React JS` application and published it on Git to be easily accessed by all the respondents (`https://gfmateus99.github.io/novobanco-survey/`). A more detailed view of this application, its characteristics, and its internal structure is provided in Annex I.

Table 5.3: Answers provided to the respondents to classify tweets.

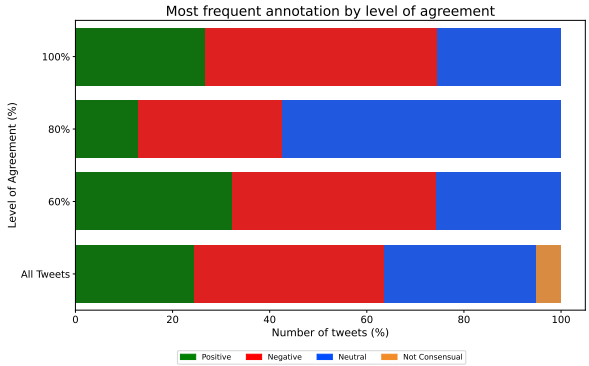
Label	Description
POSITIVE	The tweet contains positive language directly related to *novobanco*, such as expressions of success, compliance with payment deadlines, or a positive attitude.
NEGATIVE	The tweet contains negative language directly related to *novobanco*, such as expressions of criticism, judgment, questioning competence, suspicions of corruption, or expressions of sarcasm.
NEUTRAL	The tweet contains a mix of positive and negative language related to *novobanco* where none stand out. The tweet is not objective, is a subjective opinion, or is not directly related to *novobanco*.

Having all the participant's answers together and to better understand the consistency and agreement of the respondent's answers, we did inter-code reliability using the Fleiss kappa Coefficient [79], which allows us to understand the reliability of agreement (k) between the respondents. Applying this to all tweets manually classified, a value of $k = 0.501$ was obtained, meaning there is moderate strength of agreement between our respondents. This value cannot be considered a bad result considering the subjectivity inherent to some tweets. To better understand the respondent's answers, we can look to Table 5.5(a) where the levels of agreement in tweets are stated. The levels of agreement vary according to the number of equal answers. Thus, a level of agreement of 100% means that all the five answers were the same, a level of agreement of 80% means that there are four equal answers in 5 possible for that tweet, and the same logic applies to the other

levels. As explained before, tweets that do not have a majority of votes (at least a level of agreement of 50%) will be discarded. This resulted in the loss of 13 tweets from the 250 classified manually (5.2%) while the rest 237 (94.8%) were used. Next, in Figure 5.5(b) we can observe that the distribution of each sentiment is similar for all levels of agreement, meaning that in none of the levels exist a high unconformity and variance when compared to other levels of agreement.

Level of Agreement	Count	% of Total
100%	90	36%
80%	54	21.6%
60%	93	37.2%
< 60%	13	5.2%
Total:	250	100%

(a) Annotator agreement rates.



(b) Most frequent annotation by level of agreement.

Figure 5.5: Results of the manual classification agreement.

With this test set obtained, we can advance to the next phase, where it will be analyzed if our sentiment analysis is behaving well. As this can be seen as multi-class classification, to compare and analyze methods, we used mainly the `F1-score` that combines recall and precision into a single metric. To start, we analyze the score of *SentiLex-PT* with Equations 5.1 (standard) and 5.2 (measures the degree of sentiment). For Equation 5.2, we test several approaches to find the best range of degrees where tweets are negative, positive, and neutral. After some tests, we found that this method could improve the general score by classifying tweets with a sentiment between the interval $]-0.1, 0.1[$ as neutral. These are tweets with low sentiment, so they probably are more neutral than positive or negative. Regarding the *EMOTAIX.PT* dictionary using the degree of sentiments to evaluate sentiments did not improve the results in any test. Consequently, for *EMOTAIX.PT*, we used the traditional method that follows Equation 5.1.

Using the best methods found for each of the dictionaries, it was observed that in general *SentiLex-PT* could provide a better `F1-score` for all sentiments (Table 5.4). Knowing that it would be tough to improve the results of each dictionary alone, we tried to create a method that could combine both *SentiLex-PT* and *EMOTAIX.PT* to improve the general score. After some tests, we found that if we classify as negative, tweets with a degree of sentiment provided by *SentiLex-PT* in the interval $]-0.3, 0[$, and for the others, use the *EMOTAIX.PT* classification, we could improve the general score. This new Ensemble method worsened the neutral `F1-score`, but our main objective was to improve the positive and negative scores as those can probably influence the models more. This new ensemble method classifies tweets following Equation 5.3, where t represents a tweet and d is the degree of sentiment in tweet t.

$$ensemble(t,d) = \begin{cases} emotaix & \text{for} \quad d \notin]-0.3, 0[\\ -1 & \text{for} \quad d \in]-0.3, 0[\end{cases} \tag{5.3}$$

Table 5.4 resumes the results of this analysis, having in the count all sentiments individually and globally. The advantage of our ensemble method is that it can provide good accuracy for both positive and negative tweets, which are the ones that can influence most of our models. Regarding the neutral tweets, we could not improve the overall classification too much, probably because our dictionaries are limited and do not have all words. In short, when classifying tweets as positive and negative, our ensemble method has good accuracy, while the classification of neutral tweets is not so trustworthy.

	SentiLex-PT (Degree)			***EMOTAIX.PT***			***Ensemble***		
	Precision	Recall	F1-score	Precision	Recall	F1-score	Precision	Recall	F1-score
Positive	0.36	0.15	**0.21**	0.57	0.13	**0.21**	0.86	0.18	**0.30**
Negative	0.67	0.34	**0.45**	0.81	0.17	**0.29**	0.56	0.50	**0.53**
Neutral	0.46	0.9	**0.60**	0.34	0.85	**0.48**	0.38	0.64	**0.48**
Total	0.50	0.46	**0.48**	0.57	0.38	**0.46**	0.60	0.44	**0.51**

Table 5.4: Comparison of approaches used for Sentiment analysis.

Figure 5.6 lets us easily understand the sentiment trend related to *novobanco* along time. For this, we used our ensemble method to classify tweets as positive, negative, or neutral and assign a value of 1, -1, and 0 to each category, respectively. Next, for each day, we sum the values and plot the grey line (*Sentiment count*) where we can note that the negative sentiment prevails over time. Usually, the sum is much more significant when the negative sentiment prevails than when the positive sentiment prevails. However, this grey line has many oscillations, which makes it challenging to analyze the sentiment trend over time. Consequently, we plot another line (*Sentiment count with seven days moving average*) to have a clear line that lets us analyze the sentiment over time more straightforwardly. This line is a moving average of 7 days, which allows for analyzing the trend more clearly and with less variation. In this moving average, the green regions show zones where the number of positive tweets prevails, while the red show zones where the negative tweets stay. Analyzing this, we can clearly understand that the sentiment related to *novobanco* is primarily negative. However, there are some periods where the positive sentiment prevails, and it is interesting to note that the frequency of these regions appears to increase over time. Also, since July 2021, the amount of negative sum in tweets has decreased, which can be directly related to the decreasing number of tweets related to *novobanco*. Ally, we could also notice that either around July 2020 or July 2021, the number of negative sentiments increased compared to the other months, which can be related to the summer season.

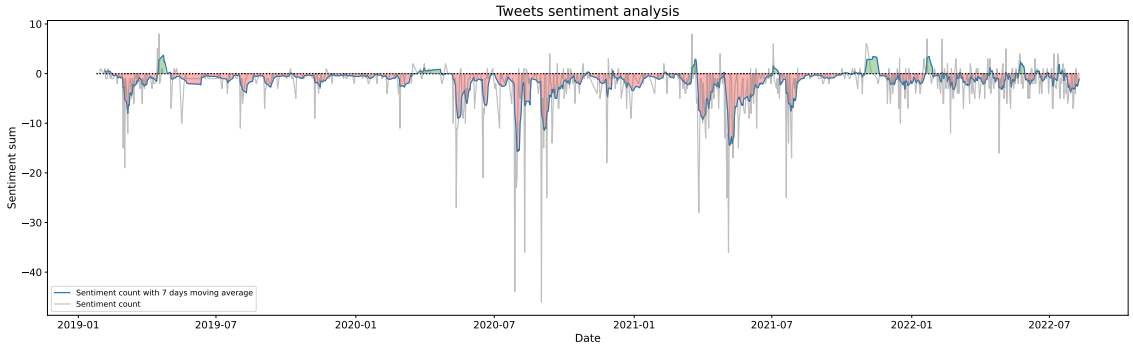


Figure 5.6: Sentiment analysis using our ensemble model

5.4 Topic modelling

Besides the sentiment analysis providing a good indication of the content of the tweets, we can see that results show some subjectivity and are not perfect. As a result, we decided to do a topic modeling analysis where we will try to group tweets according to their topics instead of having the sentiment of tweets. The typical way to perform this is using Latent Dirichlet Allocation (LDA) model.

The LDA model is a probabilistic model where the different texts are represented as random mixtures over latent topics [80]. In other words, the LDA algorithm will look at the different texts to generate several groups of similar words, where each group generated corresponds to a topic. Despite LDA performing well for regular and big texts, when it comes to short texts, like tweets, the results decrease because of the minimal word co-occurrence information [81]. Several different strategies have been proposed to deal with the problem of short texts [82–84] but the one that gained more relevance due to its effectiveness was the Dirichlet Multinomial Mixture (DMM) [85]. DMM alleviates the LDA assumption that each text is a random mixture of latent topics. Knowing that the short text content is limited, DMM assumes that every text corresponds to a single topic. For this work, we will use an DMM derivation that integrates Gibbs sampling, presented in [86]. Authors found out that DMM combined with Gibbs sampling could find the number of topics automatically, with an appropriate balance of homogeneity and completeness, and still fast to compute.

5.4.1 DMM with Gibbs Sampling

To better explain the way this method works, we decided to create a simple analogy. For this analogy, we can think of a game with many players, and to each player is assigned a list of words. The game's objective is for players to form groups based on each list of words. Players with a similar list of words should be in the same group, and players with a different list of words should not be in the same group. Initially, players will be randomly assigned to random groups, and at each round of the game, each player will need to re-choose a group based on the two following rules:

- Choose a group with more people.
- Choose a group whose participants share similar words with him.

As the number of rounds goes on and knowing that we have many more groups available than is necessary, some groups will get bigger, others will get smaller, and some groups will be empty. In the end, we should expect that only a part of the groups have players, and the players in the same group share a similar list of words. If this occurs, it means that the game's objective has been reached.

DMM with Gibbs Sampling works exactly like this game but with the change that it does not use human minds to find new groups. More formally, in DMM with Gibbs Sampling, we will have an input D that is the number of tweets that will be analyzed, a list V that contains all of the different words that appear in the tweets, and a number K that is the number of groups/clusters available. In the beginning, we will assign tweets randomly to the K available clusters, recording information about the clusters where each tweet is ($\overrightarrow{z}$), the number of words in cluster z (m_z), and the number of occurrences of word w in clusters z (n_z^w). After this, in each interaction of this method, the tweets will be re-assign to each group based on the conditional distribution $p(z_d = z|\overrightarrow{z}_{\neg d}, \overrightarrow{d})$, where $\neg d$ means the cluster label of document d is removed from $\overrightarrow{z}$ to avoid to create biased results. Each time a a document get's a new cluster all the variables $\overrightarrow{z}$, m_z, n_z and n_z^w are updated accordingly.

It is expected that if we define K with a big number, as the number of interactions goes on, the tweets will start to be re-organized and find the correct number of clusters K for the specific data. Allowing that a word to appear multi times in a tweet, the conditional probability $p(z_d = z|\overrightarrow{z}_{\neg d}, \overrightarrow{d})$ is derive in Equation 5.4.

$$p(z_d = z|\overrightarrow{z}_{\neg d}, \overrightarrow{d}) = \frac{m_{z,\neg d} + \alpha}{D - 1 + K\alpha} \frac{\prod_{w \epsilon d} \prod_{j=1}^{N_d^w} (n_{z,\neg d}^w + \beta + j - 1)}{\prod_{i=1}^{N_d} (n_{z,\neg d} + V\beta + i - 1)} \tag{5.4}$$

The first part of the Equation 5.4 relates to the first rule of the game presented above ("Choose a group with more people"). This part tries to join tweets in big groups instead of being isolated. The $m_{z,\neg d}$ is the number of tweets in each group excluding tweet d, and α is a parameter that needs to be defined manually. A value of α near 0 makes the probability of tweets choosing an empty group low, while a big value for α makes the probability of tweets choosing an empty table high.

The second part of the Equation 5.4 relates to the second rule of the game presented above ("Choose a group whose participants share similar words"). This part tries that tweets choose groups that share similar words. The $n_{z,\neg d}^w$ is the number of occurrences of word w in group z and $n_{z,\neg d}$ is the number number of words w in group z, without considering the tweet d in both cases. Larger $n_{z,\neg d}^w$ indicates that the tweet d shares similar interests with group z, so the probability of tweet d choosing group z will be larger. This part also has a parameter β that must be defined manually. If a value of 0 is given to β,

tweets will never join tables that do not have a word of tweet d. A low value of β is not reasonable because tweet d can share multiple words with group z, but because others do not appear in the group, tweet d will not join that group. Moreover, as it is allowed that a word appears multiple times in each tweet, we need to add $\prod_{j=1}^{N_d^w}$ to the expression where N_d^w is the number of occurrences of word w in tweet d.

5.4.2 Coherence Metric

In topic modeling, one of the significant issues is how to evaluate the multiple models and choose the best. The best and trustful way is to look at the topics and manually evaluate the coherence. However, this can be tedious as the range of parameter values is high, and sometimes we end with many topics found. Consequently, some metrics were created that allow understanding of how much a topic is coherent. Some metrics use statistics and probabilities, while others look for the semantic similarity of words in each topic. However, this second way needs large and good texts to train, which becomes more difficult for the European Portuguese language. Considering this, we decided to use a metric and human classification mix to evaluate the different topics. In our approach, we started by using a coherence metric to find a good range of values to test and understand the influence of the different parameters on the results. After that, we evaluate the topics manually with a smaller range of values.

The coherence metric chosen for this work follows the Equation 5.5 [87]. For each topic, this metric counts the number of times two words (v_i, v_j) of the same topic co-occur and divides this by the number of documents in which the word v_j appears. For obtaining the total coherence for a topic T, the logarithm of this division is summed for every combination of words (v_i, v_j) of a topic. This metric will usually give negative values, as normally, the number of documents in which a word v_j appears will be higher than the number of documents in two words of the same topic (v_i, v_j) co-occur. Thus, as the denominator of the Equation 5.5 usually is higher, the result of the division will be a number under one, which will provide a negative value when the logarithm is applied. Finally, an value ϵ is added to prevent having a logarithm of zero.

$$coherence(T) = \sum_{(v_i, v_j) \in T} \log \frac{D(v_i, v_j) + \epsilon}{D(v_j)} \tag{5.5}$$

5.4.3 Experiments

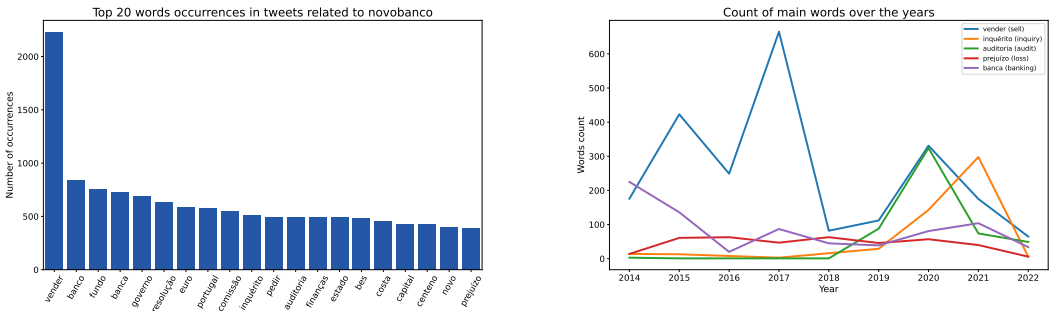
Knowing that all the tweets that are being analyzed were obtained through the *"Novo Banco"* and *"novobanco"* keywords, we decided to take out these words to avoid these words creating garbage and damaging our results. This step is similar to adding the words *"Novo Banco"* and *"novobanco"* to our stopwords list to be removed. After this additional step, twelve tweets ended with no words, so they were excluded from our initial dataset of 14650 records. Furthermore, to better understand this dataset, we run

some statistics presented in Table 5.5. Note that during this section, all statistics and analysis were done after all preprocessing steps presented in 5.2 plus the additional step of taking out the words *"Novo Banco"* and *"novobanco"*. An interesting thing to note through the statistics of Table 5.5 is that the preprocessing steps reduced 4526 words from the vocabulary.

Table 5.5: Statistics about tweets related to novobanco.

Number of tweets	**Size of vocabulary (before preprocessing)**	**Average (Standard deviation) of tweets length**	**Minimum (Maximum) tweet length**
14638	6449 (10975)	8.3 (2.9)	2 (28)

Therefore, in Figure 5.7(a) we list the top 20 word occurrences in tweets related to novobanco, while in Figure 5.7(b) it is presented the occurrences of some words over the years. It is interesting to observe that the word sell (*"vender"*) has significantly more occurrences than the other ones over all the years, and words like inquiry (*"inquérito"*) and audit (*"auditoria"*) have an increase in last years.



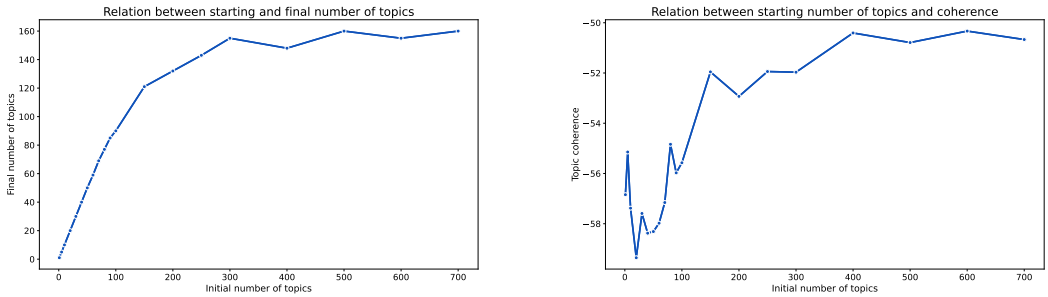
(a) Top 20 words occurrences.

(b) Occurrences of words over the years.

Figure 5.7: Occurrences of main words in tweets related to novobanco.

To start finding the best parameters for DMM with the Gibbs Sampling model, we investigate how the number of initial topics (K) influences the number of final topics. For this, we fixed the number of iterations to 30 and both α and β parameters to $0,1$. The results of this search are presented in Figure 5.8. The first graph (5.8(a)) shows the relation between starting and the final number of topics while the second graph (5.8(b)) presents the coherence obtained through the Equation 5.5 for different K values. Focusing on the first graph, we can observe that from the starting number of topics K of 300, the final number of topics found converges for a value near 150, indicating that a value for $K = 300$ is sufficient for the algorithm to find the best topics. Although looking at the second graph, we can see the coherence start converging when $K = 400$. Analyzing both graphs together, we can see that from values of $K = 400$ the number of topics converges,

and the coherence does not change a lot which can be a good sign that a value of $K = 400$ is good enough for our model.

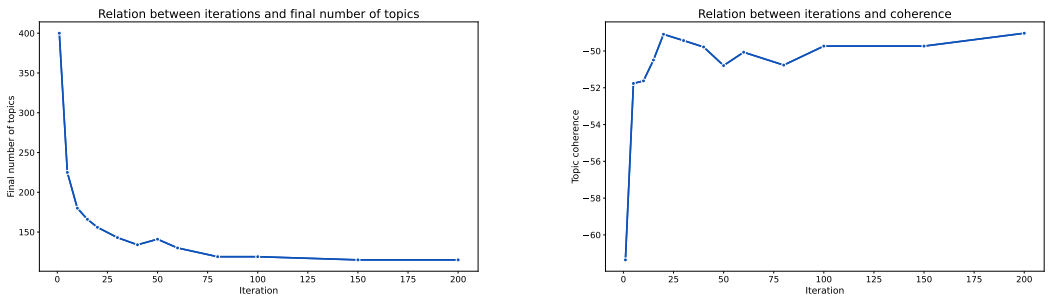


(a) Influence of starting number of topics.

(b) Coherence for different starting number of topics.

Figure 5.8: Relation between the starting number of topic with the final number of topics and coherence.

With a good value for K found, the next test will try to find how many iterations we should run to find the best topics. For this one, we fixed the number of initial topics K to400, both α and β parameters to $0,1$, and ran the model with different values for iterations. Figure 5.9 shows the results of this experiment. Looking at the graphs presented in this Figure, we observe that as the number of iterations increases, the number of final topics decreases, and the coherence increases. From iteration 20, the number of topics found started converging, getting a more stable value since iteration 80. The topic's coherence reaches the lowest value at iteration 20, and from this iteration on, coherence has slight variations but without deviating too much from the other values. Observing both graphs together, a value higher than 80 for the number of iterations seems good enough to find the lowest number of topics with good coherence on topics. We decide for the following experiments to choose 100 to the number of iterations.



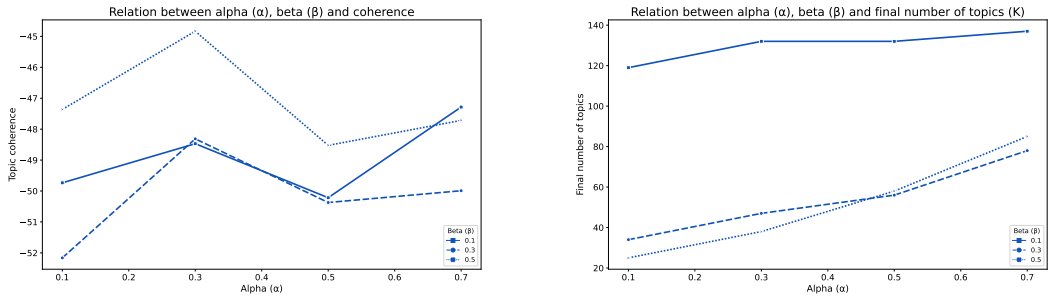
(a) Influence of the number of iterations.

(b) Coherence for the different number of iterations.

Figure 5.9: Relation between the number of iterations with the final number of topics and coherence.

After finding a good value for the starting number of initial topics K (400) and for

the number of iterations (100) defined, the parameters that still need to be studied are α and β. Recalling what was explained before, α will mainly influence the number of groups, while β will mainly influence how similar tweets from the same topic are. For the next test we defined a range of values for α of $\{0.1, 0.3, 0.5, 0.7\}$ and for β of $\{0.1, 0.3, 0.5\}$. Figure 5.10 provide the results of this experiment. Looking at the first graph of this Figure (5.10(a)), where it is analyzed the relation of α and β with coherence, the first conclusion that we can take is that, in general, the model gets the best coherence for all values of α when $\beta = 0.5$. However, this graph also shows that when $\alpha = 0.3$, all the different models have a peak for all β values. Considering the second graph (5.10(b)), where the relation of α and β with the final number of topics is analyzed, we can observe that as the value of α increases, the value of final topics increases for all different β. This was expected because as the value of α increases, the probability of the number of groups growing will increase. Analyzing both graphs together, it appears that a value of $\beta = 0.5$ combined with $\alpha = 0.3$ provides the best model regarding consistency and the final number of topics.



(a) Influence of alpha and beta.

(b) Coherence for the different alpha and beta values.

Figure 5.10: Relation between alpha and beta values with the final number of topics and coherence.

The experiments explained above allow to obtain what seems to be reasonable values for the different parameters that need to be defined in DMM with Gibbs Sampling. We did some extra tests varying these parameters to evaluate manually if the topics found would be better. However, the differences did not appear to be significant, so we decided to stick to those values. Our final model was trained with $K = 400$, $\beta = 0.5$, $\alpha = 0.3$, 100 iterations, and resulted in 61 different topics. However, having a model with the best parameters is not enough for topic modeling as the interpretability of topics found is also an important step. To start, we analyze the frequency of different topics to understand if some topics could be discarded (Figure 5.11). This Figure shows that most topics are not frequent while some stand out from others. To get an idea, from the total of 61 topics found in our final model, 33 had one document, 10 had two documents, and 7 had two documents. This indicates that these topics are not very relevant but before we remove we take a look inside each of them to understand if some could be a potentially good topic. However, in general, none of them appeared to have relevant or significantly different

information compared to the topics with more documents. In the end, we removed all of them to have a cohesive set of topics, leading to 11 final topics.

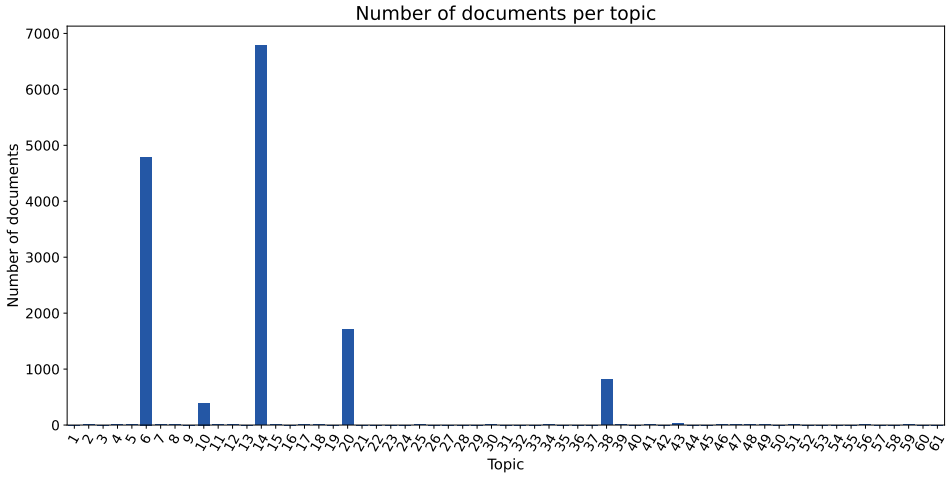


Figure 5.11: Number of documents per topic found in our final model.

With a number of topics more manageable to analyze, we can do a more concise manual investigation of the content of each topic individually. For each set of words, we tried to find a name that could describe the topic well and if each has good or poor coherence, and the results are shown in Table 5.6. For a simple understanding in each line of the Table, the first number corresponds to the topic number and matches the topic number in Figure 5.11. Analyzing the results, we could understand that most words inside each topic obtained were related, and it was possible to find a subject for each topic. Despite this, there were some topics where it was more challenging to find a simple word with the capacity to describe the topic. A case of this is topics 17 and 18, where we could not find a subject with the ability to describe the content. However, observing Table 5.6, it is interesting to see that the topics with more documents (6, 10, 14, 36, 20, and 38) are not too difficult to explain.

After having the topics classified, in Figure 5.12, we provide an analysis of the topics over time, relating them with the sentiment analysis previously done through our ensemble model. In this Figure, we can find in the positive part of the x-axis the number of tweets of the five main topics over time. Analyzing this part of the graph, firstly, we can notice that *Politics and government* and *Financial problems* topics have considerable relevance until the beginning of 2022 compared to the other topics. Moreover, in all periods close to summer, these topics seem to have more preponderance relating to the different periods. From 2022 onward, the relevance of these topics decreases and resembles the other topics. Moving now to the analysis of the negative part of the x-axis, we can find the sentiment count over time with a seven-day moving average (similar to Figure 5.6. However, in this case, we inverted the axis of the positive tweets to be easy to compare to topics). Analyzing both graphs together, we can find that the periods where

most tweets related to *Politics and government* and *Financial problems* are released are also periods where the sentiments have the highest negative rate, meaning that probably the news about these two topics are almost all negative which makes sense because *Financial problems* topic should contain almost negative tweets. Furthermore, it is also possible to observe that from 2022 onward, the relevance of *Politics and government* and *Financial problems* topics decrease matches with the period where the frequency of green zones (positive sentiment) starts to increase, which can indicate that the positive opinion about the bank is growing, and the problems of the bank are decreasing.

Table 5.6: Final topics found with the best model after removing topics with low frequency (less than 4).

	Number of documents	**Topic**	**Words**
6	4791	Politics and government	vender, governo, inquerito, auditoria, comissao, costa, centeno, pedir, parlamento, psd
8	6	–	empresa, desconhecer, grau, clube, envolvimento, boaventura, esquema, cesar, publico, ministerio
10	394	Credits and growth perspective	rating, moody, dbrs, lixo, manter, alta, perspetiva, melhorar, positivo, cortar
14	6791	Financial problems	vender, fundo, banco, banca, euro, resolucao, portugal, prejuizo, financas, trabalhador
17	8	–	partilha, operacao, injecao, restante, criar, fazer, pinheiro, plano, atualidade, mesa
18	13	–	felix, custar, facilidade, lar, troco, ver, bagao, obrigacao, mourinho, transmissao,
20	1705	Bank problems	bes, bloco de esquerda, lesado, emigrante, cliente, comercial, tribunal, papel, vender, solucao
38	810	Sports	clube, liga, futebol, bessa, benfica, mundial, portugues, premio, uefa, best
43	29	Names	rtp, controlar, silva, martim, jose, antonio, pedro, eduardo, ana
56	13	Cinema and movies	cinema, cineclube, pedir, final, testar, exibidor, filme, covid, grande, cliente
59	5	Magazines and apps	jornal, disponivel, revista, consulta, tornar, manha, permanente, correio, app, sapo

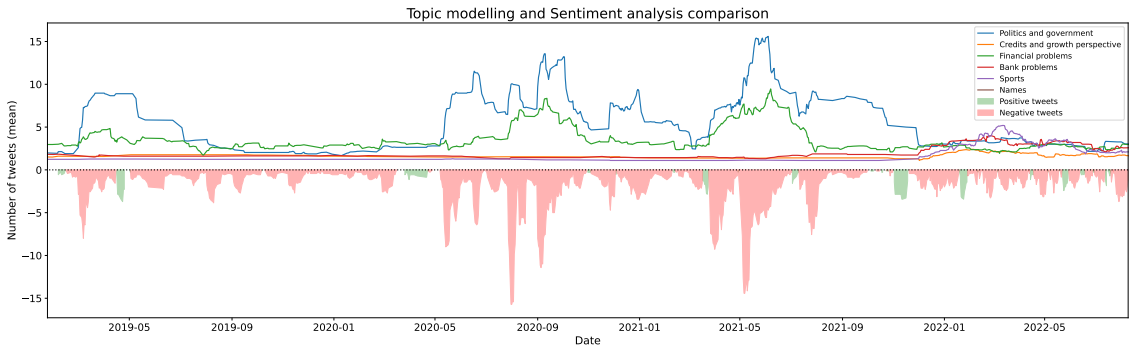


Figure 5.12: Trend of the main topics and sentiment (with 7 days moving average) over time.

6

Data center baseline analysis

This chapter describes the baseline analysis of novobanco data center servers, providing some powerful insights for the model we want to develop. Furthermore, in this chapter, we also study some relations between novobanco data center servers usage and human context (i.e., tweets related to novobanco).

6.1 One server, different patterns

Regarding the data center exploratory data analysis of Chapter 3, specifically, Figure 3.7, we could understand that each server has a specific pattern and besides the existence of some relations between servers of the same layer, along time the patterns change for each server. Figure 6.1 shows the CPU usage of one server of the first layer over time and highlights different grey patterns in different regions. Moreover, we can notice that most of the time, the CPU workload follows a constant trend, and nothing unusual happens, so if ranges like this one were always selected to predict, the predictions would be considerably better compared to other ranges that contain unique patterns.

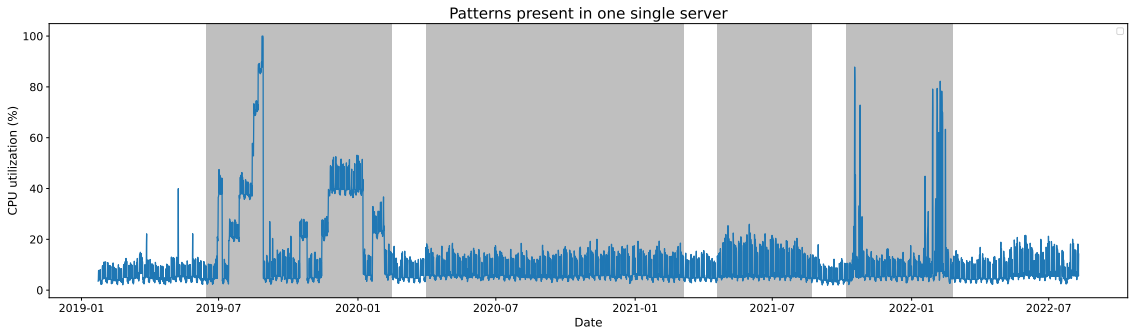
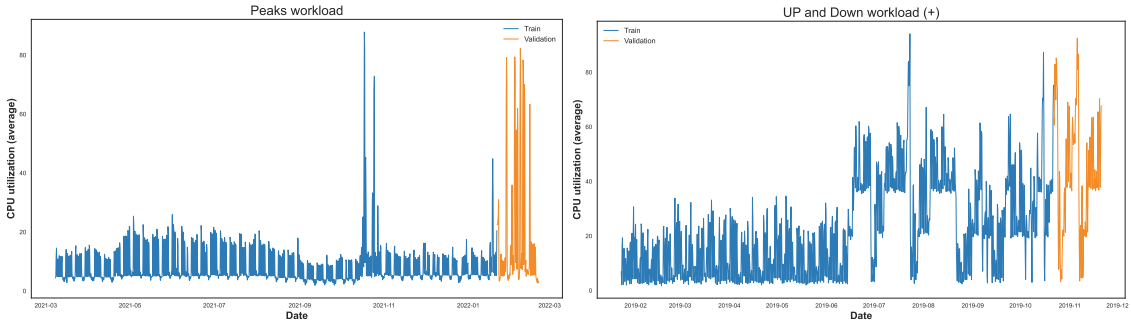


Figure 6.1: Patterns present in one single server. Different patterns (highlighted in grey) are found over time for each server.

Knowing that this phenomenon occurs among all servers and as our objective is to have one model that can be used for all servers, instead of initially analyzing each server individually, we tried to find examples of the main patterns that occur among the servers

in the analysis. In other words, we created 11 slots of 365 days in a way that each slot contains a pattern (figure 6.2 shows an example of this, presenting two slots of 365, where each Figure includes a different pattern). The motivation behind this was to understand how models perform regarding different scenarios and to confirm our state-of-the-art premise, stating that for different workload patterns, different models yield better estimations (Table 2.1). In the following sections, we will describe how we tested each of the models selected for this work to obtain the baseline results.



(a) Peaks pattern.

(b) UP and Down pattern with high variation.

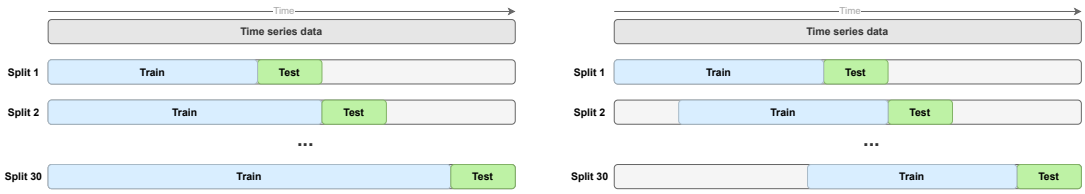
Figure 6.2: Examples of workload patterns selected to observe model performance.

6.2 Baseline Models study

The data analysis was done previously made it possible to observe the presence of seasonalities, mainly daily and weekly. Additionally, with the data analysis step and considering the project objectives, it is more beneficial to forecast 1-day using the 6 hours step dataset as the range of points is more extensive. This way, forecasting 1-day of data means predicting 4 points ahead of data (24 hours / 6 hours step = 4). As we are trying to see how models perform over time, a 30-fold cross-validation approach with a 1-day forecast prediction was used to evaluate the baseline model's accuracies. An extensive cross-validation range was chosen to have a high confidence level in the model's prediction accuracy.

Concerning cross-validation in time series, many methods are frequently employed to ensure that information from the future is not taken into account when forecasting points from the past. One is forward chaining cross-validation, where we select a subset of data to train and forecast the following data points, T. The true values of points T that was previously predicted will be given to the model to get the next prediction T+1, as can be easily understood in Figure 6.3(a). However, this approach has some disadvantages, as it can bring leakage to future forecasts. That is why a new technique named blocked cross-validation was introduced. Opposite to the previous one, blocked cross-validation sets a default range for training, and that range is used in all iterations of the cross-validation

folds, as shown in Figure 6.3(b). To obtain the baseline results, we made use of blocked cross-validation. Each prediction was made by training the model with data until the prediction day. For example, when predicting the CPU usage for a day T, the model was trained with data until T-1 day, and so on, until predicting the 30 days necessary for cross-validation. To evaluate predictions, the Mean Squared Error (MSE) and MAPE were calculated for each fold and then averaged to produce the final score for each model to assess predictions.



(a) Forward chaining cross-validation.

(b) Blocked range cross-validation.

Figure 6.3: Illustration of the 30 days cross-validation procedures applied for testing the models.

As was previously demonstrated, not all models accept all types of covariates. Except for the Exponential Smoothing model, all the other models accept future covariates, so the rest of the models were tested with the known future covariates as day, month, and year. Regarding past covariates, just the Transformer and TFT models allow their usage, so we did not use past covariates in any model to find the best parameters in this first test. For each pattern selected, we tested with six different models, three deep learning models LSTM and, Transformer (normal and TFT version), and three machine learning models, Prophet, SARIMA, and Exponential Smoothing. All these models were tested and trained equally to maintain non-biased results. In order to find the best parameters for each model in each pattern, we run the models with different combinations of parameters, as will be explained next.

Exponential Smoothing

The exponential smoothing model has two possible parameters to optimize: the type of trend component and the seasonal periods. Seasonal periods are related to the range of the main seasonal period in the data. Knowing this data has four entries per day, we tested this parameter with 4 (daily seasonality) and 28 (weekly seasonality). The trend component is related to the type of decomposition to use. It can be either additive, meaning that the trend is linear, or multiplicative, meaning that seasonality may change over time with the trend. The model that provided the best results for all patterns had a seasonal period of 28 and an additive trend component.

Prophet

Similar to the exponential smoothing model, Prophet also has a trend parameter that can be additive or multiplicative. In this case, the trend was defined as multiplicative, as it led to better results. The growth function was kept as linear (default setting), and the weekly, monthly, and daily seasonality parameters were set as True, not only because of the EDA study (Section 3) but also because it led to better results in all patterns.

SARIMA

Opposite to the previous two models, SARIMA has much more parameters to optimize. There is a (p, d, q) component similar to the one of ARIMA model (where p stands for the AR order, d for the integration order, and q for the MA order), but also a (P, D, Q, s) component that is used for model the seasonal terms (where P stands for the seasonal AR order, D for the seasonal integration order, Q for the seasonal MA order, and s for the periodicity present in data).

To select the appropriate SARIMA parameters for each pattern, we first study the ACF and PACF as they can help us to determine the order of AR and MA terms. Although these functions do not directly determine the AR and MA model's order, they can provide an idea of which one is a good fit for the data. The appropriate AR and MA components can be defined by combining the ACF and the PACF plots, observing the amount of relevant statistical lags in each plot. For example, to find the value for the AR component, we need to look at the point where PACF drop, while to find the value for the MA component, we need to look to the point where ACF drop. With these insights about the values of the components, we selected a range of values that fulfilled the insight of the ACF and PACF tests and did a grid search of the parameters. To select the best model for each pattern, the AIC and BIC were investigated, where the model with the lowest AIC was selected for each pattern.

Long-short Term Memory

LSTM is the first deep learning model that we will analyze, and the problem of deep learning models is the number of parameters that need to be defined and the amount of time that needs to train compared to the machine learning models. Due to this, LSTM and the remaining deep learning models used in this work (Transformer and TFT) were trained to add an early stopping procedure to the training process with patience of 5 epochs. This will stop training if, during five consecutive epochs, the loss does not decrease.

To find the best model for each pattern, we did a grid search over a set of parameters as the number of layers ([1, 2, 3, 4, 5]), dropout ([0,0.15,0.3,0.5]), hidden dimension ([5,10,20,50,100]), learning rate ([1e-3,1e-4,1e-5]), input chunk length ([28, 120]), and loss (MSE and *L1*, equation 6.1 and 6.2 respectively, where n is the number of samples, $\hat{y}_i$

is the predicted value for sample i, and y_i is the true value for sample i). The optimizer was kept as Adam, and the rest parameters were kept as default. Like in SARIMA model, there was no model configuration that suited all patterns.

$$MSE_{loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.1}$$

$$L1_{loss} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{6.2}$$

Transformer

The next model implemented was the Transformer. As LSTM, Transformer also has many parameters to optimize and was tuned on the number of encoder layers in the encoder ([3,4,5]), the number of decoder layers in the decoder ([3,4,5]), the dimension of the feedforward network model ([512, 1024, 2048]), dropout ([0,0.15,0.3,0.5]), learning rate ([1e-3,1e-4,1e-5]), and loss (MSE and *L1*, equation 6.1 and 6.2 respectively). The optimizer was kept as Adam, and the rest parameters were kept as default.

Temporal Fusion Transformer

The final baseline model implemented was the TFT model. For this model, the parameters were tuned on dropout ([0,0.15,0.3,0.5]), learning rate ([1e-3,1e-4,1e-5]), input chunk length ([28, 56]), and batch size ([16, 32]). The optimizer was kept as Adam, and the rest parameters were kept as default. Parameters such as the number of attention heads and the number of LSTM layers were kept as 4 and 1, respectively, as it is stated to be good default values.

After doing a grid search of parameters for all models, we ended up with different configurations of each model for each pattern. Except for the Exponential Smoothing and Prophet, where the configuration maintained the same for all patterns, the remaining model parameters chosen for each pattern are presented next in Table 6.1. (The order of parameters in Table for SARIMA is (p, d, q)(P, D, Q)s; for LSTM is (nº of layers, dropout, learning rate, loss); for Transformer is (dropout, learning rate, loss, nº of encoder layers, nº of decoder layers, dimension of the feedforward); and for TFT is (dropout, learning rate, input chunk length, batch size).

6.2.1 Results

After running every baseline model chosen in each pattern, except for Exponential Smoothing and Prophet, we did not obtain a configuration for a model that is the best for all patterns, which makes sense because as the number of parameters that are available to optimize increases, the number of different configurations for each model also increases.

Table 6.1: Best model parameters for each pattern.

Pattern	SARIMA	LSTM	Transformer	TFT
Constant workload (-)	(3,0,3)(1,0,3)28	(2,0,50,0.00001,'L1')	(0.15,0.001,MSE,4,5,1024)	(0.15,0.001,56,16)
Constant workload (+)	(1,0,2)(3,0,2)28	(2,0.3,150,0.001,'L1')	(0,0.001,'L1',3,3,512)	(0,0.001,28,16)
Down workload	(4,0,3)(4,0,1)28	(2,0.15,150,0.001,'L1')	(0,0.0001,'L1',5,4,1024)	(0.5,0.001,28,32)
Growing Up workload (-)	(0,0,1)(4,0,2)4	(2,0.15,50,0.001,MSE)	(0.15,0.00001,MSE,5,4,2048)	(0,0.001,28,16)
Growing Up workload (+)	(1,0,1)(3,0,2)4	(1,0,150,0.001,MSE)	(0,0.0001,'L1',5,3,2048)	(0.15,0.001,28,16)
Low-High change workload	(2,0,3)(0,0,2)4	(2,0,5,0.001,MSE)	(0.3,0.001,MSE,5,4,2048)	(0.15,0.001,56,16)
High-Low change workload	(4,0,3)(2,0,2)28	(1,0,150,0.001,MSE)	(0.15,0.0001,'L1',5,4,1024)	(0,0.001,56,16)
Peaks workload	(4,0,3)(2,0,3)28	(2,0,150,0.001,'L1')	(0.15,0.0001,MSE,4,4,2048)	(0.3, 0.001, 28, 32)
Random workload	(1,0,2)(2,0,3)28	(2,0.15,150,0.001,'L1')	(0,0.001,'L1',4,3,512)	(0,0.001,56,16)
UP and Down workload (-)	(4,0,3)(2,0,1)28	(1,0,150,0.001,'L1')	(0.3,0.001,'L1',4,5,512)	(0,0.001,28,16)
UP and Down workload (+)	(0,0,3)(3,0,1)4	(2,0,5,0.001,'L1')	(0,0.00001,'L1',5,5,1024)	(0,0.001,56,32)

Table 6.2 resumes the results obtained for each pattern, showing which model obtained the best MAPE and the mean of MAPE for every pattern. Looking at this table, the first conclusion that can be taken is that for different workloads, different models yield better estimations, which confirms the state-of-the-art premise in our dataset. This means that there is not an optimal model for this dataset, and it gives forces to the theory that an ensemble model probably is the best and most potent for this dataset. Another interesting thing is the difference in each pattern between the best MAPE, and the average MAPE of all models. In some patterns, as the *High-Low change workload* or *Growing Up workload (-)*, we get a best MAPE that is almost 2-3 times lower than the average MAPE value in all patterns. Moreover, it was also possible to notice that the TFT model never had a better performance than others for any pattern.

Note that the results in the table were obtained using blocked cross-validation with 335 days of train. To compare, we also tested the models, but now, instead of training with 335 days, we used just 150 days to train, which increased the best MAPE in 1.8% and the average MAPE in 8.15%. This test was done to understand if we decreased the training days, the results decreased a lot, but the conclusion is that it decreased a little, but not significantly.

Looking at the bar plot present in Figure 6.4, we can understand in a better way the differences of MAPE of the best model in each pattern and how we end up in some cases with errors for the best model so much lower than the average errors of all models. For a better visualization, due to the differences in errors range, we restricted the y-axis to the interval of [0, 100], which caused that in some cases, where the error was superior to 100%, we cannot see the end of the bar like in Exponential Smoothing and SARIMA bar in *Peaks workload*. One of the most curious observations looking at this graph is to observe that SARIMA is, in general, one of the best models found in this dataset, but in some cases, as the *UP and Down workload (-)* it presents an error a lot bigger than the others models for the same pattern. In addition, observing this Figure is also possible

Table 6.2: Mean Absolute Percentage Error of the best predictor depending on workload pattern. (- and + refer to workloads with low and high variation, respectively.)

Pattern	Best model	MAPE	MAPE (Average)
Constant workload (-)	SARIMA	1.50%	2.61%
Constant workload (+)	Transformer	8.19%	14.31%
Down workload	SARIMA	14.34%	36.07%
Growing Up workload (-)	Exponential smoothing	4.96%	8.46%
Growing Up workload (+)	LSTM	11.36%	20.77%
Low-High change workload	Exponential smoothing	18.61%	30.73%
High-Low change workload	SARIMA	15.32%	41.07%
Peaks workload	LSTM	61.94%	99.15%
Random workload	SARIMA	18.37%	36.11%
UP and Down workload (-)	Prophet	24.72%	42.81%
UP and Down workload (+)	SARIMA	16.75%	55.23%
	Average (335 days back)	**17.83%**	**35.12%**

to observe that the TFT model provided results very similar to the Transformer model, being that, in 5 of 11 patterns, TFT over passed Transformer, while in the remaining six patterns the Transformer yield lowest errors.

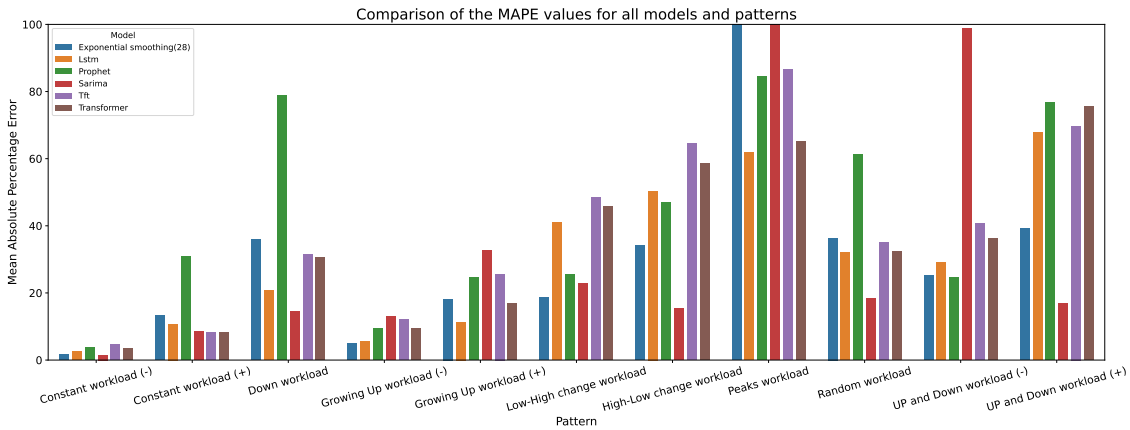


Figure 6.4: Comparison of the MAPE values for all models and patterns.

To finish this analysis, Table 6.3 presents the individual model performance along all patterns, which represents the error that we could obtain if, for each pattern, we use each model with the best parameters obtained. It is interesting to observe that the LSTM in the patterns test was just chosen as the best two times, but overall it is the best, while SARIMA was the model that was more times chosen (5 times in 11) but it is not one of the best models in overall terms. On the opposite, the two best overall models (LSTM and Exponential smoothing) together are just chosen as best four times which is lower than the number of times the SARIMA model is chosen.

Table 6.3: Mean Absolute Percentage Error of each model individually.

SARIMA	Prophet	Exponential Smoothing	LSTM	TFT	Transformer
36.8%	42.5%	31.3%	30.4%	38.8%	34.71%

6.2.2 Covariates study

In the previous study, we did not use past covariates because just the Transformer and TFT model allows the usage of them. Concerning the metrics analysis done previously (Chapter 3) with the total features of the dataset, it was selected 9 of them (i.e., Read wait time, Write wait time, Disk Write operations, File system Shrinking, Input Bandwidth, Output Bandwidth, Input non-unicast packets, Random-access memory and Round trip average).

To study if these variables can improve results, we decided to do a Recursive Feature Selection (RFE) over the Transformer model. We made a recursive features selection to observe which patterns the covariates could influence positively and which covariates we should use. The transformer model was used over the TFT model because, in general, it showed slightly better results. Table 6.4 resumes this analysis and presents firstly the MAPE obtained without using covariates. Secondly, the MAPE is obtained using covariates. In cases where the usage of covariates decreases the MAPE error, it is presented which combination of covariates provides the lowest error. In cases where covariates did not improve the results, it was kept with -.

In the end, the transformer model made use of 3 different covariates (Random-access memory, Write wait time, and Round trip average). Using covariates decreased the general error by 2.14%, which was not as significant as we expected. Furthermore, except for the *Peaks workload* where the usage of RAM and RTA improved the results in 17.46%, in all other patterns where the use of covariates decreased error, the decrease was minimal and not so significant.

6.3 Servers usage versus Twitter analysis

Having already analyzed Twitter and server usage separately, we will now analyze these things together. For this, we create a simple dashboard present in Figure 6.5 where it is possible to compare some graphs of different variables over the same period. In this dashboard, there are three graphs. The first was plotted two lines with the average CPU usage, one regarding the first layer (blue line) and the other regarding the second layer (orange line). The second graph presents the sentiment over time with seven days moving average (like in 5.6), and the third graph presents in percentage the number of tweets

Table 6.4: Influence of covariates in the transformer model predictions. (- and + refer to workloads with low and high variation, respectively.)

Pattern	Best MAPE without covariates	Best MAPE using covariates	Metrics used
Constant workload (-)	3.40%	**3.02%**	RAM, WWT, RTA
Constant workload (+)	**8.19%**	8.19%	-
Down workload	**30.51%**	30.51%	-
Growing Up workload (-)	9.38%	**9.11%**	WWT, RTA
Growing Up workload (+)	16.88%	**16.01%**	RAM
Low-High change workload	45.64%	**41.11%**	RTA
High-Low change workload	**58.51%**	58.51%	-
Peaks workload	65.01%	**47.55%**	RAM, RTA
Random workload	**32.41%**	32.41%	-
UP and Down workload (-)	**36.28%**	36.28%	-
UP and Down workload (+)	**75.56%**	75.56%	-
Average	34.71%	**32.57%**	-

of each category (positive, negative, and neutral). This third graph also shows that the number of neutral tweets is usually more significant than the number of tweets in the other two categories. The objective of joining and analyzing together these three graphs is to understand if there is any relationship that is possible to see with the naked eye and could give us vital information to develop our final model.

However, looking at these graphs, we can see no evident and trivial relation. It can be seen that since the frequency of positive zones increases (second graph), it also matches the mean decrease in CPU usage in layer 1 (first graph). This may indicate that the most positive sentiment people have about the bank, the less necessity they have to go to the bank applications. Furthermore, it is also interesting to see that the increase in CPU usage in layer two around May 2021 (first graph) is attended by a rise in negative feelings (second graph). However, the same did not occur in June 2020, when the negative sentiment rose.

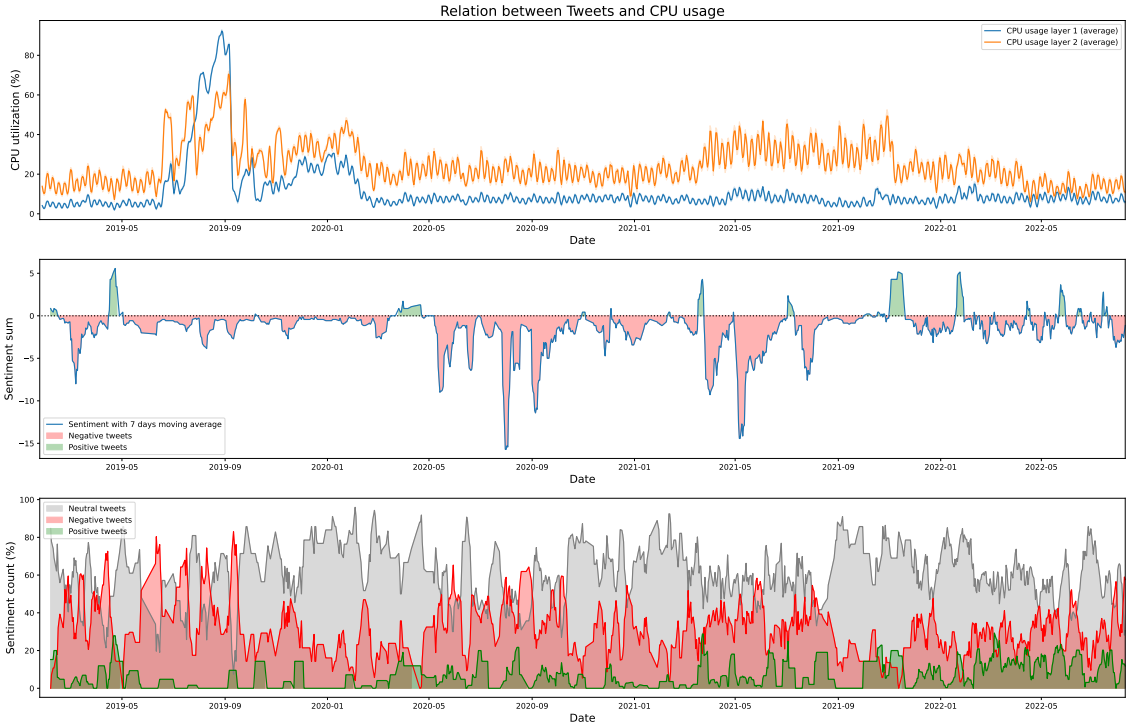


Figure 6.5: Dashboard to compare the average server's usage with tweets related to *novobanco*.

7

EAMDrift: Methodology and Results

This chapter presents the EAMDrift model, a new model created to achieve this project's requirements. We start by explaining how we created the EAMDrift, describe its architecture, and present the results of our experiments. Ultimately, we compare our model to the baseline models used in this work and validate the results in a different dataset.

In the previous chapters, we presented this work problem and objective, studied the essential materials and works to this project, presented how we obtained and analyzed the datasets, and provided an initial analysis of baseline models performance and datasets relations. With the robust information that we could obtain through all this analysis, we conclude that to have the best possible performance, our model should be an Ensemble because, for different models, different patterns yield better predictions (Table 6.2). Moreover, we wanted our solution to be interpretable because, in this way, we could automatically study the influence of human context and have more confidence and certainty in predictions. However, to study the influence of human context, this model has to accept the usage of covariates. We also wanted to add to our model a mechanism to retrain automatically and where we could combine different models' predictions to achieve a better final prediction. The main problem with our model requirements is between ensemble models and interpretability as to have an interpretable ensemble model, or we use interpretable models in our ensemble approach (which would decrease the accuracy), or we had to adapt the perspective of our approach.

7.1 Proposed model architecture

To meet our model requirements in a way that we could achieve the project objective, we propose EAMDrift. EAMDrift is a new machine-learning model that combines data preprocessing techniques, and interpretable and non-interpretable models, through an innovative model architecture. This model meets our requirements as it accepts covariates, is an ensemble, can automatically find points to retrain, combines all model predictions to achieve a better prediction, and is interpretable. However, to solve the trade-off between

interpretability and ensemble model, we changed our perspective a little and adjusted the interpretability from the prediction model itself to the ensemble part, where the models used to predict a certain point are chosen. Moreover, this model also has a mechanism that alerts for drift changes in the workload, providing an extra level of interpretability. This drift change detector will be used to alert and retrain the model.

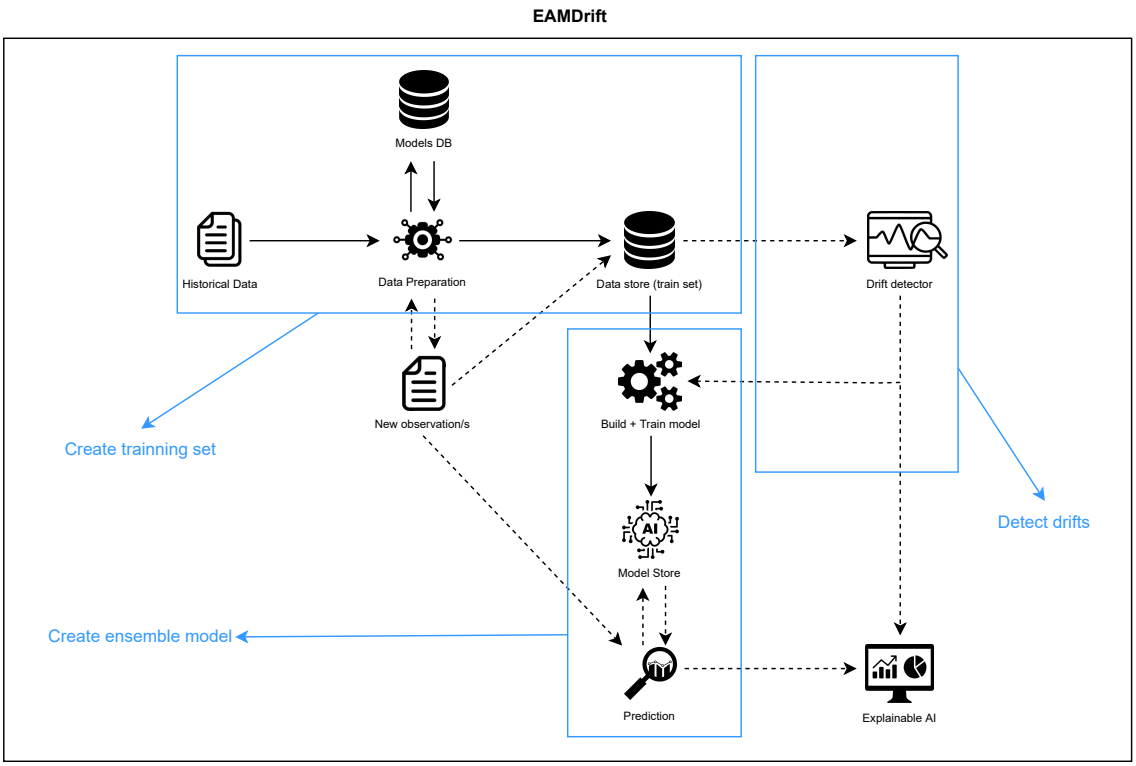


Figure 7.1: Model architecture

The architecture of our proposed model (EAMDrift) is depicted in Figure 7.1. Overall, this model has three main components, i.e., *Create training set*, *Create ensemble model*, and *Detect drifts*, that will be described briefly next.

- In the first step (*Create training set*), the model starts by using historical workload and covariates data to create a new training set that will be used to train our ensemble model. This historical data will be split into different sliding windows, each with the same size. Then, for each sliding window, we will: run different models (previously selected by the user) and find the best, extract statistics about that window (like mean, number of peaks, standard deviation, and others), and pre-process covariates data (i.e., for categorical covariates the model chooses the most frequent and for numerical the model sums that values for each window). In the end, we will have a train set with the columns: statistics, processed covariates, and best model. Each line of the train set will represent a sliding window.

- In the second step (*Create ensemble model*), the model will use the training set that was previously created and build an ensemble model. This ensemble model uses

RuleFit, an interpretable machine learning model based on rules as previously explained in 2.3.3 and will have as input the columns referring to statistics and processed covariates and as output the column with the best model. In each prediction, the RuleFit model will output probabilities for the selection of each model, and the final prediction will be a combination of the output of different models multiplied by their respective probabilities. Unlike the usual time-series models, our model does not use the time-series points directly to select the best models but instead uses statistics and related data.

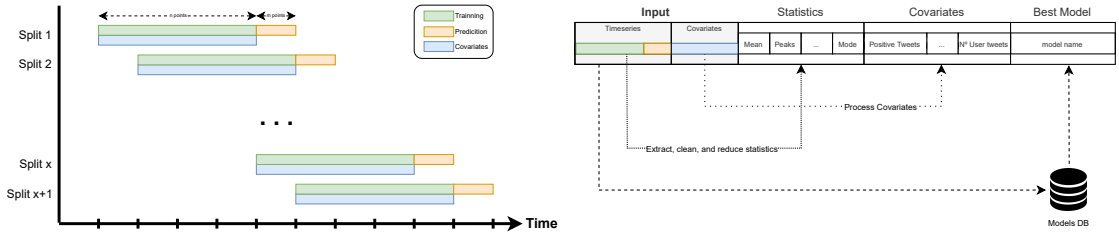
- The third step (*Detect drifts*) occurs during real time usage. When new points are given to the model, the model process this data to create an input in the necessary format of the ensemble model. At the same time, this processing is done, and the model tests if these new points trigger the drift detector and, therefore, the need to retrain the model. In the case of yes, the model retrains before giving the respective predictions.

Create initial training set

Before creating the training set, the user needs to make a crucial step: creating the Models Database. This database is the name of the model the user has chosen to use during training, and his model works with any model as long as it is implemented in the code. With this defined, the model will receive historical data to be trained as input. The model expects this data to be a time series DataFrame, where there is a column for dates, another for the variable we want to predict, and the rest regarding covariates (either numerical or categorical) that the user wants to add. Next, the pre-processing step will start, and the historical input will be divided into different splits, as demonstrated in Figure 7.2(a). The user needs to define the number of n training points and m prediction points, and the user can also select the number of splits he wants to create to train the model. In cases where the user does not select any number for the number of splits, the model will automatically find the maximum number of splits that can be created depending on the values of n and m variables. In the end, each split will have a part of training (time series of the variable user wants to predict), a part of prediction (to validate predictions for the respective split during training), and a part of covariates (time series of different covariates the user wants to use).

With the split ranges defined, the next step will be to process each split, as depicted in Figure 7.2(b). For each split, we will extract statistics from the training points, test models from the Models DB using either training and prediction points to validate forecasts, and finally format the covariates.

The statistics extraction was done using a Python library that can automatically compute numerous features of time series data called tsfresh [88]. tsfresh extracts around 800 statistics, from simple and more known ones to others more complex. However, handling such a large number of statistics would take apart all interpretability of the



(a) Splits of the historical data to train.

(b) Schema of each training set line.

Figure 7.2: Demonstration about how are historical data used to create the training set.

model as it would be many statistics to process. As a result, we use data pre-processing and feature reduction techniques to reduce the number of features. The model starts by removing the columns with more than 50% of null values, the columns with more than 95% feature similarity in values and variance, and the correlated columns. On average, these steps reduced the statistics number from 800 to 250. However, this continued to be a very high number to analyze, so we did an additional feature selection technique based on the ElasticNet regression. ElasticNet is a linear regression model that combines the properties of Lasso and Ridge Regression by adding L1 and L2 regularization terms. ElasticNet minimizes the objective function depicted in equation 7.1, where the first term of the equation is the MSE loss. The second term is the L1 regularization term, which encourages sparsity (more features being set to zero) by adding a penalty for non-zero weights. The third term is the L2 regularization term, which encourages small weights by adding a penalty for large weights. In the equation, y is the target sample, w is the weight vector, n is the number of samples, and p is the number of features. ElasticNet has two parameters to be defined, the $l1_ratio$ (ρ in the equation) and the α, to balance L1 and L2 regularization. A larger value for alpha means stronger regularization, and a larger value for $l1_ratio$ means more L1 regularization, which leads to sparsity (if $l1_ratio = 1$, only L1 regularization is used, and if $l1_ratio = 0$, only L2 regularization is used). Our objective in using ElasticNet is to select the most important features while simultaneously shrinking the less important, leading to a more interpretable model. The best parameters found for $l1_ratio$ and α were 0.7 and 0.9, respectively, leading to a feature reduction from 250 to 45 on average.

$$
\begin{aligned}
L_{enet}(w) &= \frac{1}{2n}\|y - X_w\|_2^2 + \alpha\rho\|w\|_1 + \frac{\alpha(1-\rho)}{2}\|w\|_2^2 \\
&= \frac{1}{2n}\|y - X_w\|_2^2 + \alpha\rho\sum_{j=1}^{p}|w_j| + \frac{\alpha(1-\rho)}{2}\sum_{j=1}^{p}w_j^2
\end{aligned}
\tag{7.1}
$$

To test models, for each split, we use the training points to train every model present in Models DB (in the case of models with parameters, in this phase, they have to be already defined) and predict the following m points. After this, the model that provides the best accuracy will be registered, as is demonstrated in Figure 7.2(b). Finally, the

format covariates step is the simplest of these three, and the objective is to aggregate the n covariates points in one line. For numerical covariates, we sum the values of all points, whereas, for categorical, we count the most common category in that period. The most common category will be after encoding with the $OneHotEncoder$ function from *scikit-learn* because the input for our ensemble model needs to be numerical. Our model can also add time covariates (i.e., day, month, and year). In cases where the user wants to add these covariates, our model adds the date from the last training point. For example, if a specific split's range of training points were from 01-01-2020 to 01-04-2020, the day, month, and the year chosen would be regarding the 01-04-2020 date.

At the end of these steps, we will have a training set with multiple lines, where each line will correspond to a split. In each line, we will have columns regarding statistics, columns regarding covariates, and a column that identifies the best model for the respective split.

Create ensemble model

With the training set created, the model is ready to train the ensemble part. Here we will train a model where the main objective will be to give probabilities to the models present in Models DB. This model will receive the columns regarding statistics and covariates from the training set and will output probabilities for each model based on the best model column. We selected the RuleFit model, previously presented in 2.3.3, adapting the output part to solve our probability task. RuleFit was chosen because it is an interpretable model that can be easily understood and explained. It generates a set of human-readable rules that can be used to make predictions.

After training, the RuleFit model will be stored to predict new points in the future. When a new point appears to predict the next m points, this point will be added to the data, and we will pre-process to create a new line similar to the ones created for the training set (but now without the *best column* column). This new line will have the same format as the ones given to the RuleFit model to train so that RuleFit can output a prediction. RuleFit will output an vector Y with the format $n * 1$ (7.2). n will correspond to the size of the Models DB, and each line of the matrix will be the probability of use for each model. Moreover, the probability values will be between 0 and 1, where a value closer to 1 indicates that it will have more impact on the final prediction.

$$Y = \begin{pmatrix} p_1 \\ p_2 \\ \dots \\ p_n \end{pmatrix} \tag{7.2}$$

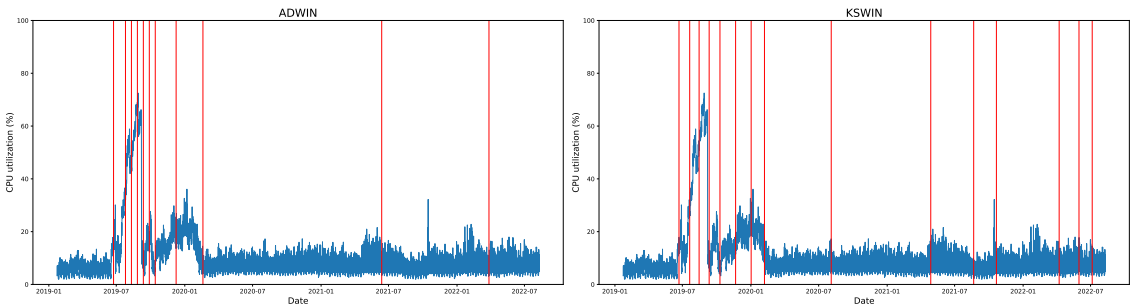
With the matrix Y obtained, our model will run every model present in Model DB with the training points of the split that is being predicted, creating a new matrix P with the same

format of Y, but in these cases with the model's predictions. The final prediction points will be predicted based on equation 7.3, where the matrices P and Y are multiplied.

$$pred(Y,P) = \sum_{i=1}^{n} Y_i * P_i \tag{7.3}$$

Drift detector

The final part of our model is the Drift detector. The objective of this component will be to find changes in the data distribution that may indicate the need for retraining. Moreover, this will also warn the user to be more careful when analyzing the workload in the following days. When a new point arrives in the model, the drift detector will be run, and if needed, the model will be retrained before giving the prediction. For this, we tested two concept drift detectors, ADWIN and KSWIN, that were previously described in section 2.3.4. These two approaches were tested in the same workload to understand the performance of each of them. The results are presented in Figure 7.3.



(a) Drifts detected using ADWIN with $\delta = 0,002$.

(b) Drifts detected using KSWIN with $\alpha = 0,0001$.

Figure 7.3: Examples of detected drifts using two different methods.

For both drift detectors, we tested different parameters, and the ones that provided better results were ADWIN with $\delta = 0.002$ (11 drifts detected) and KSWIN with $\alpha = 0.0001$ (15 drifts detected). Observing Figure 7.3(a), we can see that when abrupt changes happen, ADWIN detects many drifts, but when the changes are not so abrupt, the detection decreases. Thus, without the abrupt change of workload in the beginning, the drifts detected do not appear to perform well. Even when we change the value of the parameter δ, the drifts detected would increase, but many trash drifts would be added. However, this drift can provide good information when critical changes happen in the workload.

Conversely, KSWIN has an overall better performance. In the abrupt part of the workload, KSWIN do not detect so many drifts when compared to ADWIN. However, the detected drifts along the workload are more well dispersed when compared to ADWIN. KSWIN do not detect so many drifts in the abrupt change zones of the workload but seems to find drift in more critical zones. Moreover, KSWIN can detect drifts more easily in zones where the changes are not so abrupt, which is also an advantage for the objective

of our work. In general, KSWIN this drift can anticipate the workload changes, even the not critical ones, without adding drifts in zones that do not need one. As a result, for EAMDrift, we decided to choose as a drift detector the KSWIN.

In our method, to avoid that in zones where the workload changes a lot, the model is continuously retraining, a retrain interval can be defined. If this interval is defined as 14 days, the model must wait at least 14 days to be retrained again. Thus, if drifts are detected on 1, 7, and 20 days of a month, the model will retrain in the 1 and 20 days. Regarding day 7, the model will ignore that drift and continue as if it did not happen.

Model Interpretability

As previously mentioned, the interpretability in EAMDrift does not explain the prediction itself. Instead, the interpretability is in the ensemble and allows us to explain each model's choice. Firstly, we can observe which rules and features contribute more to the prediction. Secondly, knowing that this model uses a combination of different predictors to have the final forecast, EAMDrift allows us to know which models are contributing to a specific prediction and what weight was attributed to each of those models. Moreover, each rule has a support that is related to the number of points that satisfy a rule. Rules with more significant values will also allow us to understand which relations between input variables are present in our data. This will be particularly important to study the relations between the data center's workload and human context.

EAMDrift also has a drift detector implemented, and although the drift is here to detect moments of possible retraining, it also warns the user. This warning makes it possible to alert the user to be more attentive to the following periods.

Other details about EAMDrift

EAMDrift works with any model as long as the user implements it in the code. In cases where the user selects a model X, and during training, the model is just chosen as the best for one prediction, it will be discarded from the pole of the predictors because it did not classify enough points. However, if drift is detected later and the model needs to be retrained, this model X will have the opportunity to enter again in the pole of predictors.

Furthermore, our model allows us to select just a restricted number of predictors in each prediction. For example, suppose we have six models in the Models DB, and we select a restricted number of predictors equal to three in each prediction. In that case, the model will use only the three predictors with more probability of being used. There are more parameters that the user can be tunning, such as the size of the dataset (in case the user wants a fixed size training set), the number of training points, and covariates to be used to predict the next. This model allows the user to select a number of training points and a different number of covariates to be used. Also, the user can select between MSE, Mean absolute error (MAE), or MAPE, which metric wants to use to select the best models. Lastly, there is a parameter to define the train set length that restricts the number

of points used in each retrain (in our work, we maintained this parameter equal to 250, meaning that in each retrain, the model is going to use the points regarding the last 250 days).

7.2 Implementation of EAMDrift

EAMDrift was implemented in Python (version 3.9.12) on Anaconda using mainly `Pandas`[1], `NumPy`[2] and `scikit-learn`[3] libraries and for the machine learning models we used mainly `River`[4], `Darts`[5], and `statsmodels`[6] libraries. The training and inference of EAMDrift were executed in a 6-core Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz.

The creation of the training set was paralyzed through the function `Parallel` from `joblib`[7] to reduce the processing time. We used six jobs to take maximum advantage of the cores of the PC. Finally, the features were selected through `tsfresh`[8] software.

The machine learning models chosen for our ModelsDB component were the baseline ones except for the TFT. This model was not chosen because the results were very similar to the Transformer, and overall Transformer obtained better results. Furthermore, adding this model would increase the complexity, with no significant apparent gains. Regarding the five remaining models, Prophet and Exponential smoothing did not need parameters tunning, so they were used the same way they were tested for obtaining the baseline results. Regarding SARIMA, LSTM, and Transformer, this process was more difficult. As we can observe through table 6.1 for these three models, no configuration suited all patterns. As a result, we used different configurations for each model based on their performance, i.e., the configurations used for each model correspond to the configurations where those models were selected the best. For example, the Transformer was selected as best in *Constant workload (+)*, so only one configuration was used. However, LSTM was selected in both *Growing Up workload (+)* and *Peaks workload*, so the two configurations that correspond to these patterns were used. SARIMA is the model with more configuration as it was selected as best five times.

Ideally, we would like to use all of the different configurations found for these three models, but this would increase the computation to a level that would be impossible to achieve with the available computational resources. In Figure 7.4, we present the average time in minutes to train each model and, on top of each bar, the number of times the models take to train when compared to the Exponential Smoothing (the one that needs less time to train). Thus, it is easy to conclude that the models with different

[1] https://pandas.pydata.org/
[2] https://numpy.org/
[3] https://scikit-learn.org/stable/
[4] https://riverml.xyz/0.11.1/
[5] https://unit8co.github.io/darts/index.html
[6] https://www.statsmodels.org/dev/index.html#
[7] https://joblib.readthedocs.io/en/latest/index.html
[8] https://tsfresh.readthedocs.io/en/latest/index.html#

configurations also require more time to train, so using all the combinations of parameters in our model is not viable.

Figure 7.4: Time to train each model.

7.3 Model optimization

As we mentioned previously, EAMDrift have some parameters that need to be tuned. For this optimization, during this section, each test was run in all the 11 patterns previously selected and explained in 6.1. The tests were run in the same circumstances as the baseline models were tested (30-fold cross-validation approach with a 1-day forecast) to have a suitable model optimization for all the different patterns that occur among servers of our dataset. Furthermore, testing the parameters in all patterns allows us to compare the results with the baseline ones and understand how our model performs compared to the baseline ones.

The first optimization done was about discovering which metric we should choose to select the best models. This is, during the creation of the training set, we need to have a metric that will classify the errors of each model to discover which ones provide better results. For our work, we tested the three metrics implemented (MSE, MAE, and MAPE), and the results are presented in Figure 7.5, where we present the mean error of the predictions among the 11 patterns, concerning the metric selected. Observing this Figure, we can conclude that both MSE and MAPE provide similar results and are significantly better when compared to MAE. As a result, both MSE and MAPE could be chosen, so we decided to choose MAPE for our model.

With the metric chosen, we start to test the influence of different parameters in our predictions, both in terms of the ensemble model's accuracy and cross-validation error. Our proposed model accepts two values for the observation window sizes. One is regarding the observation window of the target variable, and the other is regarding the observation window of the covariates set. Thus, we can define an observation window for our target variable of 28 days and a 14 days window for our covariate set. To understand the effect of both windows, we started by testing the impact of our target variable window in predictions. For this, we tested five different observation windows such as 1, 7, 14, 28,

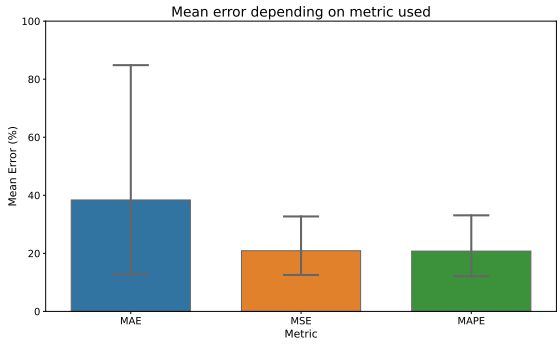
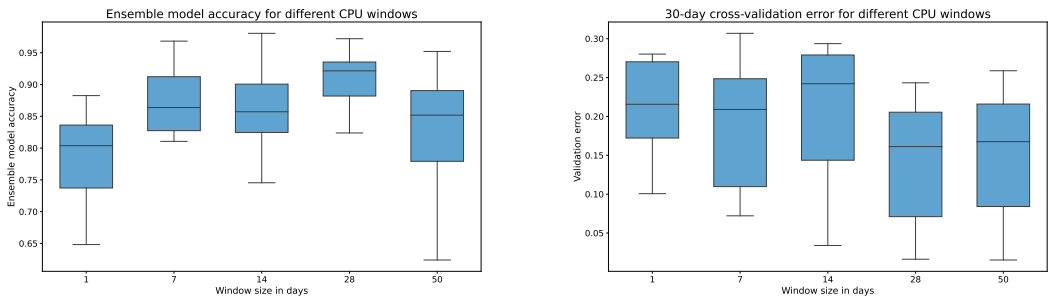


Figure 7.5: Mean error depending on the metric used.

and 50 days (note that each day has 4 points as the data is in 6 hours range), maintaining the covariates observation window fixed to 7 days (Figure 7.6). Observing the two graphs in the Figure, we can easily understand that the best window for the target variable is of 28 days, and it is also the one that provides lower cross-validation errors. A window of 7 days also provides a could accuracy for the ensemble model, but no such a good cross-validation error as of the 28 days. Moreover, having an observation window bigger than 28 days increases the ensemble model accuracy dispersion and the cross-validation error. Due to this, we select a 28 days window size for our target variable.



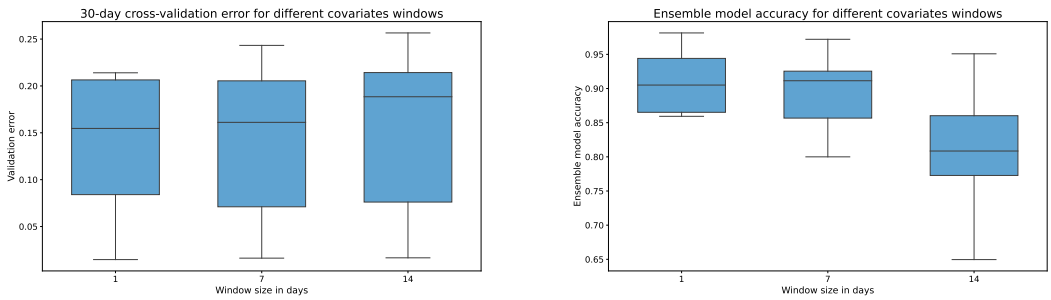
(a) Ensemble model accuracy.

(b) 30-day cross-validation error.

Figure 7.6: Accuracy and validation error of testing different windows sizes for CPU.

After having a good observation window, our next test is related to discovering the best observation window for the covariates set. We tested windows of 1, 7, and 14 days and the results are shown in Figure 7.7. For this test, the 1-day observation window was the one that provided better results in terms of the accuracy of the model, as it has a lower quartiles dispersion and an overall higher accuracy. When analyzing the cross-validation error, we can note that the errors do not change too much depending on the covariate's window size. However, we can notice that although the discrepancy is not significant, the 1-day window still provides lower overall errors.

Having an observation window for our target variable of 28 days and the covariates set of 4 days, we tested the EAMDrift with different sets of models to comprehend how



(a) Ensemble model accuracy.

(b) 30-day cross-validation error.

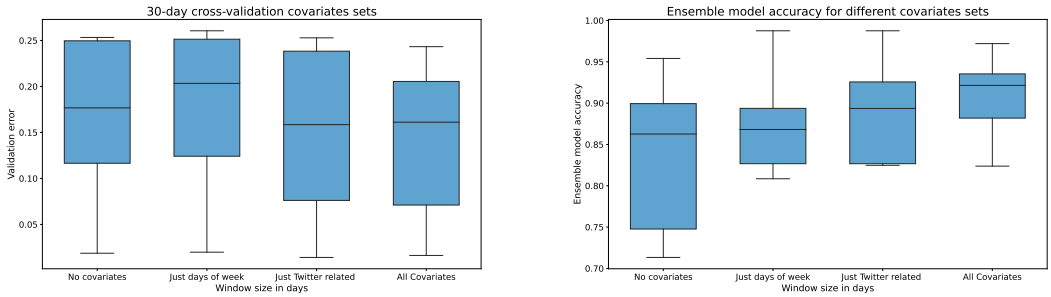
Figure 7.7: Accuracy and validation error of testing different windows sizes for covariates.

and which models influence the final predictions and if some models could be removed. Figure 7.8 presents the cross-validation errors of running our model with different sets of models (i.e., running with all models, just with the deep/machine learning models, and just using the best and the best three models). The main conclusion is that using all models provides significantly better results than using sets of models. When observing the results of using just deep and machine learning models separately, the differences are slight but with lower results than when using all models, meaning that combining either machine and deep learning models contributes to better overall predictions. Furthermore, another conclusion that these tests gave us is that using all predictors provides better results than just using the best predictors at each time. Observing the last two box plots of Figure 7.8, it stays clear that using the best predictor or the three best predictors in each prediction decreases the model's overall performance. This indicates that combining different predictors in the way EAMDrift does can bring real advantages to the general performance of the model.

Figure 7.8: Testing different model configurations.

Lastly, we studied the influence of covariates in our model. For our tests, there are two possibilities of covariates: dates and tweets. Dates are covariates with the day, month, and year of the last day before the forecast. Tweets contain seven features (number of positive/negative/neutral tweets, the overall sentiment in a day through our ensemble

method (presented on 5.3.3), the topic of tweets, and the total number of tweets either regarding news and all users). We tried our model using all covariates (i.e., days of the week and Twitter), with just one type of covariates and without no covariates (Figure 7.9). Analyzing both graphs, it is trivial that it is not complicated to detect that the model that uses all available covariates provides better ensemble model accuracy and lowers training errors. It is also interesting to note that the model that provides worse results is the model that does not use covariates, suggesting that covariates influence predictions.



(a) Ensemble model accuracy.

(b) 30-day cross-validation error.

Figure 7.9: Influence of covariates in our proposed model.

7.3.1 Results

After performing the necessary parameters optimizations in EAMDrift, we tested the model with the best parameters in the 11 patterns to obtain the final predictions to be compared to the ones provided by the baseline models. Table 7.1[9] shows the errors of the best baseline model and our proposed model, whereas the lowest errors for each pattern are highlighted in bold.

The first observation that can be made is that overall our model can outperform the best baseline for each pattern. Our model could improve the predictions, except for the *High-Low change workload* and *Random Workload* patterns where the baselines had a slightly lower error. This suggests that our model can automate the task of choosing the best model for each prediction. Moreover, we can also conclude that, on average, our model achieves a lower MAPE and MSE error than the best baselines for each pattern. Regarding the MAPE error, our model achieved a minor error by 2.31%, and regarding MSE, our model also achieved a lower error by 47.4078. This demonstrates that combining predictions of different models through our ensemble model can improve the final predictions.

Furthermore, recalling Table 6.3 from preciously Chapter, where we analyzed the overall individual performance of the baseline models along all patterns, and, examining

[9]The best baseline model name is not presented in this Table to be easier to interpret, but at the end of this chapter, we present Table 7.5 with detailed information about each baseline model performance.

Table 7.1: Errors of the best baseline model and our proposed model for each workload pattern.

Pattern	**Best model error**		**Proposed model error**	
	MAPE	**MSE**	**MAPE**	**MSE**
Constant workload (-)	1.5%	**0.0013**	**1.47%**	0.0017
Constant workload (+)	8.19%	32.82	**6.51%**	**1.7245**
Down workload	14.34%	1.2945	**9%**	**1.1502**
Growing Up workload (-)	4.96%	**8.7361**	**3.84%**	10.3266
Growing Up workload (+)	11.36%	10.8438	**8.52%**	**5.7609**
Low-High change workload	18.61%	34.8263	**15.47%**	**24.1448**
High-Low change workload	**15.32%**	**18.8123**	20%	27.6671
Peaks workload	61.94%	661.0882	**49.02%**	**215.7311**
Random workload	**18.37%**	**39.3044**	19.93%	61.055
UP and Down workload (-)	24.72%	36.8096	**20.89%**	**18.6455**
UP and Down workload (+)	16.75%	96.6057	**15.96%**	**53.4495**
Average	17.81%	85.5584	**15.5%**	**38.1506**

it with the results of Table 7.1, it is possible to infer that our model outperforms every baseline models individually by 15% to 25%.

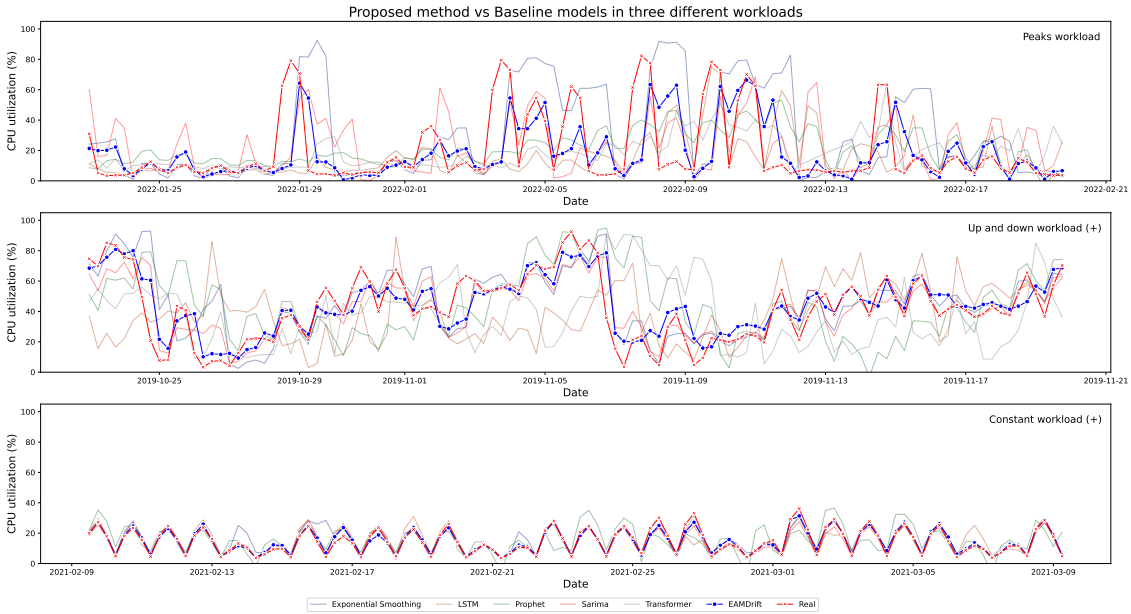


Figure 7.10: Proposed method vs. baseline models in three different workloads.

After analyzing the results provided by our proposed model, next, we provide a visual analysis of how our model is behaving when compared to the baseline ones. We selected three different patterns that are interesting to analyze. In Figure 7.10, we present the predictions of our proposed model proposed and of the baseline models used in this work. Each graph in this Figure corresponds to a pattern identified in the top right corner and

two thickest lines, one in red with the real values and one in blue with the predictions made by our model. The remaining lines correspond to the baseline models.

Although this image can not be visualized with a high level of detail with the naked eye, we can conclude that our model provides better results when compared to the baseline in all three patterns. Even in the two first patterns, where we can note the high presence of burst, our model can adapt well. Furthermore, in the third graph, where a cyclic pattern is presented, all the models provide a good forecast, except for the Prophet, which had some parts that failed to yield reasonably (we can note lines in green that do not follow the trend). Observing these three patterns, our model can continuously adapt well to the burst without losing accuracy in simple patterns.

7.4 Simulation scenario and model interpretability

Having the model optimized for all patterns, we will test our model as it was in a real situation, where it will have more points to predict and will need to retrain through the drift detector. Moreover, we will also use this analysis to study the model's interpretability and explain how we can understand the relations between servers and Twitter. For this, we will use a workload with the average of the first layer and the average of the second layer. These two workloads bring advantages when studying relations between data center usage and human context as it will not have biased results of fails or errors that could have occurred in a single server.

First layer analysis

Analyzing the first layer workload, Figure 7.11 presents the true workload (blue and orange), the predictions of EAMDrift (green), and the moments where the model retrained (red). In this workload, the model has been retrained 16 times (16 drifts detected with at least 14 days interval). For this workload, EAMDrift used 23 features (13 concerning workload statistics, 7 concerning the Twitter analysis, and 3 concerning day, month, and year).

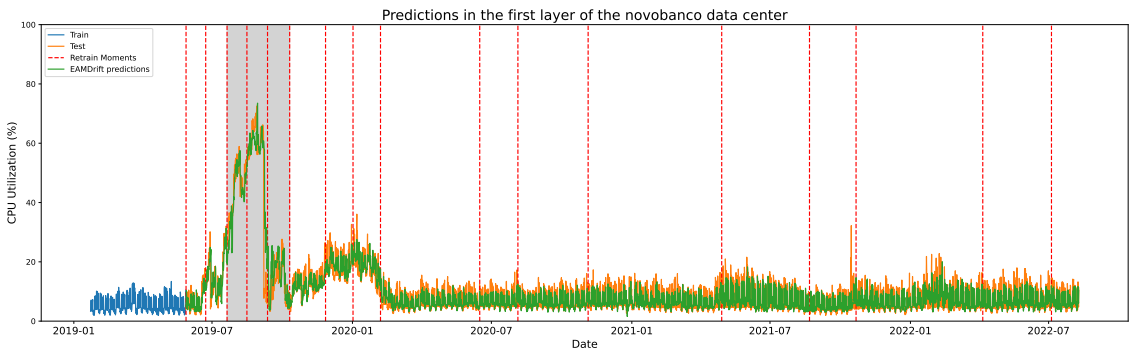


Figure 7.11: Predictions in a simulations scenario of the workload average of the first layer.

Concerning interpretability, this model offers a drift detector that is easy to interpret visually and the rules created by the ensemble model. Along this pattern, around 1600 rules were generated in each train, but as a consequence of LASSO regression, on average, just 300 had an assigned weight. Moreover, to explain how our model combines individual predictions, we will analyze in detail the highlighted part in Figure 7.11, which is an exciting part of this workload. In Figure 7.12, we can observe the highlighted workload with a more extensive viewing perspective in the first graph and the weights assigned to each model to obtain the final prediction in the second. (The Transformer does not appear because it was discarded during training due to not being considered the best at any point, so the multi-class classification can not use it). Looking at Figure 7.12, we can note that while the workload is growing, the SARIMA model is highly participating in the final predictions. When the workload goes down, SARIMA loses importance, whereas the remaining models take a contrasting path.

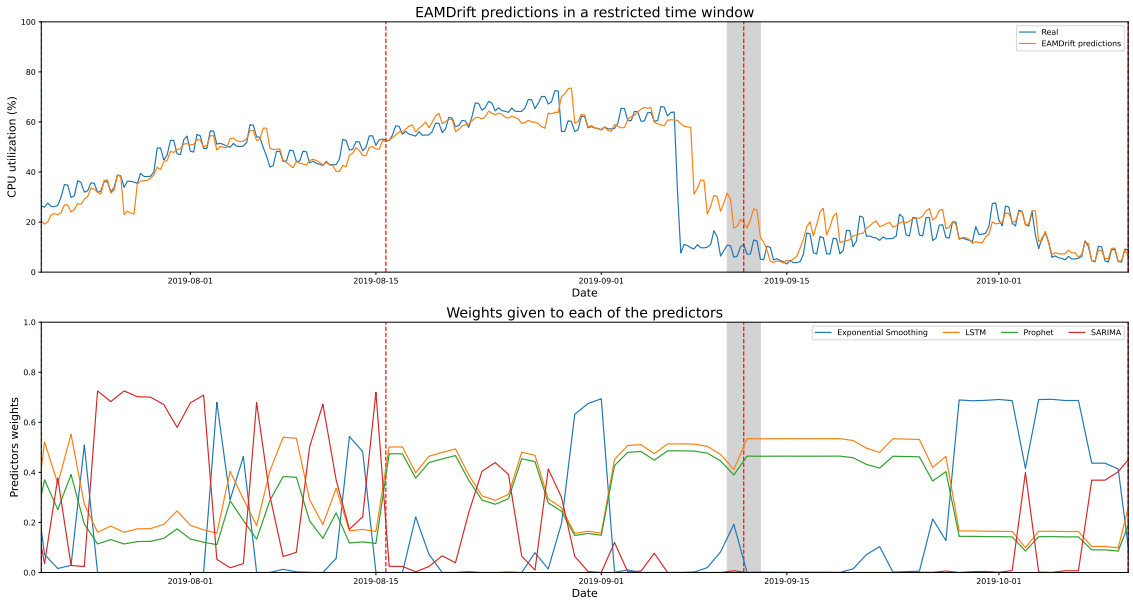


Figure 7.12: Detailed analysis of the highlighted part in Figure 7.11.

To study the rules created by our model, we have chosen the retrain point highlighted in Figure 7.12. The rules that will be analyzed next were created based on points before 11 September 2019 (the retraining date). Table 7.2 shows three rules with the highest coefficient. The first rule relates the number of points above the mean with the number of consecutive changes in the workload. It is a rule that relates to what happens before the retrain, as the workload changes fast. The second and third rules can suggest some bad feelings about the bank before the rules generation point, as changes in the workload are related to the `number_negative_tweets` higher than 14 and `sentiment_final_ensemble` lower than 2.72. (`sentiment_final_ensemble` is our method for classifying tweets sentiments presented in 5.3.3).

Lastly, it is also possible to analyze the importance of the individual features where the

Table 7.2: Rules with the highest coefficient for the first layer workload.

	Rules	Coefficient
Rule 1	count_above_mean <= 52.5 **AND** absolute_sum_of_changes > 324.61	6.09
Rule 2	sentiment_total_degree <= 2.72 **AND** all_users_tweets_count > 1238.0 **AND** absolute_sum_of_changes > 324.61	4.13
Rule 3	variance <= 7.216586351394653 **AND** sentiment_final_ensemble <= 6.0 **AND** number_negative_tweets > 14.0	3.13

most important concerning Twitter and workload statistics were `number_neutral_tweets` and `sum_values`, respectively. `number_neutral_tweets` had an importance of 0.1, and `sum_values` is a statistic that computes the sum over the time series values and has an importance of 0.25.

Second layer analysis

Passing to the second layer workload analysis, EAMDrift used 36 features (more 13 statistics than the ones used in the first layer). Similar to what was done for the first layer, in Figure 7.13, we present the true workload, the predictions of EAMDrift, and the moments where the model retrained.

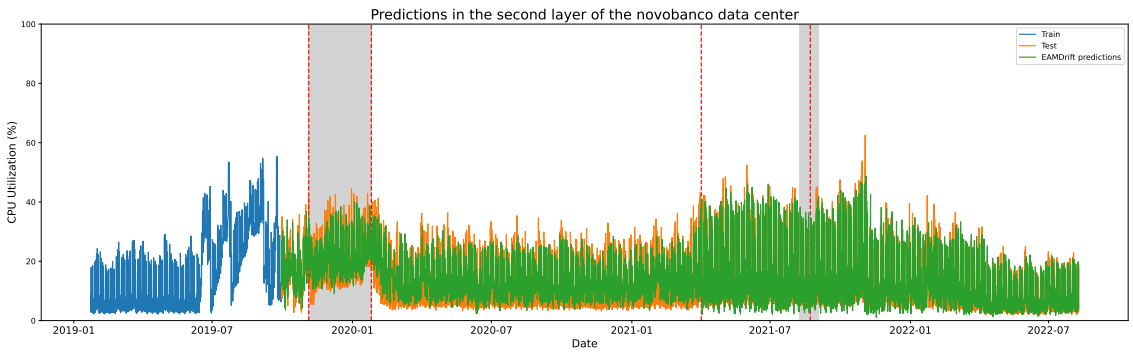


Figure 7.13: Predictions in a simulations scenario of the workload average of the second layer.

Along this pattern, around 1800 rules were generated in each train, but as a consequence of LASSO regression, on average, just 200 had an assigned weight. Again, we highlighted a part of Figure 7.13 to analyze it in more detail (between the first and second drift detected). The results are presented in Figure 7.14 where in the first graph, we can note the cyclic pattern, and in the second, the contributions of each model for the final predictions. LSTM, Prophet, and Transformer have a similar contribution along this window, while SARIMA and Exponential Smoothing change between them. When the contributions from SARIMA grow, the Exponential Smoothing one's decrease, and the

same vice-versa. In zones with more variation, SARIMA has significant importance, while in others, Exponential Smoothing is the one that contributes more.

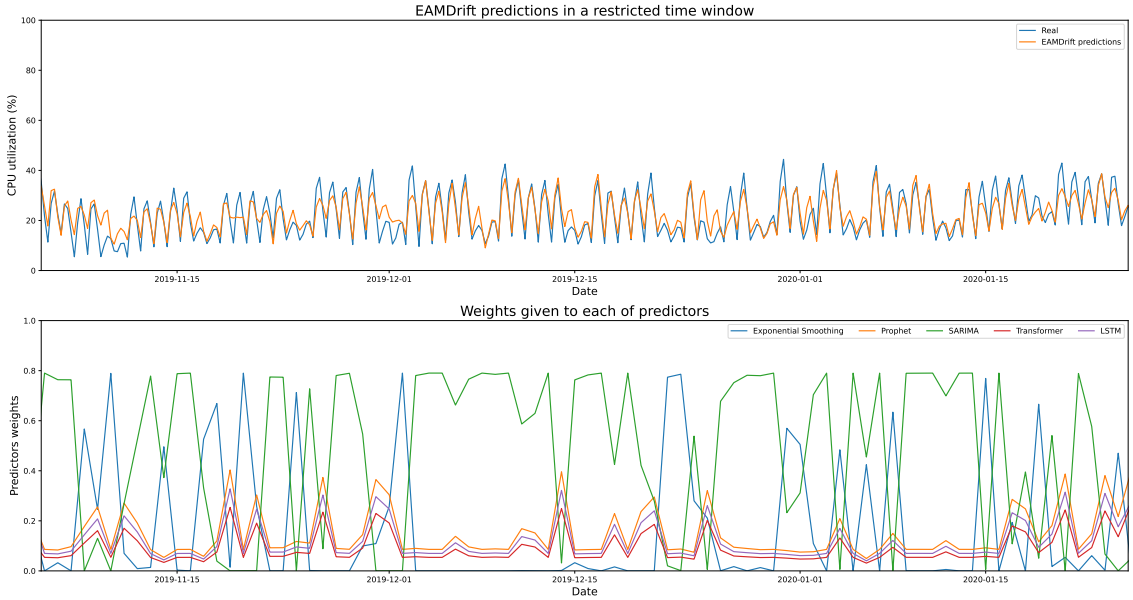


Figure 7.14: Detailed analysis of the highlighted part in Figure 7.13.

For this workload, to analyze the rules, we selected the last drift detection point (highlighted drift in Figure 7.13). Table 7.3 presents the three obtained rules with the highest coefficients. All rules are related to the variance in the data. This can be related to the growth and decrease of distribution in data. Compared to the first layer, there are no such understandable rules related to Twitter. The second layer has the function of processing, so this can be the reason why it is challenging to find relations.

Table 7.3: Rules with the highest coefficient created by the ensemble model for the second layer.

	Rules	**Coefficient**
Rule 1	all_users_tweets > 26.0 **AND** skewness > 0.64 **AND** kurtosis <= 0.08 **AND**	2.68
Rule 2	asymmetry statistic > 24.44 **AND** longest_strike_below_mean > 17.5 **AND** sentiment_final_ensemble <= 0.08	2.61
Rule 3	mean_abs_change > 6.665 **AND** sum_values > 4417.63 **AND** sentiment_final_ensemble <= 34.0 **AND** number_peaks_n_20 > 18	2.31

At last, the individual features with more importance regarding Twitter and workload statistics were `sentiment_final_ensemble` and `number_peaks_n_20`, respectively. The

`sentiment_final_ensemble` had an importance of 0.23, and `number_peaks_n_20` is a statistic that represents the number of peaks in a window of 20 points and has an importance of 0.2.

7.5 Validation of results

Our models already obtained results bring good perspectives for future usage. Regarding the *novobanco* data center, we have already seen that our model can combine predictions from different models to achieve a better one. However, to identify our model's real weaknesses and how it is generalizing, it is essential to validate the model in other datasets. To finalize our study about EAMDrift, we decided to provide validation in different environments and datasets. More than just analyzing the results in the *novobanco* data center workloads, we tested our model in another dataset from the Google Cluster Trace (GCT) [89].

Google Cluster Trace

The GCT is a set of workload data observed across 8 Google Borg computing clusters for May 2019. A Google Borg is a system that works as a cluster manager. It can execute hundreds of thousands of jobs from several different applications, where these executions are divided into different clusters (each cluster has thousands of physical machines). Moreover, a cell is a set of machines with the same cluster manager, and the Google Cluster Trace dataset has different cells. For our investigation, only one cell was used, named *2019-05-c*, and whose time zone is in America/New York (UTC-05:00). In this cell, the data have a 5-minute step for the entire month of May and provides CPU and Memory metrics (just CPU will be used). For this work, we processed this data, filled in the missing data, and grouped data in hours, giving us 744 entries (1 per hour). In Figure II.14, presented in Annex II, we provide a graph that shows the workload of the Google Cluster cell chosen.

Tests setup

We tested each workload with our proposed model, with Exponential Smoothing and with LSTM, which were the best machine and deep learning models found. We also tested with a variation of our model that can be seen as a non-interpretable EAMDrift model. Recalling the Kim *et al.* and Iqbal *et al.* works [24, 25] explained in the related work section, they have also created ensemble models to predict workload. The first work used RDF, and the second a SVM. As the code is not publicly available, we cannot directly test the performance of each of their models on ours. Instead, we adapted our proposed model and created two news that, instead of using RuleFit as the ensemble model, use a RDF model and another the SVM model[10]. The objective is to compare the effectiveness

[10]The implementations used for these models were from *scikit-learn*.

of RuleFit with other black box models acting as our ensemble selector.

The parameters of Exponential Smoothing and LSTM were tuned in the same way we adjusted the baseline models in Chapter 6, while in the non-interpretable EAMDrift, the parameters were kept equal to the original EAMDrift. For the *novobanco*, we forecast one day (meaning forecast 4 points at each time), and for the GCT dataset, we predicted 6 hours (representing forecast 6 points at each time). A 6 hours prediction for the GCT workload was chosen to make the prediction task more challenging since the workload does not have many anomalies (Figure II.14).

For each dataset, we have used 30% to train and the remaining to forecast. Regarding novobanco datasets, there were given around 1500 points (375 days) to train and 3500 points (875 days) to cross-validate, and for the GCT workload, 223 (9 days) points were given to train and 525 (21 days) to cross-validate. Due to the dataset from GCT not being in percentage, we standardized the data before testing the models. Moreover, regarding EAMDrift methods, for *novobanco* datasets were used Twitter and data covariates and for the GCT dataset, just date covariates were used.

Results

The results are presented in Table 7.4 (*novobanco 1* and *novobanco 2* concerning the average server from layers 1 and 2, respectively). Compared to EAMDrift and its variations, the baseline models have significantly higher errors. A reason for this is that even though predictions were made by training the model with data until the prediction point, the parameters in these models are just tuned once (in the beginning), which will probably worsen results over time.

Regarding all variations of EAMDrift, the difference in the average MAPE among all datasets was less than 1%, and for the MSE, it was less than 4. Our proposed model provided lower errors in two of the three datasets compared to the SVM version. Compared to the RDF version, our proposed model had higher errors for all datasets, even though the difference was low. Through this study, using non-interpretable versions of EAMDrift model could improve outcomes, but not so significantly to assume they are better for this problem. Thus, interpretability is a viable solution to be adopted as do not have a high loss of accuracy and provides additional mechanisms that can be useful for some tasks.

Table 7.4: Validation of our proposed model.

Method	**novobanco 1**		**novobanco 2**		**Google Cluster Trace**		**Average**	
	MAPE	MSE	MAPE	MSE	MAPE	MSE	MAPE	MSE
SARIMA	30.45%	19.41	49.11%	35.34	38.56%	390.97	39.37%	148.57
LSTM	33.82%	21.06	40.99%	27.86	57.77%	216.12	44.19%	88.35
EAMDrift (SVM)	**16.92%**	**3.58**	26.87%	14.61	28.19%	124.41	23.99%	47.53
EAMDrift (RDF)	18.46%	3.87	**25%**	**13.94**	**27.32%**	**114.47**	**23.59%**	**44.09**
EAMDrift	19.87%	4.29	26.77%	14.24	27.80%	123.41	24.81%	47.31

Table 7.5: Errors of baseline models and our proposed model for each workload pattern.

Pattern	Exponential smoothing		LSTM		Prophet		SARIMA		Transformer		EAMDrift	
	MAPE	MSE	MAPE	MSE	MAPE	MSE	MAPE	MSE	MAPE	MSE	MAPE	MSE
Constant workload (-)	1.69%	0.0027	2.68%	0.0157	3.75%	0.0109	1.50%	**0.0013**	3.40%	0.0178	**1.47%**	0.0017
Constant workload (+)	13.36%	3.8438	10.76%	5.9625	30.78%	15.0991	8.46%	3.5764	8.19%	32.8295	**6.51%**	**1.7245**
Down workload	35.81%	2.7445	20.82%	3.3525	78.86%	5.9827	14.34%	1.2945	30.51%	10.3593	**9%**	**1.1502**
Growing Up workload (-)	4.97%	**8.7361**	5.57%	17.4672	9.41%	43.3950	12.96%	52.1773	9.39%	43.3903	**3.84%**	10.3266
Growing Up workload (+)	18.18%	40.5569	11.36%	10.8438	24.72%	28.8096	32.70%	34.6690	16.88%	18.0637	**8.52%**	**5.7609**
Low-High change workload	18.61%	34.8263	41.09%	190.8459	25.39%	51.9550	22.89%	30.7193	45.64%	194.5346	**15.47%**	**24.1448**
High-Low change workload	34.08%	39.6715	50.38%	67.9875	47.03%	127.3723	**15.32%**	**18.8123**	58.52%	98.9778	20%	27.6671
Peaks workload	116.42%	913.2009	61.95%	661.0882	84.55%	463.5740	162.85%	221.1341	65.00%	629.4878	**49.02%**	**215.7311**
Random workload	36.38%	41.5569	32.13%	74.6546	61.27%	185.6023	**18.37%**	**39.3044**	32.41%	79.3801	19.93%	61.055
UP and Down workload (-)	25.10%	22.0930	29.12%	68.1501	24.72%	36.8096	98.85%	159.1541	36.28%	50.6637	**20.89%**	**18.6455**
UP and Down workload (+)	39.23%	182.0562	67.91%	433.2557	76.72%	669.0725	16.75%	149.1833	75.57%	336.6757	**15.96%**	**53.4495**
Average	31.26%	117.2081	30.34%	278.8424	42.47%	147.9712	36.82%	64.5478	34.71%	135.8527	**15.5%**	**38.1506**

8

Conclusion

In recent years, machine learning has experienced tremendous growth, and companies are increasingly adopting new mechanisms to automate tasks and improve their operations. With the growth of data generated in every business sector, machine learning is a powerful tool for extracting valuable insights and performing difficult or impossible tasks for humans. Data centers do not escape this trend, and the growth of cloud computing makes workload forecasts critical to future management mechanisms. Predicting the number of operations done in the next moments can improve performance, provide better client services, increase reliability, and reduce costs (equipment and energy). Nowadays, the research and academic industry are making a considerable effort to bring automation to data centers. Works show that a single workload predictor is insufficient to handle all patterns that the network can have, so ensemble approaches have provided better results (Section 2.1.4.2).

Moreover, a new machine learning branch related to interpretability is growing. In critical areas, such as healthcare or finance, where the consequences of incorrect predictions can be meaningful, forecasts alone are not enough. Instead, explanations for those predictions are required to give confidence and safety when taking actions based on predictions. Also, interpretability in the model can explain how each variable interacts to produce the final result.

With all these in mind, this work aimed to predict data center usage, add interpretable mechanisms, and study relations between data centers and human context. As a result, we proposed a novel machine learning method called EAMDrift. EAMDrift is an ensemble method that combines multiple predictions from individual models chosen by the user (EAMDrift works with any individual predictor). Have interpretable mechanisms that allow us to understand relations between the target variable and covariates (in our work between CPU and tweets) and why the model assigns each weight to each individual predictor. Finally, our model innovates compared to the state-of-the-art analyzed ones, having an intelligent mechanism to find points to retrain (drift detector) that also alarms the user that some change in the workload trend can happen in the following days.

8.1 Main Contributions

The main contributions of this work were:

- A multi-model method that automatically selects the bests predictors at each point. Our model assigns weights to each predictor using a multi-class regression, where each class corresponds to a method. Results show that our proposed model has high accuracy in different workloads patterns;
- A model that accepts past covariates (either numerical or categorical). Our model can also automatically add covariates related to dates (day, month, and year);
- A novel multi-model method that has interpretable mechanisms. Also, our proposed model, more than predicting, allow to study of relations between the target and covariates variables (in this work, between CPU and tweets);
- A method based on statistics to be easier to interpret.
- A retrain method that strategically finds potential points to model be retrained;
- A comparison and study between our proposed model with others in different datasets (from *novobanco* data center and Google cluster).

In this work, we have conducted comprehensive evaluations on different workloads with different models. The results show that our model outperforms the baseline ones in 15% to 25% either in *novobanco* data center and Google cluster datasets. Compared to the best ensemble non-interpretable models from the ones we analyze, our model achieved a slightly higher error (1-3%). Thus, this work suggests that interpretable machine learning models can be a viable solution for data center workload prediction.

Moreover, through our NLP analysis, we could conclude that most Tweets were related to financial problems and were mostly negative, but no relation was found by the naked eye. When applied to EAMDrift, with the tests conducted, covariates related to the human context could improve the ensemble model accuracy and, thus, the final predictions. However, we did not find any rule that could indicate a correlation between usage and human context that remains along an entire single workload. Instead, we could note some interesting relations in some parts of workloads. Analyzing results regarding the first layer, some connections between server usage and human context were found, but for the second, that was more challenging. The first layer deals with the front-end part of applications, while the second has the function to process operations, and that can be a reason not to find trivial relations.

While further research is needed to understand this method's capabilities and limitations fully, the initial results were promising. EAMDrift has the potential to make significant contributions to the field of machine learning and can be tested in a wide

range of areas, for example, predicting a company's stocks by taking into account the news.

Finally, our methodology is easy to implement, and we believe it has the potential impact on future research and applications. The main code used in this thesis is publicly available on https://github.com/gfMateus99/Master_Thesis and a more detailed view about EAMDrift, its usage and methods can be seen in https://github.com/gfMateus99/EAMDrift.

8.2 Limitations and future work

The presented work had some limitations that should be taken into consideration. The first and the one that impacted this work more was the computational resources that we had available. Since we work with confidential data, it was impossible to run our models on any computer with more processing and graphic capacity, which would accelerate the velocity of the tests.

Furthermore, the work made in this thesis can be a starting point for developing future work related to interpretability in this area. Firstly, creating an environment that could automate our model's processes would be interesting. Our suggestion would be to assign a part of a server to this task, where the server would aggregate the metrics from the data center and tweets related to *novobanco* and every day would automatically run the model to predict the following points (could be even 1-day predictions or multiple-days predictions to have different forecasts). This could be a precaution mechanism for the *novobanco* data center management teams.

Secondly, we would like to have more datasets with a good range of days to test the performance of EAMDrift in different environments (actually, few datasets are publicly available, and the ones available have a small range of days). Even the google cluster dataset does not have a complex workload to predict with limited tests. Lastly, instead of studying the relations between data center usage and tweets, it would be interesting to explore the relationship between data center usage and the stock market where publicly company stocks are traded. The stock market is also an essential indicator of the overall health of a company's economy, as the stock market's performance is often seen as a reflection of the general company events [90]. However, any part of *novobanco* company is listed on the stock market, which makes this analysis unfeasible.

Bibliography

[1] *17 reasons cloud computing is growing like crazy*. Last visited on February 2022. URL: https://www.hso.co.uk/blog/cloud/17-reasons-cloud-computing-is-growing-like-crazy (cit. on p. 1).

[2] M. Armbrust et al. "A view of cloud computing". In: *Communications of the ACM* 53.4 (2010), pp. 50–58 (cit. on p. 1).

[3] G. Aggarwal. *Council post: How the pandemic has accelerated cloud adoption*. Last visited on February 2022. 2021-12. URL: https://www.forbes.com/sites/forbestechcouncil/2021/01/15/how-the-pandemic-has-accelerated-cloud-adoption/ (cit. on p. 1).

[4] V. K. Jayakumar et al. "A Self-Optimized Generic Workload Prediction Framework for Cloud Computing". In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2020, pp. 779–788 (cit. on pp. 1, 9, 13).

[5] W. Iqbal, J. L. Berral, D. Carrera, et al. "Adaptive sliding windows for improved estimation of data center resource utilization". In: *Future Generation Computer Systems* 104 (2020), pp. 212–224 (cit. on pp. 2, 11–13).

[6] N. R. Herbst, S. Kounev, and R. Reussner. "Elasticity in cloud computing: What it is, and what it is not". In: *10th international conference on autonomic computing (ICAC 13)*. 2013, pp. 23–27 (cit. on p. 2).

[7] M. Pandey. *AWS Scaling (reactive vs proactive vs predictive)*. 2018-11. URL: https://medium.com/@damadhav/aws-scaling-reactive-vs-proactive-vs-predictive-2701ad6d48c9 (cit. on p. 2).

[8] *What is a data center?* 2022-05. URL: https://www.cisco.com/c/en/us/solutions/data-center-virtualization/what-is-a-data-center.html (cit. on p. 5).

[9] *Data Center Networking*. Last visited on January 2022. URL: https://www.manageengine.com/network-monitoring/data-center-networking.html (cit. on p. 5).

[10] H. Qi et al. "Data center network architecture in cloud computing: review, taxonomy, and open research issues". In: *Journal of Zhejiang University SCIENCE C* 15.9 (2014), pp. 776–793 (cit. on p. 5).

[11] T. Wang et al. "Rethinking the data center networking: Architecture, network protocols, and resource sharing". In: *IEEE access* 2 (2014), pp. 1481–1496 (cit. on p. 5).

[12] H. Dong et al. "Next-Generation Data Center Network Enabled by Machine Learning: Review, Challenges, and Opportunities". In: *IEEE Access* (2021) (cit. on p. 8).

[13] M. Xue et al. "Machine learning security: Threats, countermeasures, and evaluations". In: *IEEE Access* 8 (2020), pp. 74720–74742 (cit. on p. 8).

[14] A. A. Bankole and S. A. Ajila. "Cloud client prediction models for cloud resource provisioning in a multitier web application environment". In: *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*. IEEE. 2013, pp. 156–161 (cit. on p. 9).

[15] S. Islam et al. "Empirical prediction models for adaptive resource provisioning in the cloud". In: *Future Generation Computer Systems* 28.1 (2012), pp. 155–162 (cit. on p. 9).

[16] *Time series analysis: Definition, types, techniques, and when it's used*. Last visited on February 2022. URL: https://www.tableau.com/learn/articles/time-series-analysis (cit. on p. 9).

[17] W. Fang et al. "Rpps: A novel resource prediction and provisioning scheme in cloud data center". In: *2012 IEEE Ninth International Conference on Services Computing*. IEEE. 2012, pp. 609–616 (cit. on pp. 9, 13).

[18] R. N. Calheiros et al. "Workload prediction using ARIMA model and its impact on cloud applications' QoS". In: *IEEE transactions on cloud computing* 3.4 (2014), pp. 449–458 (cit. on pp. 9, 13).

[19] X. Wang et al. "Online cloud resource prediction via scalable window waveform sampling on classified workloads". In: *Future Generation Computer Systems* 117 (2021), pp. 338–358 (cit. on pp. 9, 11).

[20] G. Lai et al. "Modeling long-and short-term temporal patterns with deep neural networks". In: *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 2018, pp. 95–104 (cit. on p. 9).

[21] T. Khan et al. "Workload forecasting and energy state estimation in cloud data centres: ML-centric approach". In: *Future Generation Computer Systems* 128 (2022), pp. 320–332 (cit. on pp. 9, 13).

[22] K. Zhu et al. "Differentiated transmission based on traffic classification with deep learning in datacenter". In: *2020 IFIP Networking Conference (Networking)*. IEEE. 2020, pp. 599–603 (cit. on pp. 9, 13).

[23] F. Estrada-Solano, O. M. Caicedo, and N. L. Da Fonseca. "Nelly: Flow detection using incremental learning at the server side of sdn-based data centers". In: *IEEE Transactions on Industrial Informatics* 16.2 (2019), pp. 1362–1372 (cit. on pp. 9, 13).

[24] W. Iqbal et al. "Adaptive prediction models for data center resources utilization estimation". In: *IEEE Transactions on Network and Service Management* 16.4 (2019), pp. 1681–1693 (cit. on pp. 9–11, 13, 91).

[25] I. K. Kim et al. "CloudInsight: Utilizing a Council of Experts to Predict Future Cloud Application Workloads". In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. 2018, pp. 41–48. DOI: 10.1109/CLOUD.2018.00013 (cit. on pp. 9, 10, 13, 91).

[26] C. Liu et al. "An adaptive prediction approach based on workload pattern discrimination in the cloud". In: *Journal of Network and Computer Applications* 80 (2017), pp. 35–44 (cit. on pp. 9, 10, 13).

[27] R. A. Jacobs et al. "Adaptive mixtures of local experts". In: *Neural computation* 3.1 (1991), pp. 79–87 (cit. on p. 10).

[28] Y. H. Hu, S. Palreddy, and W. J. Tompkins. "A patient-adaptable ECG beat classifier using a mixture of experts approach". In: *IEEE transactions on biomedical engineering* 44.9 (1997), pp. 891–900 (cit. on p. 10).

[29] B. L. Dalmazo, J. P. Vilela, and M. Curado. "Online traffic prediction in the cloud: a dynamic window approach". In: *2014 International Conference on Future Internet of Things and Cloud*. IEEE. 2014, pp. 9–14 (cit. on pp. 11, 13).

[30] L. Chen et al. "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization". In: *Proceedings of the 2018 conference of the ACM special interest group on data communication*. 2018, pp. 191–205 (cit. on p. 12).

[31] S. Salman et al. "DeepConf: Automating data center network topologies management with machine learning". In: *Proceedings of the 2018 Workshop on Network Meets AI & ML*. 2018, pp. 8–14 (cit. on p. 12).

[32] B. Arzani et al. "Taking the blame game out of data centers operations with netpoirot". In: *Proceedings of the 2016 ACM SIGCOMM Conference*. 2016, pp. 440–453 (cit. on p. 12).

[33] A. Tongaonkar, R. Keralapura, and A. Nucci. "Santaclass: A self adaptive network traffic classification system". In: *2013 IFIP Networking Conference*. IEEE. 2013, pp. 1–9 (cit. on p. 12).

[34] A. Kansal et al. "Virtual machine power metering and provisioning". In: *Proceedings of the 1st ACM symposium on Cloud computing*. 2010, pp. 39–50 (cit. on p. 13).

[35] J. Herzen. *Time Series forecasting using past and future external data with darts*. 2022-12. URL: https://unit8.com/resources/time-series-forecasting-using-past-and-future-external-data-with-darts/ (cit. on p. 14).

[36] E. S. Gardner Jr. "Exponential smoothing: The state of the art". In: *Journal of forecasting* 4.1 (1985), pp. 1–28 (cit. on p. 15).

[37] J. Brownlee. *A gentle introduction to exponential smoothing for time series forecasting in Python*. 2020-04. URL: https://machinelearningmastery.com/exponential-smoothing-for-time-series-forecasting-in-python/ (cit. on p. 15).

[38] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018 (cit. on p. 15).

[39] S. J. Taylor and B. Letham. "Forecasting at scale". In: *The American Statistician* 72.1 (2018), pp. 37–45 (cit. on p. 15).

[40] M. Masum. *Identifying AR and Ma terms using ACF and PACF plots in time series forecasting*. 2022-06. URL: https://towardsdatascience.com/identifying-ar-and-ma-terms-using-acf-and-pacf-plots-in-time-series-forecasting-ccb9fd073db8 (cit. on p. 17).

[41] S. Kokilavani et al. "SARIMA modelling and forecasting of monthly rainfall patterns for Coimbatore, Tamil Nadu, India". In: *Current Journal of Applied Science and Technology* 39.8 (2020), pp. 69–76 (cit. on p. 17).

[42] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 18).

[43] C. Olah. *Understanding LSTM networks*. 2015-08. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (cit. on p. 18).

[44] A. Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017) (cit. on p. 18).

[45] A. Zeng et al. "Are Transformers Effective for Time Series Forecasting?" In: *arXiv preprint arXiv:2205.13504* (2022) (cit. on p. 19).

[46] B. Lim et al. "Temporal fusion transformers for interpretable multi-horizon time series forecasting". In: *International Journal of Forecasting* 37.4 (2021), pp. 1748–1764 (cit. on pp. 19, 20).

[47] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. "Fast and accurate deep network learning by exponential linear units (elus)". In: *arXiv preprint arXiv:1511.07289* (2015) (cit. on p. 20).

[48] Y. N. Dauphin et al. "Language modeling with gated convolutional networks". In: *International conference on machine learning*. PMLR. 2017, pp. 933–941 (cit. on p. 20).

[49] R. Guidotti et al. "A survey of methods for explaining black box models". In: *ACM computing surveys (CSUR)* 51.5 (2018), pp. 1–42 (cit. on p. 21).

[50] C. Molnar. *Interpretable machine learning*. Lulu. com, 2020 (cit. on pp. 21, 22).

[51] J. H. Friedman and B. E. Popescu. "Predictive learning via rule ensembles". In: *The annals of applied statistics* 2.3 (2008), pp. 916–954 (cit. on pp. 22, 23).

[52] H. Singh. *Understanding gradient boosting machines*. Last visited on December 2022. 2018-11. URL: https://towardsdatascience.com/understanding-gradient-boosting-machines-9be756fe76ab (cit. on p. 23).

[53] M. Z. A. Mayaki and M. Riveill. "Autoregressive based Drift Detection Method". In: *arXiv preprint arXiv:2203.04769* (2022) (cit. on p. 24).

[54] N. Jayawardena. *Data drift*. Last visited on December 2022. 2021-06. URL: https://towardsdatascience.com/data-drift-part-1-types-of-data-drift-16b3eb175006 (cit. on p. 24).

[55] C. Raab, M. Heusinger, and F.-M. Schleif. "Reactive soft prototype computing for concept drift streams". In: *Neurocomputing* 416 (2020), pp. 340–351 (cit. on p. 24).

[56] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for machine learning*. Cambridge University Press, 2020 (cit. on p. 24).

[57] *Reliance on social media in today's society*. Last visited on February 2022. 2019-10. URL: https://www.insegment.com/blog/reliance-on-social-media-in-todays-society/ (cit. on p. 35).

[58] A. Hutchinson. *Here's why Twitter is so important, to everyone*. Last visited on February 2022. 2016-03. URL: https://www.socialmediatoday.com/social-networks/heres-why-twitter-so-important-everyone (cit. on p. 35).

[59] L. Borges. *PS Contra Aumento de Remunerações E Prémios no Novo banco*. Last visited on February 2022. 2020-05. URL: https://www.publico.pt/2020/05/13/politica/noticia/ps-aumento-remuneracoes-premios-novo-banco-1916384 (cit. on p. 41).

[60] V. Kharde, P. Sonawane, et al. "Sentiment analysis of twitter data: a survey of techniques". In: *arXiv preprint arXiv:1601.06971* (2016) (cit. on pp. 46, 48).

[61] J. Hirschberg and C. D. Manning. "Advances in natural language processing". In: *Science* 349.6245 (2015), pp. 261–266 (cit. on p. 46).

[62] H. Bonthu. *Rule-based sentiment analysis in Python for Data Scientists*. 2021-06. URL: https://www.analyticsvidhya.com/blog/2021/06/rule-based-sentiment-analysis-in-python/ (cit. on p. 47).

[63] S. Chakravarthy. *Tokenization for natural language processing*. 2020-07. URL: https://towardsdatascience.com/tokenization-for-natural-language-processing-a179a891bad4 (cit. on p. 47).

[64] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009 (cit. on p. 47).

[65] A. Branco. *LX-Stopwords*. URL: https://hdl.handle.net/21.11129/0000-000B-D31A-A (cit. on p. 47).

[66] P. Carvalho and M. J. Silva. "SentiLex-PT: Principais caracteristicas e potencialidades". In: *Oslo Studies in Language* 7.1 (2015) (cit. on pp. 47, 48).

[67] P. Qi et al. "Stanza: A python natural language processing toolkit for many human languages". In: *arXiv preprint arXiv:2003.07082* (2020) (cit. on p. 47).

[68] C. D. Manning, P. Raghavan, and H. Schütze. "The term vocabulary and postings lists". In: *Introduction to information retrieval*. Cambridge University Press, 2018, pp. 19–45 (cit. on p. 47).

[69] V. Balakrishnan and E. Lloyd-Yemoh. "Stemming and lemmatization: a comparison of retrieval performances". In: (2014) (cit. on p. 47).

[70] A. Beri. *Stemming vs lemmatization*. 2021-01. URL: https://towardsdatascience.com/stemming-vs-lemmatization-2daddabcb221 (cit. on p. 47).

[71] D. A. Pereira. "A survey of sentiment analysis in the Portuguese language". In: *Artificial Intelligence Review* 54.2 (2021), pp. 1087–1115 (cit. on p. 48).

[72] S. F. O. Costa. "Adaptação e teste de uma base lexical de palavras emocionais para o português europeu:(EMOTAIX. PT)". In: (2012) (cit. on p. 50).

[73] A. Piolat and R. Bannour. "EMOTAIX: un scénario de Tropes pour l'identification automatisée du lexique émotionnel et affectif". In: *LAnnee psychologique* 109.4 (2009), pp. 655–698 (cit. on p. 50).

[74] D. A. Shamma, L. Kennedy, and E. F. Churchill. "Tweet the debates: understanding community annotation of uncollected sources". In: *Proceedings of the first SIGMM workshop on Social media*. 2009, pp. 3–10 (cit. on p. 51).

[75] H. Saif et al. "Evaluation datasets for Twitter sentiment analysis: a survey and a new dataset, the STS-Gold". In: (2013) (cit. on p. 51).

[76] S. Rosenthal, N. Farra, and P. Nakov. "SemEval-2017 task 4: Sentiment analysis in Twitter". In: *arXiv preprint arXiv:1912.00741* (2019) (cit. on p. 51).

[77] K. Kenyon-Dean et al. "Sentiment analysis: It's complicated!" In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 2018, pp. 1886–1895 (cit. on p. 51).

[78] S. Mohammad. "A practical guide to sentiment annotation: Challenges and solutions". In: *Proceedings of the 7th workshop on computational approaches to subjectivity, sentiment and social media analysis*. 2016, pp. 174–179 (cit. on p. 52).

[79] J. L. Fleiss. "Measuring nominal scale agreement among many raters." In: *Psychological bulletin* 76.5 (1971), p. 378 (cit. on p. 52).

[80] D. M. Blei, A. Y. Ng, and M. I. Jordan. "Latent dirichlet allocation". In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022 (cit. on p. 55).

[81] C. Li et al. "Enhancing topic modeling for short texts with auxiliary word embeddings". In: *ACM Transactions on Information Systems (TOIS)* 36.2 (2017), pp. 1–30 (cit. on p. 55).

[82] L. Hong and B. D. Davison. "Empirical study of topic modeling in twitter". In: *Proceedings of the first workshop on social media analytics*. 2010, pp. 80–88 (cit. on p. 55).

[83] X. Quan et al. "Short and sparse text topic modeling via self-aggregation". In: *Twenty-fourth international joint conference on artificial intelligence*. 2015 (cit. on p. 55).

[84] X. Yan et al. "A biterm topic model for short texts". In: *Proceedings of the 22nd international conference on World Wide Web*. 2013, pp. 1445–1456 (cit. on p. 55).

[85] K. Nigam et al. "Text classification from labeled and unlabeled documents using EM". In: *Machine learning* 39.2 (2000), pp. 103–134 (cit. on p. 55).

[86] J. Yin and J. Wang. "A dirichlet multinomial mixture model-based approach for short text clustering". In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2014, pp. 233–242 (cit. on p. 55).

[87] D. Mimno et al. "Optimizing semantic coherence in topic models". In: *Proceedings of the 2011 conference on empirical methods in natural language processing*. 2011, pp. 262–272 (cit. on p. 57).

[88] M. Christ et al. "Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package)". In: *Neurocomputing* 307 (2018), pp. 72–77 (cit. on p. 76).

[89] J. Wilkes. *More Google cluster data. Google research blog*. 2011 (cit. on p. 91).

[90] G. Campbell. *Does the stock market reflect the economy?* 2021-05. URL: https://www.economicsobservatory.com/does-the-stock-market-reflect-the-economy (cit. on p. 96).

[91] *GET /2/tweets/search/all | docs | twitter developer platform*. Last visited on February 2022. URL: https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-all (cit. on p. 105).

A

Datasets specification

Table A.1: Media accounts that were used to retrieve data. (Information present in the table was last consulted in September 2022)

Name	**Identifier**	**Number of tweets**	**Followers**
CM Jornal	@cmjornal	399.5K	474.8K
CNN Portugal	@cnnportugal	410.5K	413.7K
Diário de Notícias	@dntwit	328.5K	226.7K
Expresso	@expresso	514.6K	572K
Jornal de Negócios	@JNegocios	216.1K	194.7K
Jornal de Notícias	@JornalNoticias	279.9K	537.2K
Lusa Notícias	@Lusa_noticias	351K	277.2K
Observador	@observadorpt	350.2K	290.5K
Jornal Económico	@ojeconomico	181.8K	237.5K
Publico	@Publico	543.6K	887.6K
Sapo	@sapo	112.9K	286K
SIC Noticias	@SICNoticias	403.5K	959.4K
Sol	@SolOnline	225.1K	276.4K
TSF	@TSFRadio	281.7K	311K
RTP Notícias	@RTPNoticias	110.8K	443.5K
RTP 1	@RTP1	24.8K	232.7K

Table A.2: Features description of Twitter Dataset. Reasons descriptions for some features to have been discarded are presented in table A.4.

Feature	**Description** (Obtained from [91])	**Type**	**Removed**	**Reason**
conversation_id	The Tweet ID of the original Tweet of the conversation(which includes direct replies, replies of replies)	string	NO	-
text	The content of the Tweet.	string	NO	-
created_at	Creation time of the Tweet.	date (ISO 8601)	NO	-
lang	Language of the Tweet, if detected by Twitter. Returned as a BCP47 language tag.	string	NO	-
author_id	Unique identifier of this tweet user.	string	NO	-
attachments	Specifies the type of attachments (if any) present in this Tweet.	object	YES	1
source	The name of the app the user Tweeted from.	string	NO	-
reply_settings	Shows who can reply to this Tweet. Fields returned are *everyone*, *mentionedUsers*, and *following*.	string	YES	2
entities	Contains details about text that has a special meaning in a Tweet.	object	YES	3
context_annotations	Contains context annotations for the Tweet.	array	YES	1
public_metrics	Engagement metrics for the Tweet at the time of the request.	object	NO	-
possibly_sensitive	Indicates if this Tweet contains URLs marked as sensitive, for example content suitable for mature audiences.	boolean	YES	3
in_reply_to_user_id	If this Tweet is a Reply, indicates the user ID of the parent Tweet's author.	string	NO	-
referenced_tweets	A list of Tweets this Tweet refers to.	array	NO	-
withheld	Contains withholding details for withheld content.	object	YES	1
geo	Contains details about the location tagged by the user in this Tweet, if they specified one.	object	YES	3
id	The Tweet unique ID	string	NO	-

Table A.3: Description of metrics motorized in the novobanco data center. All the metrics are given in numerical values (with the respective scale) in a time series format. (*Reasons are presented in Table A.4)

Metric	Removed	Reason*	Description
Timestamp (Index)	NO	-	Time of occurrence of the record.
Check_MK			
Time usage by phase	YES	3	time usage of a monitoring tool that registers server metrics values.
Average	YES	3	
CPU utilization			
Per Core utilization	YES	3	CPU utilization measures how much work a CPU does to manage an operating system's tasks or process resources of a server.
Average	NO	-	
Disk IO summary			
Read wait time	NO	-	Monitors summary metrics related to active disk I/O time and the pace at which data is transferred from the hard drive to the RAM of a server.
Write wait time	NO	-	
Disk Read operations	YES	4	
Disk Write operations	NO	-	
Write Throughput	YES	4	
Read Throughput	YES	4	
File system			
Used filesystem space	YES	4	Monitors metrics related to the server file system. Each server utilizes a file system to manage available space in the server.
Filesystem Shrinking	NO	-	
Filesystem growth	YES	4	
Filesystem size	YES	2	
Free space	YES	4	
Interface			
Input Bandwidth	NO	-	Monitors different interfaces of each server as the bandwidth, packets, and errors that pass through the server network, either an input or output.
Output Bandwidth	NO	-	
Input non-unicast packets	NO	-	
Output non-unicast packets	YES	4	
Input unicast packets	YES	4	
Output unicast packets	YES	4	
Errors	YES	1	
Length of output queue	YES	2	
Memory and pagefile			
RAM used	NO	-	Monitors the memory used to maintain recent data. Specifically, it monitors the main memory of a computer (RAM), where all hardware, programs, and data that need to be quickly reached are maintained.
RAM installed	NO	-	
Errors	YES	2	
PING			
Round trip average	NO	-	Monitors how long it takes for packets to transit between a source and a target in a network.
Packet loss	YES	1	
Uptime	YES	4	Monitors the time a server runs without a restart or a shutdown, i.e., monitors the time a server is fully functional (the performance of the server).

Table A.4: Reasons for some features been discarded during data cleaning process

Reasons	**Description**
1	Discarded because of the high percentage of missing values
2	Discarded because all the values are the same in this feature.
3	Discarded because this feature has low importance for the problem that is being analyzed
4	Discarded because the feature is correlated with another one
5	Discarded because the values stagnate from a specific date and fail to provide relevant information

I

Questionnaire view

To collect the several manual annotations tweets, we need a questionnaire with the following characteristics:

- Good usability;
- Easy to share and access by all respondents;
- Present just 50 tweets to each respondent;
- Capable of presenting different tweets to different participants;
- Possibility to have hidden tests;
- Provide answers in a format easy to further analyze.

There are paid programs for the English language where we can set up all these rules, and the software finds respondents automatically and send the respective answers as APPEN[1]. Differently, there is free software such as Label Studio[2] that allows us to define everything manually and share the questionnaire with specific respondents of our choice. However, looking at our purpose, we thought that the interface provided by Label Studio was not very clear and could create some difficulties for respondents to use. Besides that, Label Studio does not allow to restrict the number of tweets for each respondent to analyze and cannot insert hidden tests.

Having this in mind, we decided to create a React app from scratch since we wanted simple software that followed the above characteristics. In this way, we could implement everything as we wanted and were necessary, providing a simple, easy, and specific app that fit our purpose. This app was published online via GitHub to be easily accessed by all participants (`https://gfmateus99.github.io/novobanco-survey/`), and the database for this app is 250 tweets selected randomly from the dataset with tweets related to *Novo Banco*. The internal structure of the developed application can be seen in I.1.

[1] `https://appen.com/`
[2] `https://labelstud.io/`

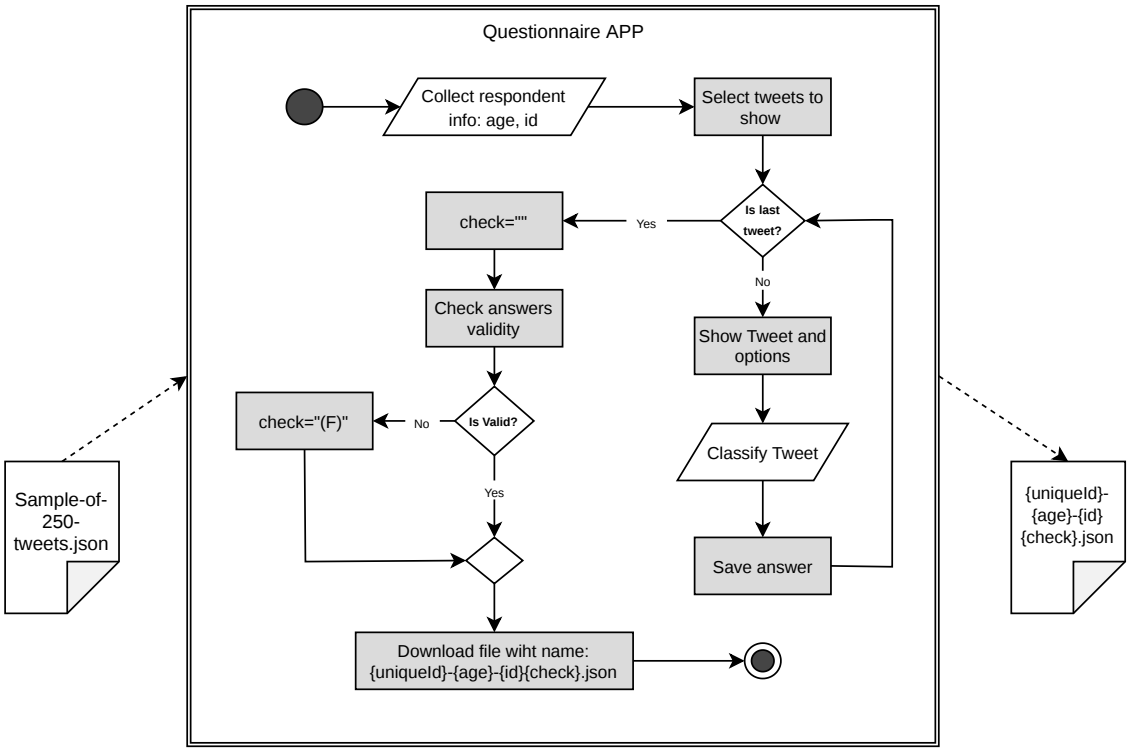


Figure I.1: Flow diagram of the internal structure of the developed application.

When the questionnaire is opened, the respondent has to insert some personal information, as shown in Figure I.2. The id camp is there to select the part of tweets that each respondent will classify (for example, id 1 will classify the first 50 tweets of the database, id 2 the tweets between positions 50 and 100 of the database, and so on). We previously assigned the id that each respondent should insert. After this, the user can start the questionnaire and will have 50 tweets to classify, as shown in I.3. When the respondent arrives at the end of the questionnaire, a file in `.json` format with the respondent's answers is automatically downloaded and in case of a download fail, the respondent will have a button available to repeat the download as Figure I.4 shows. The name of this file will be according to the information inserted by the respondent at the beginning. If the user inserted a *age="32"* and *id="4"* the file is downloaded with the name `"uniqueId-32-4.json"`. *uniqueId* is a random identifier to distinguish files of same *id*. Before the download, a hidden test is done to check if the respondent's answers were not random. For this test, we selected some tweets that the sentiment is obvious to compare with the respondent answers. If the answers did not match those tweets, it probably was a questionnaire answered with random answers, so the respective questionnaire needs to be discarded. To identify these cases, an additional camp with the `"(F)"` is added to the file's name. `"uniqueId-32-4.json"` would be a file that passed with success in our hidden tests, while `"uniqueId-32-4(F).json"` means that it did not pass in tests. To conclude the participation, respondents will have to send that file to a stated email address.

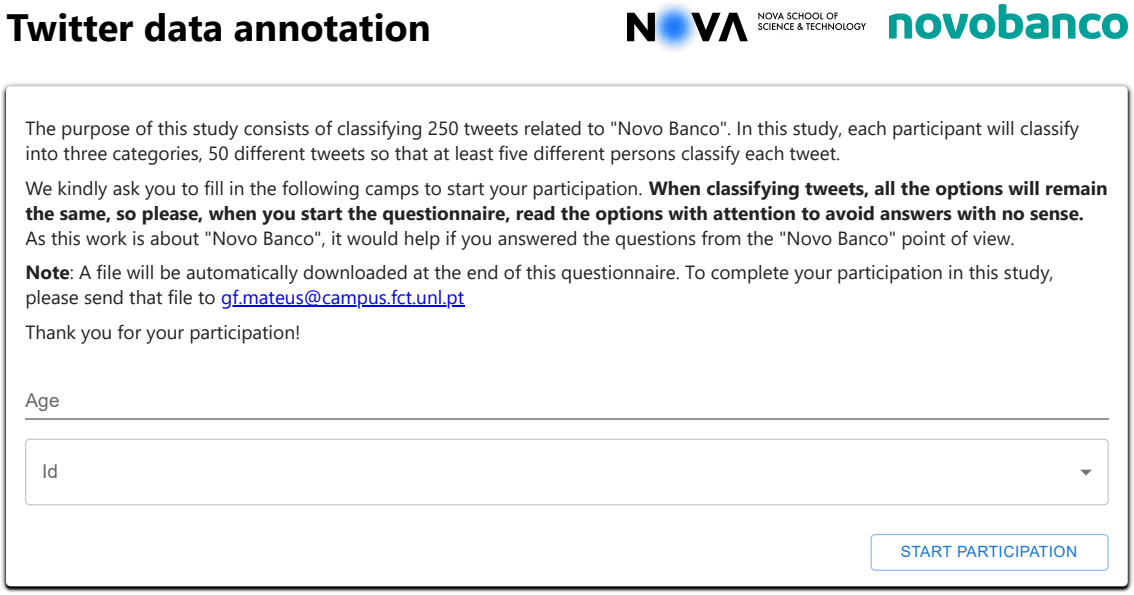


Figure I.2: First page of our questionnaire.

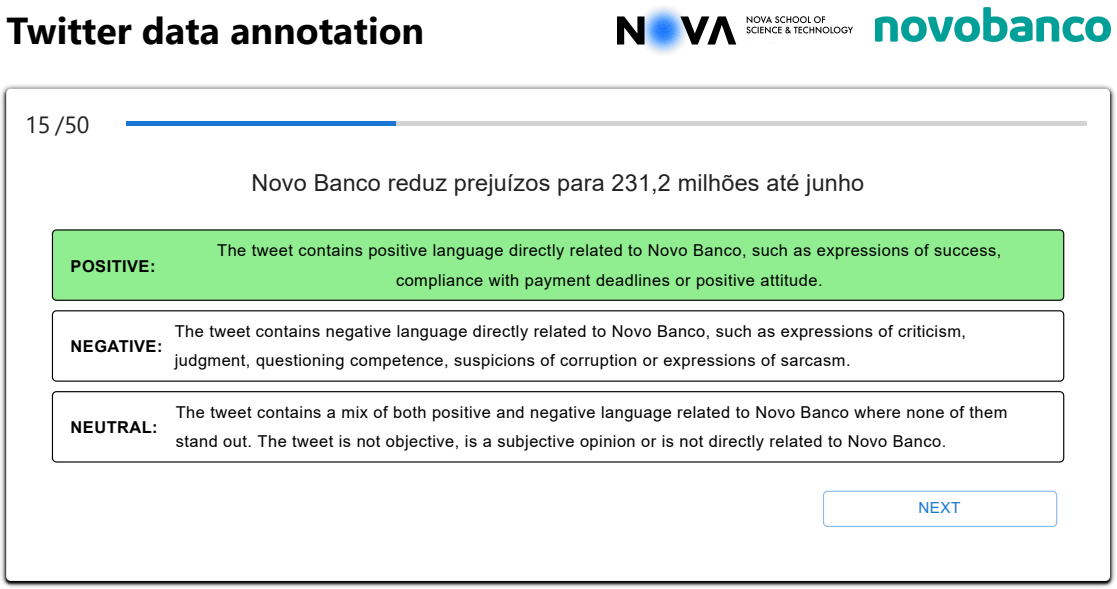


Figure I.3: Page with one question of our questionnaire.

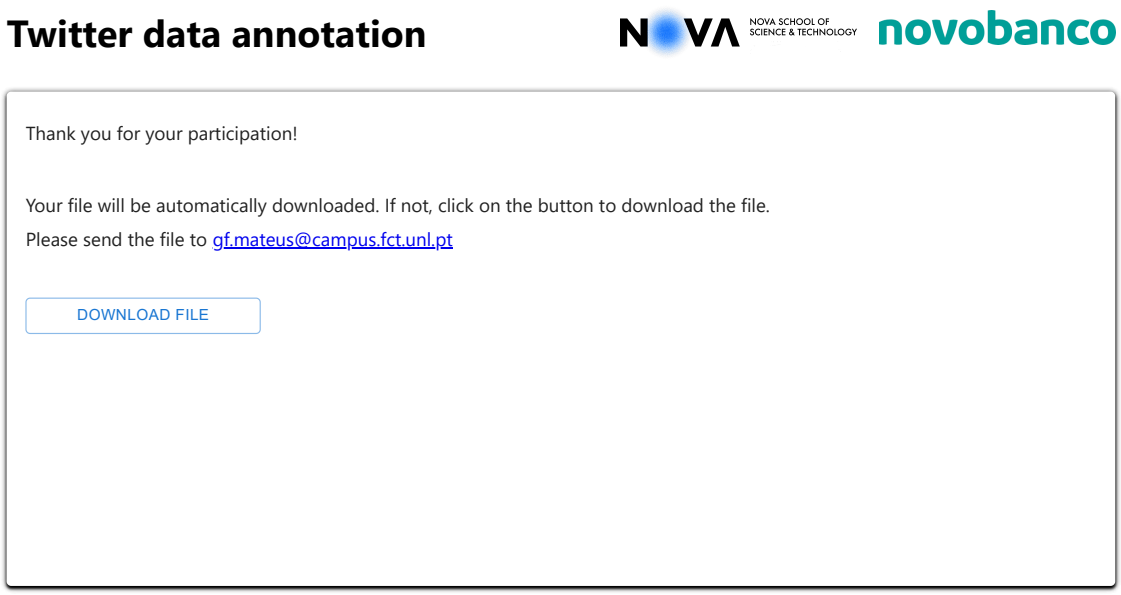
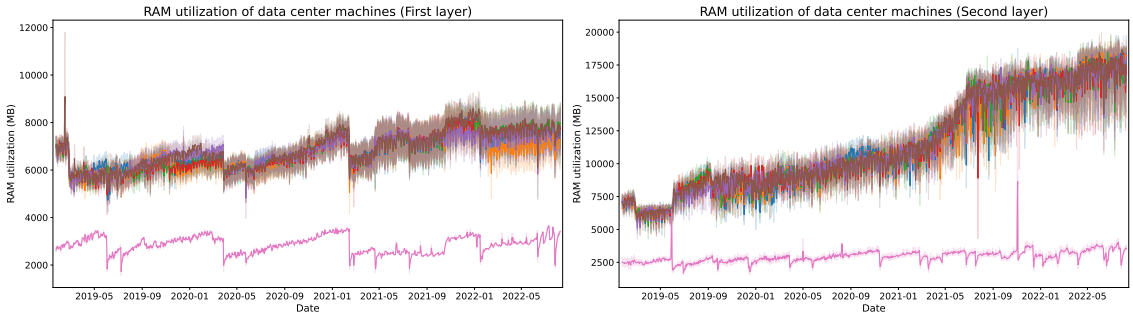


Figure I.4: Last page of our questionnaire.

II

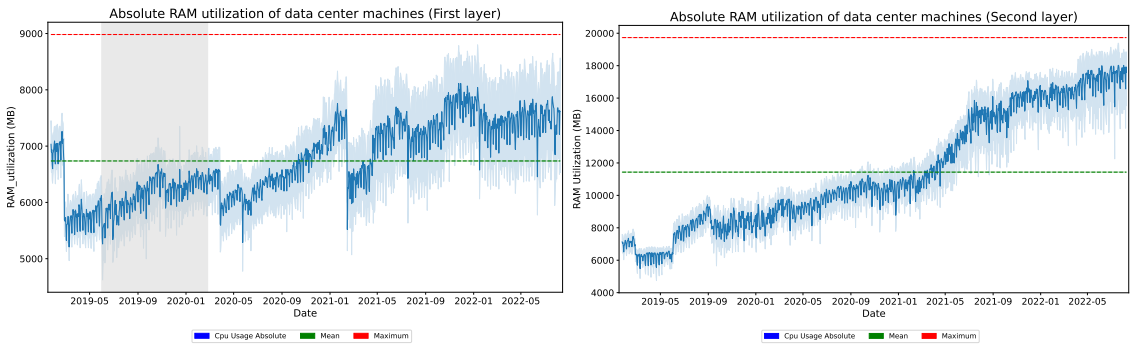
Other Metrics Plots

II.1 RAM Utilization



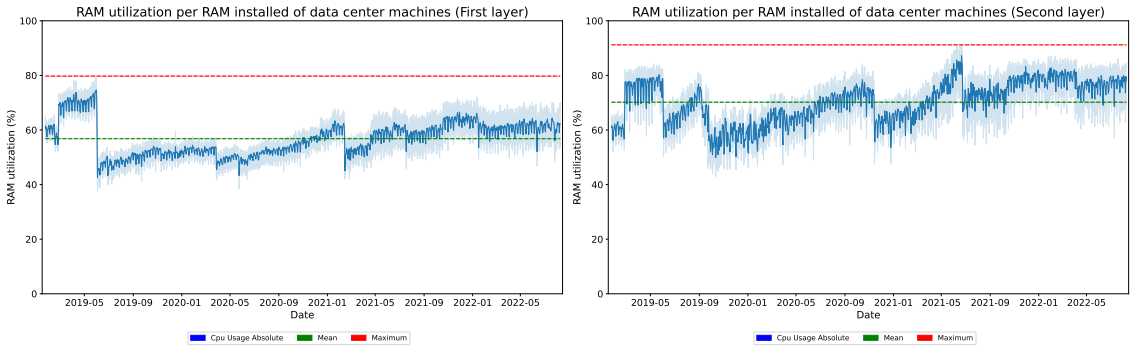
(a) RAM utilization of the servers of the first layer. (b) RAM utilization of the servers of the second layer.

Figure II.1: RAM usage layers



(a) Absolute RAM utilization of the first layer. (b) Absolute RAM utilization of the second layer.

Figure II.2: Absolute RAM utilization of servers (excluding the ones that serve as backup).

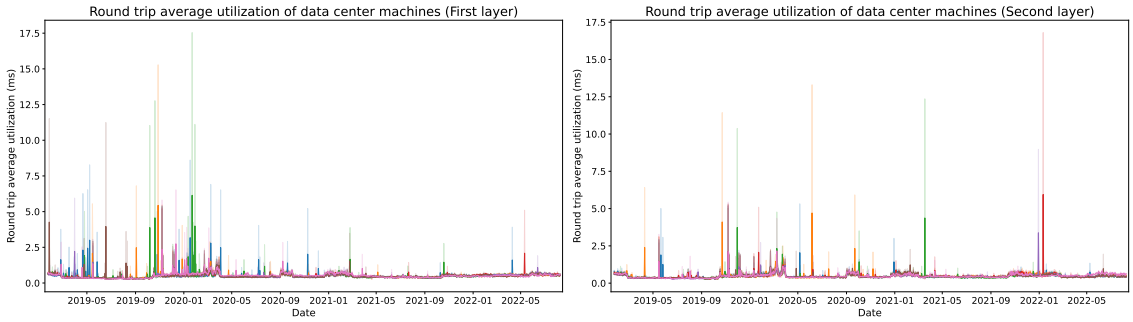


(a) Absolute RAM utilization of the first layer.

(b) Absolute RAM utilization of the second layer.

Figure II.3: Absolute RAM utilization of servers (excluding the ones that serve as backup).

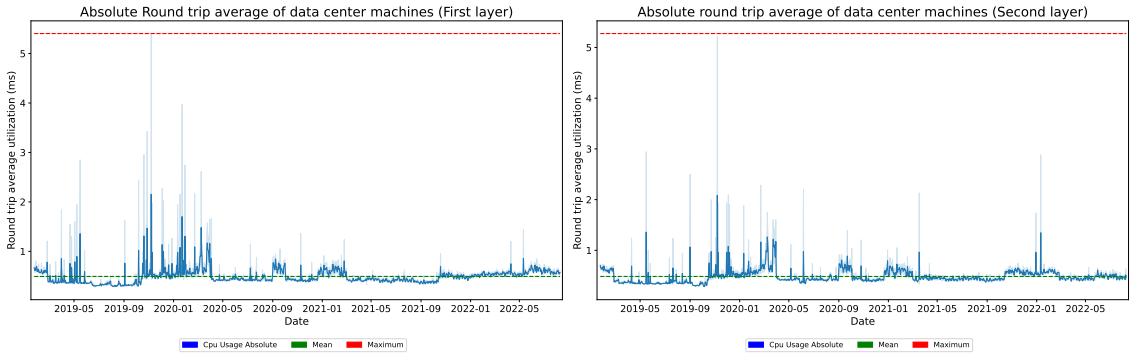
II.2 Round trip average utilization



(a) Round trip average utilization of the servers of the first layer.

(b) Round trip average utilization of the servers of the second layer.

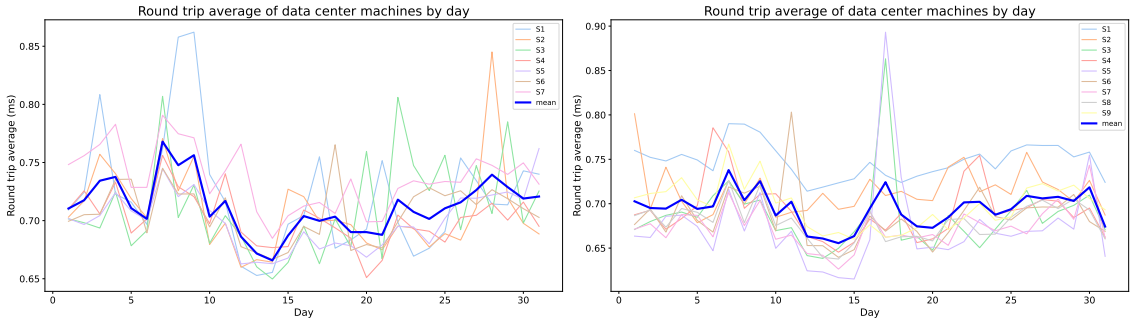
Figure II.4: Round trip average usage layers



(a) Absolute Round trip average utilization of the first layer.

(b) Absolute Round trip average utilization of the second layer.

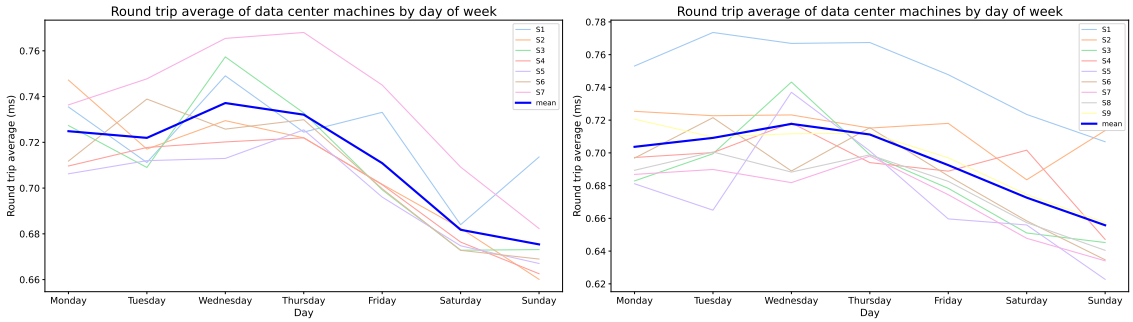
Figure II.5: Absolute Round trip average utilization of servers (excluding the ones that serve as backup).



(a) Round trip average of the first layer by day of the month.

(b) Round trip average of the second layer by day of the month.

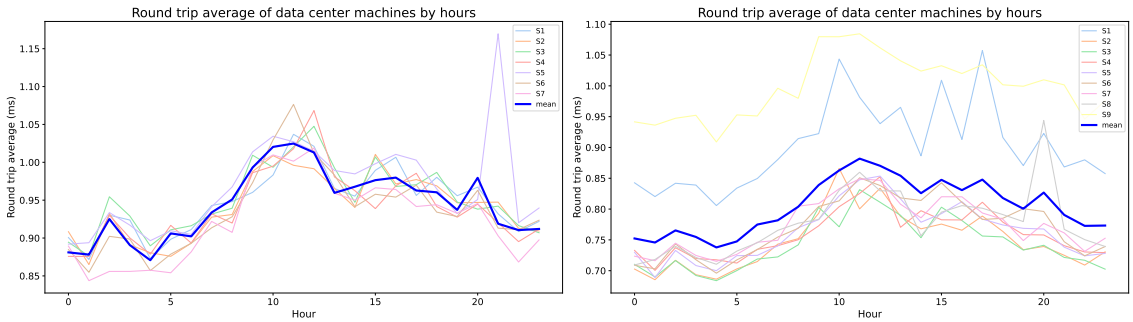
Figure II.6: Round trip average of data center machines by day of month



(a) Round trip average of the first layer by day of the week.

(b) Round trip average of the second layer by day of the week.

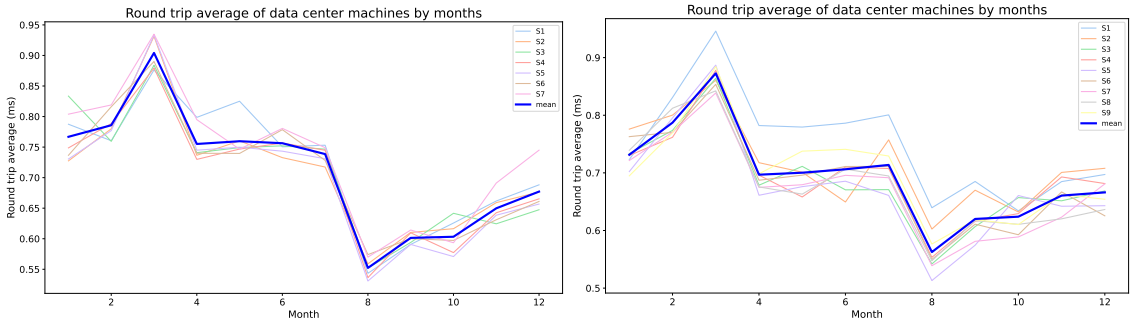
Figure II.7: Round trip average of data center machines by day of week



(a) Round trip average of the first layer by hours.

(b) Round trip average of the second layer by hours.

Figure II.8: Round trip average of data center machines by hours

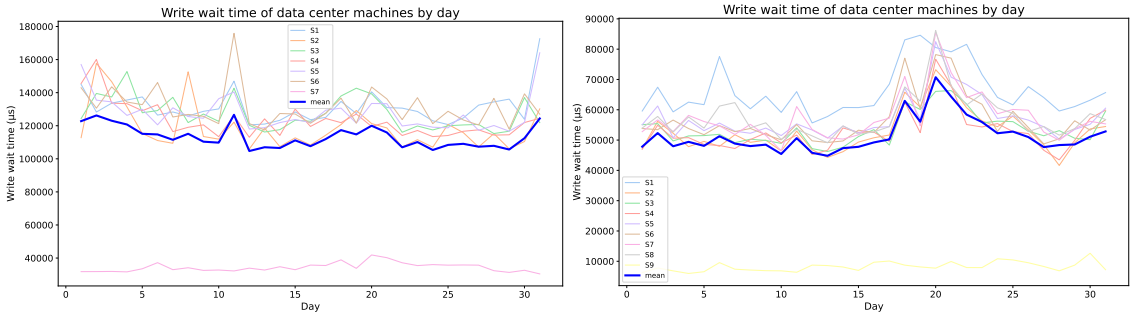


(a) Round trip average of first layer by months.

(b) Round trip average of second layer by months.

Figure II.9: Round trip average of data center machines by months

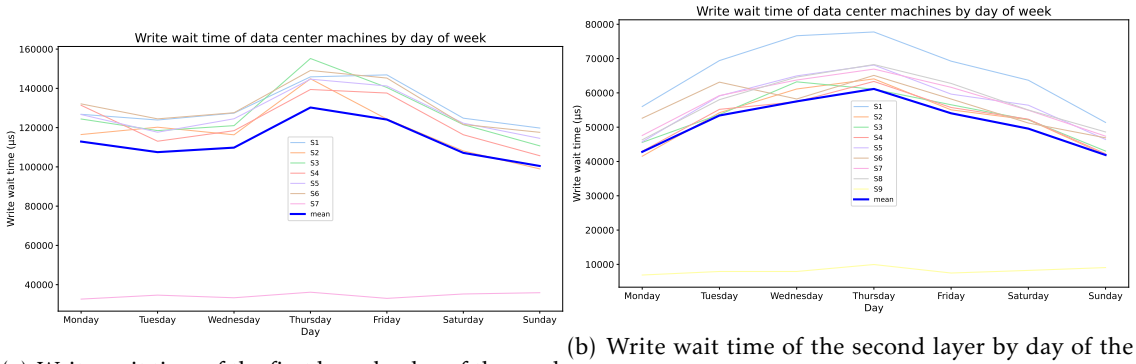
II.3 Write wait time



(a) Write wait time of the first layer by day of the month.

(b) Write wait time of the second layer by day of the month.

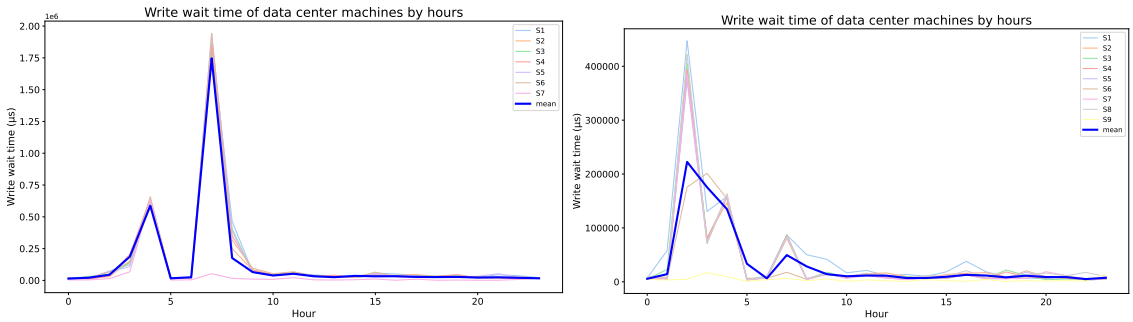
Figure II.10: Write wait time of data center machines by day of month



(a) Write wait time of the first layer by day of the week.

(b) Write wait time of the second layer by day of the week.

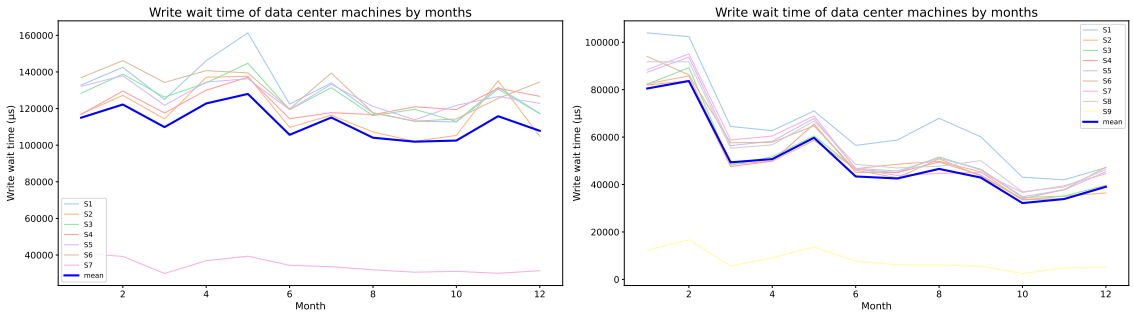
Figure II.11: Write wait time of data center machines by day of week



(a) Write wait time of the first layer by hours.

(b) Write wait time of the second layer by hours.

Figure II.12: Write wait time of data center machines by hours



(a) Write wait time of first layer by months.

(b) Write wait time of second layer by months.

Figure II.13: Write wait time of data center machines by months

II.4 Google Cluster Trace

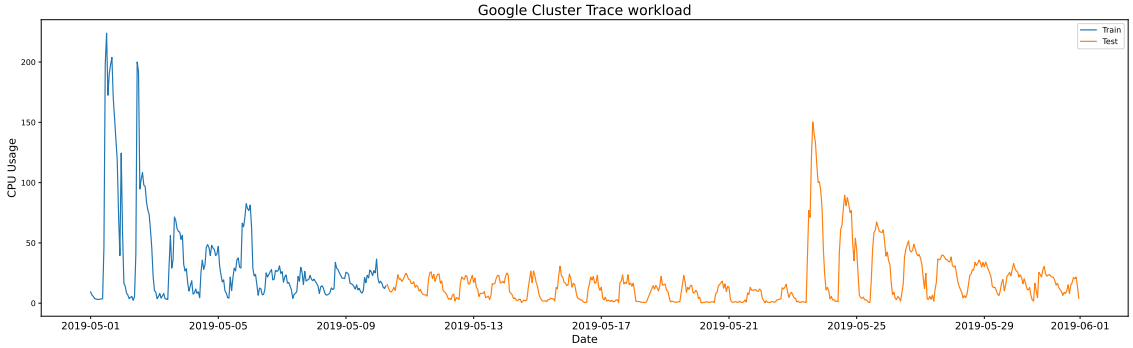


Figure II.14: Absolute CPU usage of Google Cluster Trace workload in hours.

NOVA SCHOOL OF SCIENCE & TECHNOLOGY