Universidade de Aveiro
2023

José Maria Marques Cação

Intelligent Assistant - Desenvolvimento de Algoritmos para Monitorização dos Consumos de Energia no Chão de Fábrica, Bosch

Intelligent Assistant - Development of Algorithms for Monitoring Energy Consumption on the Shop Floor, Bosch

Universidade de Aveiro
2023

José Maria Marques Cação

Intelligent Assistant - Desenvolvimento de Algoritmos para Monitorização dos Consumos de Energia no Chão de Fábrica, Bosch

Intelligent Assistant - Development of Algorithms for Monitoring Energy Consumption on the Shop Floor, Bosch

Trabalho de Projeto apresentado à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Mecânica, realizada sob orientação científica de José Paulo Oliveira Santos, Professor Auxiliar, do Departamento de Engenharia Mecânica da Universidade de Aveiro, e de Mário Luís Pinto Antunes, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

Este projeto teve o apoio dos projetos UIDB/00481/2020 e UIDP/00481/2020 - Fundação para a Ciência e a Tecnologia; e CENTRO-01-0145 FEDER-022083 - Programa Operacional Regional do Centro (Centro2020), através do Portugal 2020 e do Fundo Europeu de Desenvolvimento Regional.
Esta trabalho teve o apoio do projeto Augmented Humanity [POCI-01-0247-FEDER-046103 e LISBOA-01-0247-FEDER-046103], e contou com apoio laboratorial do Centro de Tecnologia Mecânica e Automação (TEMA), projetos UIDB/00481/2020 e UIDP/00481/2020.

O júri / The jury

Presidente / President	**Prof. Doutor André Figueiredo Quintã** Professor Auxiliar da Universidade de Aveiro
Vogais / Committee	**Prof. Doutor Pedro Alexandre de Sousa Gonçalves** Professor Adjunto da Universidade de Aveiro
	Prof. Doutor Mário Luís Pinto Antunes Professor Auxiliar da Universidade de Aveiro

Agradecimentos / Acknowledgements

Em primeiro lugar quero agradecer à minha equipa de orientação. Ao professor José Paulo por nunca me ter faltado e deixado de mãos vazias. Ao professor Mário Antunes pela disponibilidade, paciência e grande ajuda que me deu. Ao Gonçalo, que após longas horas de trabalho e frustração me continuou sempre a motivar para continuar. E ao Joaquim Ribeiro e Pedro Rendeiro, pela sua compreensão e ajuda.

Quero agradecer ao Lima, Pombeiro e Lopes, 3 dos melhores amigos que alguma vez poderia ter tido. Memórias ficarão das horas passadas a trabalhar para os projetos, das longas chamadas que duravam tardes e noites inteiras, mas especialmente das noitadas e jantaradas. Também ao Ribeiro e Sousa, os primeiros grandes amigos que pude ter a sorte de conhecer na universidade, dois grandes futuros engeheiros.

Agradeço muito também à Sofia, à Tatiana e ao André, outras 3 grandes pessoas que levarei para a vida. A toda a ajuda que me deram nestes últimos meses, numa última etapa tão dura e fatigante. Fizeram-me sentir acolhido, levaram-me a explorar novos sítios, fizeram-me feliz como nunca.

Há ainda muitas outras pessoas que nos últimos 5 anos me ajudaram e deram força. Apesar de não mencionadas, fica aqui o meu mais profundo agradecimento. Contudo, falta ainda referir as três pessoas mais importantes, que se destacam por tudo o que fizeram por mim ao longo não só destes 5 anos, mas nos últimos 23. Ao meu pai, o maior exemplo que alguma vez poderia ter tido, pela sua humildade, simpatia, alegria e espírito de sacrifício. À minha avó Estrela, que trago e trarei sempre no meu coração, e que sei o quão ela gostaria de aqui estar neste momento para me poder ver e abraçar. E principalmente à minha mãe, a mais forte mulher que conheço e alguma vez conhecerei. Por todos os sacrifícios, as viagens, o tempo que perdeu por mim. Sem ela não estaria aqui, não teria conseguido nem metade do que fiz... Nada do que eu fizer poderá alguma vez compensar o que ela (e eles) fizeram por mim. Por isso MUITO OBRIGADO.

Keywords Intelligent assistant, Industry 4.0, Machine Learning, Anomaly detection, *Autoencoders*

Abstract

In recent years, we have witnessed an enormous technological evolution with significant impacts on the industry. The rapid digitalization of processes has led to the emergence of a fourth industrial revolution, known as Industry 4.0. The increasing demands for efficiency and quality of processes have prompted industries to seek new and improved ways of acquiring and processing information from the factory floor.

In line with this trend, Bosch Termotecnologia Aveiro has initiated the Augmanity Project, which aims to adapt and digitize industrial processes in light of Industry 4.0 principles while considering the underlying environmental impacts. This work is precisely integrated into this project.

Conducted at the University of Aveiro and implemented at the facilities of Bosch Termotecnologia Aveiro, this work states the development of an intelligent assistant that is part of a data collection and monitoring architecture for the factory floor. The proposed architecture is cost-effective, incorporating a set of open-source software tools such as Cloud2Edge for data aggregation, InfluxDB for data storage, and Grafana for visualization. The intelligent assistent is equipped with Machine Learning algorithms, specifically autoencoders, whose building process is complemented by a recently developed and emerging tool called AutoML. AutoML enables a more automated model building process, as it promotes optimization and adaptation of the models to the variables to be predicted. The constructed models are then capable of making predictions of future energy consumption, which are subsequently compared with the actual values to detect anomalies. Finally, the developed assistant has various forms of interaction with the user, ranging from a graphical interface for data visualization to automated notifications based on processing results.

The results of the practical implementation of the architecture, which includes the assistant, demonstrate an excellent predictive capability of the constructed models. The implementation of AutoML as an optimization tool ensures significant improvements to the models, with performance increases of around 90% compared to base regression models. The models also show excellent results in anomaly detection, being able to adequately identify different types of anomalies. Moreover, the architecture, in addition to being validated in a simulated environment, has also been successfully implemented on the factory floor, enabling real-time processing and visualization of information within an industrial setting.

Palavras-chave Agente inteligente, Indústria 4.0, *Machine Learning*, Deteção de anomalias, *Autoencoders*

Resumo

Nos últimos anos temos assistido a uma enorme evolução tecnológica, com impactos significativos ao nível da indústria. A rápida digitalização dos processos que se tem verificado leva a que se esteja a assistir a uma quarta revolução a nível industrial, com este período a ser reconhecido como o da Indústria 4.0. As necessidades cada vez mais elevadas de eficiência e qualidade têm levado a que as indústrias procurem novas e melhores formas de adquirir e processar toda a informação proveniente do chão de fábrica.
Com isto em mente, a Bosch Termotecnologia Aveiro criou o Projeto *Augmanity*, o qual tem como principal objetivo a adequação e digitalização dos processos industriais à luz das noções da Indústria 4.0, tendo sempre em conta os impactos ambientais subjacentes. Este trabalho integrou-se exatamente neste projeto.
Desenvolvido na Universidade de Aveiro e com implementação prática nas instalações da Bosch Termotecnologia Aveiro, este trabalho expõe o desenvolvimento de um agente inteligente, o qual se integra numa arquitetura de recolha e monitorização de dados do chão de fábrica. A arquitetura proposta é de baixo custo, incluindo um conjunto de *softwares* gratuitos tais como o Cloud2Edge para agregação dos dados, o InfluxDB para o seu armazenamento e o Grafana para a visualização. Já o agente inteligente é dotado de algoritmos de *Machine Learning*, mais propriamente *autoencoders*, cujo processo de construção é complementado com uma ferramenta recentemente desenvolvida e emergente, o AutoML, que permite um processo de construção de modelos muito mais automatizado e promove a otimização e adequação dos modelos às variáveis a prever. Os modelos construidos são depois capazes de realizar previsões de consumos energéticos futuros, as quais são posteriormente comparadas com os valores reais, a fim de se detetarem anomalias. Por fim, o agente inteligente desenvolvido possui diversas formas de interação com o utilizador, que vão desde uma interface gráfica para visualização dos dados até ao envio automatizado de notificações com base nos resultados do processamento.
Os resultados da implementação prática da arquitetura, onde se enquadra o agente, revelam uma excelente capacidade de previsão dos modelos construidos. A implementação do AutoML como ferramenta de otimização garante melhorias signficativas aos modelos, com aumentos de desempenho na ordem dos 90% face a modelos regressivos base. Também na tarefa de deteção de anomalias os modelos revelam excelentes resultados, sendo capazes de identificar adequadamente diferentes tipos de anomalias. Por fim, a arquitetura, para além de validada em ambiente simulado, foi também implementada no chão de fábrica com sucesso, permitindo o processamento e visualização da informação em tempo real dentro já de um ambiente industrial.

Índice

Intentionally blank page.

Lista de Tabelas

Intentionally blank page.

Lista de Figuras

Lista de Acrónimos

AE *Autoencoder*.

AMQP *Advanced Message Queuing Protocol*.

ANN *Artificial Neural Networks*.

API *Application Programming Interface*.

ARIMA *Autoregressive Integrated Moving Average*.

AutoML *Automated Machine Learning*.

C2E Cloud2Edge.

CoaP *Constrained Application Protocol*.

CSV *Comma-Separated Values*.

DT *Digital Twin*.

EMA *Exponential Moving Average*.

ERP *Enterprise Resource Planning*.

GRU *Graded Recurrent Units*.

HTTP *Hypertext Transfer Protocol*.

IIoT *Industrial Internet of Things*.

IoT *Internet of Things*.

JSON *JavaScript Object Notation*.

KNN *K-Nearest Neighbours*.

LOF *Local Outlier Factor*.

LSTM *Long-Short Term Memory*.

MAE *Mean Absolute Error*.

MCC *Matthew's Correlation Coefficient*.

MES *Manufacturing Execution System*.

ML *Machine Learning*.

MQTT *Message Queuing Telemetry Transport*.

MSE *Mean Squared Error*.

NoSQL *Not only SQL*.

NPN *Non Public Network*.

PLC *Programmable Logic Controller*.

RF *Random Forest*.

RMSE *Root Mean Squared Error*.

RNN *Recurrent Neural Networks*.

SQL *Structured Query Language*.

SSE *Server Sent Events*.

SVM *Support Vector Machine*.

TCP/IP *Transfer Control Protocol / Internet Protocol*.

URL *Uniform Resource Locator*.

XML *Extensible Markup Language*.

Capítulo 1

Introdução

Este capítulo inicial consiste numa breve introdução ao projeto desenvolvido ao longo do semestre. É composto por um enquadramento inicial, onde se expõe a necessidade da realização do projeto, seguida pela motivação, parte onde é apresentado um conjunto de ferramentas que permitiram o seu desenvolvimento. Segue-se depois a apresentação do objetivo principal do projeto, exposição dos vários contributos que resultaram do seu desenvolvimento, e por fim uma breve explicação da restante estrutura do documento.

1.1 Enquadramento

Nos últimos séculos, as constantes evoluções que têm ocorrido a nível tecnológico levaram a um enorme crescimento e evolução da indústria. Há cada vez mais empresas a competir no mesmo espaço, com maiores instalações, capacidades de produção e maior quantidade de dados em circulação. Atualmente vivemos a quarta revolução a nível industrial, denominada por Indústria 4.0. Esta tem promovido a digitalização de todos os processos das empresas, tentando transportá-las para uma dimensão digital através de novas tecnologias como a *Internet of Things* (IoT), serviços em *cloud* ou técnicas de *Machine Learning* (ML).

Neste contexto surgiu o *Industrial Internet of Things* (IIoT), que corresponde à integração e ao uso da IoT em aplicações e ambientes industriais. Com o IIoT, é possível explorar as potencialidades dos dispositivos IoT ao nível da sua capacidade de recolha e análise em tempo real da informação, com respetiva implementação num contexto industrial. É possível aumentar a autonomia dos processos, com menor necessidade de ação humana, e assim diminuir os riscos de erros humanos. A constante recolha e análise de dados, aliada a técnicas de ML, pode permitir a identificação de ineficiências nos processos ou certas tendências nos dados [1]. É ainda possível melhorar a questão do controlo de qualidade e prever automaticamente necessidades de manutenção, ou fazer previsões relativas a falhas que possam surgir. Conjugando isto tudo é possível reduzir custos e tempos associados aos processos industriais [2].

Contudo, apesar desta evolução trazer grandes vantagens ao nível da eficiência dos processos, capacidade de processamento e produção das empresas, também aumenta a sua complexidade. Assim, é essencial garantir a integração da grande infraestrutura física (sensores, atuadores, máquinas, computadores...) com os sistemas digitais da forma o mais eficiente possível. Daqui surge o Projeto *Augmanity - Augmented Humanity* [3] onde

se integra o projeto desenvolvido. Este é coordenado pela Bosch Termotecnologia Aveiro com o propósito de conjugar a nível industrial a digitalização e adequar os processos às necessidades da Indústria 4.0 e do IIoT.

À luz do Projeto *Augmanity* já vários autores desenvolveram projetos de diferentes naturezas [4]–[7]. Rendeiro [4], Camarneiro [5] e Matos [6] desenvolveram arquiteturas de recolha e monitorização de um conjunto de dados provenientes do chão de fábrica, contribuindo para a digitalização dos processos. Já Oliveira [7] desenvolveu um trabalho de índole mais teórica, onde analisou e comparou um conjunto de técnicas de ML para tarefas de previsão e deteção de anomalias. Com estas duas dimensões distintas já abordadas, mas nunca unidas, o projeto desenvolvido propôs-se a integrar a capacidade de processamento conseguida através de técnicas de ML numa arquitetura de recolha de dados industrial. Desta junção resultou um agente inteligente, o qual a partir dos dados recolhidos do chão de fábrica é capaz de aplicar algoritmos preditivos a variáveis relacionadas com consumos energéticos. A partir das previsões promove também a deteção de anomalias, numa tentativa de identificar oportunidades de potencial melhoria dos consumos dos equipamentos. Adicionalmente, possui um conjunto variado de possibilidades de interação com o utilizador, desde interfaces gráficas de visualização dos dados até ao envio automatizado de notificações.

1.2 Motivação

Nas últimas décadas, tem havido uma crescente preocupação a nível ambiental e energético. O grupo Bosch tem sido incansável na tentativa de reduzir a pegada carbónica e as suas emissões. Desde 2020, cerca de 400 localizações do grupo atingiram a neutralidade carbónica. Adicionalmente, foram já estabelecidas metas para reduzir até 2030 mais 15% das emissões de CO_2 [8].

A evolução progressiva das indústrias para um contexto IIoT, apesar das enormes vantagens ao nível da eficiência e qualidade dos processos, acaba por implicar uma maior exigência ao nível dos equipamentos. Havendo mais equipamentos a produzir mais para suportar estas exigências, têm-se inadvertidamente maiores consumos energéticos ao nível do chão de fábrica, que se refletem num ambiente produtivo com maiores impactos ambientais. Para contrariar esta tendência surge a necessidade de encontrar soluções, dentro do contexto da Indústria 4.0, que promovam um aumento da eficiência energética dos processos, de modo a reduzir os consumos e emissões associadas.

Neste projeto propõe-se a implementação de uma plataforma, num contexto IIoT, capaz de promover uma capacidade de monitorização e processamento dos dados de consumos energéticos de equipamentos do chão de fábrica. Esse processamento implica a previsão e posterior deteção de anomalias sobre esses consumos, tentando fornecer oportunidades para eventuais reduções dos consumos dos equipamentos. Por um lado, a partir de tecnologias já existentes ao nível da Indústria 4.0, tais como *Big Data* e *Cloud Computing*, é possível desenvolver uma plataforma capaz de recolher continuamente uma enorme quantidade de dados, assim como armazená-los em bases de dados na *cloud* facilmente acessíveis via *internet*. Posteriormente, pode-se construir uma interface de visualização dos dados, para permitir uma fácil monitorização do que acontece em chão de fábrica. Paralelamente, ocorre o processamento dos dados recolhidos. Para isso, pode-se recorrer à enorme panóplia de algoritmos de ML e a técnicas inovadoras como o *Automated Ma-*

chine Learning (AutoML) para promover o desenvolvimento e implementação de modelos para deteção de consumos energéticos anómalos. Estes podem ser treinados com dados históricos recolhidos na plataforma de modo a modelar os padrões de consumo típicos dos equipamentos. Depois, realizando previsões de consumos futuros e comparando-as com os valores reais, pode-se proceder à deteção de anomalias sobre os mesmos. Por fim, a possibilidade de geração automática de notificações e avisos torna estes sistemas altamente capazes e adaptados às exigências subjacentes à Indústria 4.0.

Assim, com uma base de trabalho já de alguma forma validada por outros autores ao nível da recolha e monitorização dos dados [4]–[6] e ao nível de certos algoritmos e técnicas de ML [7], implementou-se a capacidade de processamento dos dados de modo a conseguir analisar consumos energéticos ao nível de chão de fábrica. O trabalho foi portanto dividido em dois, onde numa primeira fase houve a construção e implementação de uma arquitetura, com base nas já desenvolvidas, para promover a recolha dos dados. A segunda fase focou-se no processamento dos dados, com vista à construção do sistema inteligente e autónomo de deteção das anomalias, geração automatizada de avisos e notificações para o utilizador.

1.3 Objetivo

Como já exposto nas secções anteriores, já houve um conjunto de trabalhos realizados em colaboração com o Projeto *Augmanity* para a promoção da digitalização dos processos ao nível do chão de fábrica da Bosch Termotecnologia Aveiro. Há já propostas de arquiteturas digitais de recolha e monitorização de dados dos equipamentos, e também estudos pormenorizados relativamente às potencialidades do seu processamento. Adicionalmente, há também um conjunto de casos de estudo atualmente em vigor relacionados exatamente com estas temáticas.

Posto isto, o trabalho desenvolvido tentou integrar estas duas dimensões, a recolha de dados e o seu processamento, num único agente. O seu objetivo foi então o desenvolvimento de um agente inteligente aplicado a tarefas de previsão e deteção de anomalias ao nível dos consumos energéticos e outras variáveis relacionadas em equipamentos no chão de fábrica. Este agente integrou-se numa arquitetura de recolha e monitorização de dados, construída e implementada quase de raiz, mas já tendo como bases as previamente desenvolvidas. Por fim, a arquitetura foi implementada a um dos casos de estudo da Bosch, um compressor industrial conectado a um analisador de energia.

Para proceder ao desenvolvimento da solução proposta, e atingir o objetivo mencionado do projeto, houve ainda uma série de requisitos a cumprir:

- Sistema deve recorrer à infraestrutura 5G e à plataforma IoT atualmente existente nas instalações da empresa;
- Para obter os dados do chão de fábrica, devem ser usadas tecnologias com recurso a *Digital Twins*;
- A plataforma de apresentação e visualização dos dados deve conseguir criar gráficos interativos e de fácil leitura em tempo real;
- Sistema inteligente deve ser capaz de enviar notificações automaticamente a operadores, via email ou mensagem, sobre anomalias identificadas nos consumos energéticos dos equipamentos.

1.4 Contributos

Do trabalho desenvolvido durante o semestre, são de destacar os seguintes contributos:

- Colaboração com o Projeto *Augmanity* [POCI-01-0247-FEDER-046103 e LISBOA-01-0247-FEDER-04610] - todo o trabalho exposto neste documento foi integrado no Projeto *Augmanity*, coordenado pela Bosch Termotecnologia Aveiro. O projeto está organizado em 6 PPSs (*Product, Process, Service*), com o agente inteligente a integrar-se no PPS3 - *Industrial Internet of Things and Connectivity*;
- Repositório Github - todo o código principal desenvolvido durante este projeto encontra-se disponível publicamente através do repositório "*Intelligent Assistant*" (`https://github.com/zemaria2000/IntelligentAssistant`). Neste consta código relativo às tarefas de previsão e deteção de anomalias, à construção dos modelos de ML, aos serviços de recolha e filtragem de dados do chão de fábrica e ainda alguns ficheiros onde foram realizados testes aos modelos construidos;
- Artigo "*Industrial Internet of Things over 5G: a Practical Implementation*" [9] (submetido e publicado) - o artigo, elaborado em conjunto com a Bosch Termotecnologia Aveiro e a Fraunhofer Portugal, reflete o trabalho desenvolvido em torno da implementação de uma arquitetura IIoT que recorre à rede 5G implementada na Bosch Termotecnologia Aveiro. O agente inteligente desenvolvido neste trabalho enquadra-se na implementação da arquitetura na Bosch;
- Artigo "*Intelligent Assistant for Smart Factory Power Management*" (submetido) - este artigo retrata o trabalho exposto neste documento, que culminou na implementação de um agente inteligente, apto a realizar tarefas de previsão e deteção de anomalias, num ambiente industrial adaptado às necessidades da Indústria 4.0.

1.5 Estrutura do Documento

Após uma breve introdução ao projeto, onde foi efetuado um breve enquadramento, exposto o seu objetivo principal e ainda apresentados os principais contributos da sua realização, de seguida é descrita a restante divisão do documento:

- **Capítulo 2: Estado de Arte** - exposição inicial de um conjunto de temas, ferramentas e trabalhos realizados por outros autores que tiveram influência no trabalho desenvolvido, de modo a facilitar a compreensão dos capítulos subsequentes;
- **Capítulo 3: Solução Concetual Proposta** - apresentação da solução concetual proposta para cumprir com os objetivos do projeto;
- **Capítulo 4: Implementação da Solução** - capítulo onde são expostos os vários passos tomados até se conseguir passar da solução teórica proposta no Capítulo 3 para uma implementação prática da mesma;
- **Capítulo 5: Análise dos Resultados** - reflete os resultados da implementação prática da arquitetura. Contempla uma análise de desempenho completa dos modelos implementados na previsão e deteção de anomalias, assim como uma análise à implementação prática do caso de estudo Bosch;
- **Capítulo 6: Conclusão** - capítulo final, com a reflexão global do trabalho realizado e algumas propostas de trabalhos futuros baseados no trabalho desenvolvido.

Capítulo 2

Estado de Arte

Neste capítulo é feita uma revisão aprofundada aos principais conceitos relevantes para o desenvolvimento do projeto em causa. Este está dividido em três partes fundamentais: a primeira expõe os conceitos mais teóricos, desde pormenores relativos ao armazenamento de dados até à exposição de métodos e algoritmos para previsão e deteção de anomalias; a segunda secção introduz um conjunto de tecnologias e ferramentas existentes que tiveram impacto na implementação prática da solução; e por fim na última parte são analisados dois trabalhos anteriores realizados no âmbito do Projeto *Augmanity*, os quais constituíram uma base para o trabalho desenvolvido.

2.1 Conceitos Teóricos

Como referido acima, esta primeira parte serve para expor alguns conceitos mais teóricos relacionados com o trabalho desenvolvido. É feito um resumo global de qual a organização mais apropriada à construção de uma arquitetura completa de recolha, monitorização e processamento de dados. De seguida analisados um conjunto de importantes conceitos ao nível teórico, nomeadamente relativos a bases de dados, implementação de *digital twins* e a métodos de deteção de anomalias. Tudo isto aliado a uma análise bibliográfica completa para suportar cada uma das temáticas.

2.1.1 Arquiteturas de Recolha e Monitorização de Dados

De modo a construir uma arquitetura eficiente e capaz de transmitir grandes quantidades de dados em tempo real, a sua organização segundo blocos e camadas é essencial [10]–[13]. Há três grandes camadas que se podem distinguir numa arquitetura deste tipo:

- **Camada física** - esta camada combina todo o conjunto de equipamentos físicos ao nível do chão de fábrica e quaisquer outros dispositivos intermédios de transmissão da informação (sensores, microcontroladores, analisadores de energia, etc.). Todos estes equipamentos enviam constantemente novas informações via diferentes protocolos de comunicação. Por este motivo nesta camada física há também a necessidade de implementar dispositivos capazes de receber essa informação e agrupá-la. A esses equipamentos atribui-se o nome de *gateways*. Estes conseguem comunicar com o chão de fábrica sob a forma de vários protocolos de comunicação,

e permitem um fluxo de dados uniforme e facilitado para as restantes camadas da arquitetura;

- **Camada digital** - camada intermédia na qual é possível a identificação de um conjunto de importantes blocos com funções distintas. Começa-se por um bloco de agregação de dados, no qual a informação que chega via *gateways* é recebida e organizada. Tem-se depois um bloco de armazenamento, no qual uma ou várias bases de dados recebem a informação e guardam registos históricos. Essa informação pode depois seguir para processamento, outro bloco, onde podem ser aplicados, por exemplo, algoritmos de ML, e de onde resultam também *outputs*. Por fim tem-se o bloco de visualização, comum à camada seguinte, onde podem ser visualizados os *outputs* do processamento;
- **Camada do utilizador** - esta é a camada final onde o utilizador pode visualizar os resultados do processamento e fluxo de dados resultantes da camada intermédia. Aqui há a possibilidade do utilizador visualizar os dados segundo interfaces gráficas, receber alertas relacionados com o processamento dos dados, entre muitas outras funcionalidades. Tudo isto permite um contacto constante do utilizador com o chão de fábrica através somente do seu telemóvel ou qualquer meio digital apropriado.

Ao longo dos últimos anos, muitos autores propuseram e implementaram arquiteturas hierarquizadas e divididas em camadas. Peres *et al.* [10] desenvolveram um trabalho de índole teórica no qual propuseram uma arquitetura hierarquizada em 4 blocos, que se pode visualizar na Figura 2.1. Nesta identificam-se as 3 camadas referidas acima: aquisição de dados (camada física), organização e análise (camada digital) e visualização (camada do utilizador). O seu trabalho centrou-se em fornecer uma base para futuras implementações de algoritmos preditivos e mecanismos de tomada de decisão. Férnandez *et al.* [11] propuseram uma arquitetura em 5 níveis com vista à supervisão de uma instalação fotovoltaica. Implementaram *gateways* como ponte entre o nível físico e digital, os quais constantemente reuniam informação a partir de vários dispositivos industriais. A camada digital consistia numa plataforma IoT com mecanismos de armazenamento e processamento de dados, com uma interface gráfica (Grafana) a ser utilizada para a sua visualização. A implementação prática da sua solução mostrou-se muito viável, havendo aumentos significativos na capacidade de monitorização. Passou-se a ter registos históricos a cada 5 segundos face aos intervalos anteriores de 10 minutos entre leituras.

No contexto de desenvolver uma arquitetura que já implementa o bloco de processamento há também alguns trabalhos, mas poucos ainda com implementações validadas ao nível prático. Coelho *et al.* [12] desenvolveram uma arquitetura também dividida nas mesmas 3 camadas essenciais, aplicável à indústria da estampagem, com vista a facilitar operações de manutenção preditiva. Contudo, o trabalho teve apenas uma índole teórica, sendo proposta a arquitetura de recolha e processamento dos dados, mas não validada. O trabalho focou-se muito na comparação e avaliação do desempenho de uma série de algoritmos ML. Por outro lado, Pravin *et al.* [13] desenvolveram um trabalho muito completo, no qual expuseram uma arquitetura de recolha e processamento de dados ao nível de consumos energéticos num ambiente industrial. A sua abordagem permitiu realizar uma previsão para o dia seguinte de variáveis relacionadas com o consumo de energia no chão de fábrica, incluindo também a otimização de hiperparâmetros ao nível dos algoritmos preditivos. A implementação prática comprovou o sucesso dos algoritmos, que se revelaram aptos à realização das operações de previsão para o dia seguinte.

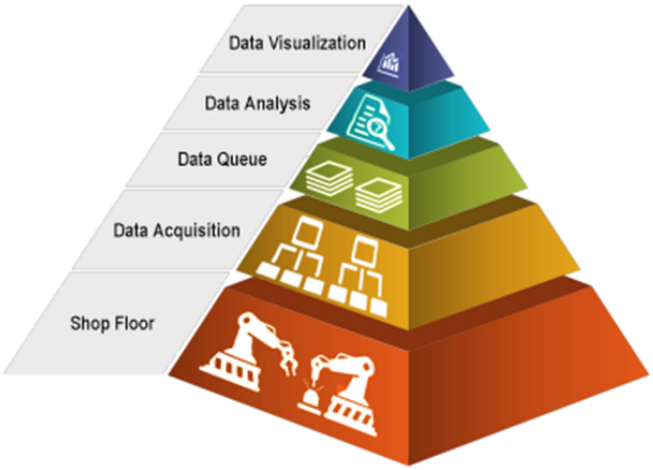


Figura 2.1: Organização base de uma arquitetura de recolha e processamento de dados (retirado de [10]).

2.1.2 Bases de Dados

Tendo em conta a enorme quantidade de informação que tem que ser recolhida em tempo real, proveniente de vários equipamentos e dispositivos no chão de fábrica, é de extrema importância fazer o seu armazenamento da forma mais eficaz possível. Isso é conseguido com recurso a bases de dados.

Atualmente, há um conjunto muito vasto de tipos de bases de dados, desde bases relacionais, orientadas para objetos, hierárquicas, entre outras. De entre os vários tipos, aquele que tem sido ao longo dos anos mais habitualmente empregue é a base de dados relacional. Nesta, os dados são organizados em tabelas, ou relações, as quais consistem num conjunto de linhas e colunas. A cada linha surge muitas das vezes associada uma chave primária, que corresponde à sua forma máxima de identificação. Este tipo de bases de dados é geralmente denominado bases SQL (*Structured Query Language*), visto que a sintaxe SQL é a normalmente empregue para trocar informações com a base de dados. Oracle, MySQL ou Microsoft SQL Server são algumas das bases de dados relacionais mais populares [14].

Por outro lado, ao longo dos últimos anos as necessidades da Indústria 4.0, a evolução da IoT e as tecnologias de *Big Data* têm levado a que haja o desenvolvimento de um grande número de bases de dados que não se enquadram na categoria das relacionais, sendo então denominadas bases não relacionais, ou NoSQL (*Not only SQL*) [15]. Neste grupo tem-se, por exemplo, as bases temporais ou de modelo coluna. Enquanto as bases de dados relacionais tendem a armazenar os dados sob a forma de tabelas, isso já não se verifica no caso das não relacionais. Nestas, os dados podem ser armazenados sob a forma de documentos (exemplos de formatos são o *JavaScript Object Notation*, JSON, e o *Extensible Markup Language*, XML), os quais podem conter um conjunto de tópicos e informações de diferentes formatos. Este tipo de estruturação permite uma maior flexibilidade em termos da adição de novas informações.

Em termos de comparação entre estes dois tipos de bases de dados, há já um extenso trabalho realizado por diversos autores [16]–[19]. Gomes *et al.* [16] fizeram uma compa-

ração entre uma base de dados relacional (MySQL) e uma não relacional (MongoDB), aplicada a um grupo de dados relativos e estatísticas do mercado comercial brasileiro. Também Györödi *et al.* [17] promoveram uma comparação entre uma base SQL (MongoDB) e uma NoSQL (MSSQL), neste caso usando uma aplicação *web* como caso de estudo. Ambos os trabalhos obtiveram resultados semelhantes, provando que as bases de dados NoSQL possuem na grande maioria das operações desempenhos superiores ao nível de tempos de operações face às bases SQL, com a MongoDB a demonstrar tempos de execução até 60% inferiores em relação a uma base SQL [16]. Já Jatana *et al.* [18] e Nayak *et al.* [19] promoveram uma análise mais ao nível teórico, expondo diferentes divisões ao nível das bases de dados não relacionais, assim como uma comparação mais aprofundada, juntamente com vantagens e desvantagens do uso de bases SQL e NoSQL.

Posto isto, na Tabela 2.1 apresenta-se um resumo com as principais características das bases de dados relacionais e não relacionais.

Tabela 2.1: Comparação entre bases de dados relacionais e não relacionais (adaptado de [20]).

	Relacionais (SQL)	**Não Relacionais (NoSQL)**
Modelo de armazenamento	Tabelas com um número fixo de linhas (registos) e colunas (atributos)	Vários modelos: chave-valor (*key-value*), documentos, colunas, gráficos, etc
Histórico de desenvolvimento	Apareceram por volta da década de 1970, com o propósito de diminuir a duplicação dos dados	Surgiram na década de 2000, com foco no aumento da escalabilidade, assim como na maior capacidade de armazenamento, para se adaptar às necessidades cada vez mais exigentes das aplicações e processos
Exemplos	Oracle, MySQL, MSSQL, PostgreSQL	MongoDB (modelo documento), DynamoDB (modelo chave-valor), Cassandra (modelo coluna), Neo4j (modelo gráfico)
Escalabilidade	Maioritariamente vertical (necessidade de adicionar mais memória ou armazenamento ao servidor já existente)	Horizontal (adição de novos servidores para cumprir com as exigências do sistema)
Organização (*schema*)	Rígida (a organização é fixa, pouco flexível)	Flexível (permite mais facilmente a adição de novos parâmetros)
***Data-to-object mapping* ***	Baseado na sua organização fixa (*schema*)	Maior flexibilidade. Organizam os dados de formas distintas e baseadas em diferentes formatos

* Refere-se ao processo de associação entre os dados que se obtêm numa dada aplicação e a sua representação na base de dados.

Analisando as características de ambos os tipos de bases de dados expostas na Tabela 2.1, pode-se verificar que as bases NoSQL apresentam uma maior versatilidade, possuindo vários modelos de armazenamento distintos. Apresentam flexibilidade acrescida, facilitando a inserção de novos dados e de diferentes tipos, e a sua escalabilidade horizontal permite que estejam totalmente adaptadas às exigências crescentes das operações em tempo real nos ambientes industriais [16], [17], [21]. Por fim, o seu aumento de popularidade leva a que haja um aumento da procura, informação e apoio existente relativamente a estas bases de dados [15]. Num ambiente industrial onde cada vez mais equipamentos surgem conectados e com necessidades constantes de monitorização, a escolha de uma base de dados NoSQL torna-se portanto a mais acertada.

2.1.3 *Digital Twins*

Aliados à Indústria 4.0 e à transformação digital que se tem assistido ao nível industrial, a utilização de *Digital Twin*s (DTs) tem-se tornado cada vez mais popular. De uma forma muito resumida, um DT corresponde a uma representação virtual de um objeto real, a qual é capaz de estar conectada ao sistema físico e em constante comunicação com o mesmo [22]–[24]. Apesar de estar muito associado às noções atuais de Indústria 4.0, este termo foi inicialmente introduzido em 2002 por Michael Grieves segundo uma nomenclatura diferente (*Mirrored Space Models*), mas já partilhando os ideais associados a espaço virtual, real e a sua conexão através de um fluxo de dados [22].

A utilização de DTs não se limita apenas a aplicações industriais e a monitorização. Nguyen *et al.* [23] introduziram as potencialidades dos DTs a áreas como a condução autónoma de veículos ou até mesmo o aumento da segurança rodoviária (através da utilização de representação virtual dos veículos na estrada); Pires *et al.* [24] sugeriram a utilização dos DTs para modelação e monitorização de redes elétricas de áreas urbanas, na área do *marketing* e compras *online* (criação de réplicas digitais dos clientes que permitem analisar diferentes peças de roupa, por exemplo), ou até mesmo na área da saúde, com aplicações viradas para os equipamentos (controlo remoto de máquinas em operações) ou para os utentes (DT de um utente dá informações em tempo real ao médico sobre o seu estado).

Segundo Kritzinger *et al.* [22], apesar de habitualmente se empregar o termo "*Digital Twin*" para todas as formas virtuais de representação de algo real, há uma distinção muito importante a ser feita, a qual pode ser constatada na Figura 2.2. Há portanto uma possível divisão em três noções distintas:

- *Digital Model* - corresponde de facto a uma representação virtual de algo físico existente, mas não possui qualquer forma de troca de dados entre o elemento físico e o elemento virtual. Corresponde basicamente a um modelo que ajuda a simular, num ambiente digital, algo relacionado com dado componente ou sistema físico (Figura 2.2a);
- *Digital Shadow* - neste caso, já existe um fluxo de dados unidirecional entre sistemas físico e digital. Isso significa que o sistema digital, apesar de não ter capacidade de enviar automaticamente dados para o físico, é capaz de estar a receber, de forma autónoma, dados em tempo real desse sistema. Deste modo, já existe uma possibilidade de proceder ao uso desta representação digital para fazer uma monitorização em tempo real do sistema físico (Figura 2.2b);
- *Digital Twin* - no caso de um DT, já pode existir um fluxo de dados bidirecional entre o sistema físico e a sua "cópia" digital. Com isto, é possível não só a monitorização do sistema real com recurso ao DT, como também o próprio controlo do sistema real com recurso somente à sua representação digital (Figura 2.2c).

Num contexto industrial, o uso de *digital shadows* numa primeira instância permite já uma capacidade de monitorização em tempo real de um componente, equipamento ou processo industrial através da sua representação virtual. Posteriormente, o uso de DTs permite também que, a partir dessa representação virtual, haja capacidade de fazer o controlo do próprio sistema físico. Adicionalmente, os dados dos DTs obtidos em tempo real podem ser diretamente processados por outras ferramentas que podem identificar anomalias, realizar previsões de necessidades de manutenção ou otimização de certos

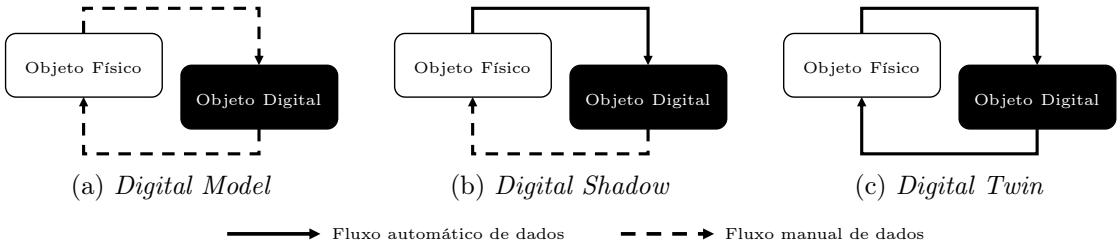


Figura 2.2: Diferentes noções associadas a *digital twins* (adaptado de [22]).

parâmetros. Por sua vez, os resultados do processamento podem ser remetidos de volta ao DT, que de forma automatizada os pode passar para o próprio equipamento físico, fazendo a alteração do seu estado.

Tudo isto reforça a ideia de que, num ambiente industrial à luz das noções da Indústria 4.0 onde há cada vez mais a digitalização e automação dos processos, a utilização de DTs constitui uma das ferramentas com mais potencialidades e capacidades.

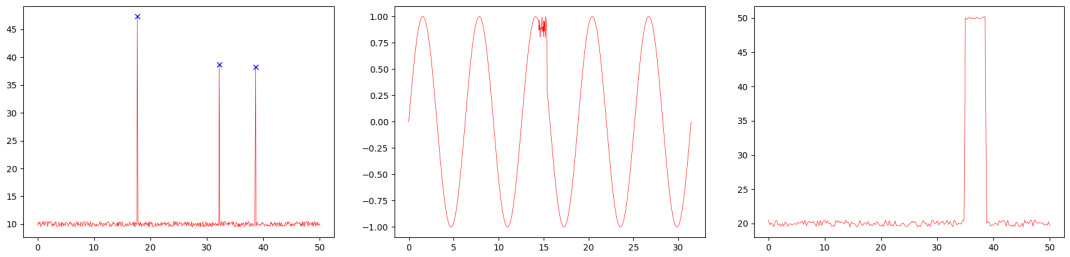
2.1.4 Deteção de Anomalias

No âmbito do desenvolvimento do agente inteligente, a deteção de anomalias é algo de extrema importância. Uma anomalia corresponde a um qualquer ponto ou sequência de pontos que se desvia do normal comportamento de uma variável [25]. Através da deteção de anomalias em tempo real ao nível dos consumos energéticos, é possível gerar notificações e avisos, encontrar oportunidades de melhoria e eventualmente otimizar estes consumos energéticos ao nível do chão de fábrica.

Olhando para uma série temporal, como é o caso da análise de consumos energéticos em tempo real, há possibilidades de ocorrência de vários tipos de anomalias [7], [25], [26] (cuja representação gráfica é apresentada na Figura 2.3):

- **Anomalias pontuais** - a forma mais simples de anomalias. Analisando um determinado período temporal, correspondem simplesmente a um ponto que se destaca claramente dos demais, com um valor excessivamente elevado ou reduzido (Figura 2.3a);
- **Anomalias contextuais** - apenas são consideradas anomalias no contexto onde ocorrem, pelo que a sua deteção e até mesmo definição já é mais complexa. Constituem pontos ou sequências de pontos que se desviam do comportamento dos restantes em seu redor, mas que olhando para uma janela temporal maior possuem valores que se enquadram com os demais (Figura 2.3b);
- **Anomalias sequenciais** - enquanto anomalias pontuais correspondem a pontos isolados que se desviam das tendências dos restantes, neste caso anomalias sequenciais correspondem a um conjunto de pontos consecutivos que possuem valores claramente distintos dos demais (Figura 2.3c).

Num contexto de análise de consumos energéticos de um qualquer equipamento é possível fazer a identificação destes tipos de anomalias, especialmente as pontuais e as sequenciais: anomalias pontuais podem constituir picos ou quebras momentâneas de consumos; já anomalias sequenciais podem corresponder a períodos prolongados de falha de energia ou de consumos energéticos excessivos num equipamento. Torna-se então



(a) Anomalias pontuais (b) Anomalia contextual (c) Anomalia sequencial

Figura 2.3: Diferentes tipos de anomalias numa série temporal.

importante que, para promover a deteção de anomalias, se desenvolva um modelo capaz de identificar diferentes tipos de anomalias.

Classificação dos algoritmos

Tal como existe uma grande variedade ao nível dos tipos de anomalias existentes, há também um conjunto muito variado e alargado de métodos e algoritmos que permitem a deteção de anomalias, muitos deles focados inclusive na deteção de um dado tipo de anomalias. Esta grande variedade leva a que haja um conjunto de possíveis divisões dos algoritmos, tal como a divisão conforme a sua família ou o seu tipo de aprendizagem [25], [27].

Uma das classificações mais amplamente usada e reconhecida é quanto à sua aprendizagem, havendo três grandes tipos de algoritmos:

- **Algoritmos supervisionados** - constituem o grupo de algoritmos que necessita obrigatoriamente de um conjunto de dados de treino. Nestes devem constar quer dados que reflitam comportamentos "regulares" quer comportamentos anómalos. A partir destes dados os algoritmos conseguem extrair padrões e comportamentos, e consequentemente classificá-los como normais ou anormais. Uma das principais desvantagens deste tipo de algoritmos é o facto da sua eficácia estar muito dependente da qualidade do grupo de dados de treino - estes devem ser quase manualmente recolhidos e escolhidos. Adicionalmente, os algoritmos são codificados para identificar apenas anomalias que tenham constado no seu conjunto de dados de treino, pelo que qualquer outro eventual tipo de anomalia corre o risco de não ser identificada;
- **Algoritmos não supervisionados** - ao contrário dos anteriores, este tipo de algoritmos não necessita de um conjunto de dados de treino explícito. Neste caso, os algoritmos analisam diretamente os dados e identificam autonomamente padrões e tendências. Quaisquer pontos ou grupos de pontos com comportamentos anormais, menores frequências de ocorrência, entre outros, acabam por ser identificados como anomalias. Este tipo de algoritmos é o mais utilizado em termos de deteção de anomalias sendo o principal exemplo destes as redes neuronais. A principal vantagem face aos métodos supervisionados é a sua maior versatilidade, uma vez que são capazes de detetar vários tipos de anomalias, inclusivamente aquelas que nunca antes foram vistas ou identificadas;

- **Algoritmos semi-supervisionados** - este tipo de algoritmos acabam por combinar as características relativas aos supervisionados e não supervisionados. Acabam por ser semi-supervisionados uma vez que utilizam um conjunto de dados de treino, mas apenas para identificar e aprender qual o comportamento "normal" da série temporal [27]. Assim, quando aplicados a uma série temporal de teste ou real, qualquer comportamento que fuja ao normal que resultou do processo de aprendizagem acaba por ser considerado uma anomalia. Assim, este tipo de algoritmos acaba por combinar as vantagens dos dois anteriores: têm maior capacidade para identificar um maior conjunto de anomalias (característica dos não supervisionados) e simultaneamente mais robustez na identificação das anomalias (característica dos supervisionados), dado que não se baseiam apenas na sua capacidade individual de identificar padrões comportamentais na série temporal.

Quanto à família, existe um grande número de possíveis classificações dos algoritmos, desde os métodos de previsão, onde se recorre a técnicas de previsão de séries temporais para fazer uma previsão dos próximos instantes dos dados, métodos baseados na distância, que identificam anomalias baseando-se na distância relativamente a outros pontos da série temporal, até métodos de distribuição, que utilizam noções de estatística e probabilidades. Uma explicação mais detalhada pode ser encontrada no trabalho desenvolvido por Schmidl *et al.* [27].

Algoritmos mais comuns para deteção de anomalias

Como visto acima, existe atualmente uma grande variedade de algoritmos de deteção de anomalias, que tem levado autores a proceder a trabalhos de comparação com vista a encontrar aqueles que revelam melhores desempenhos [26]–[28].

Cook *et al.* [26] realizaram um levantamento de técnicas e métodos de deteção de anomalias em séries temporais que se enquadram no paradigma IoT. Um exemplo disso é a aplicação destes algoritmos a valores obtidos em tempo real através de sensores instalados em diversos equipamentos. Em termos dos métodos mais abordados, tem-se um conjunto variado de redes neuronais, desde artificiais (ANNs) até recorrentes (RNNs), as quais apresentam evoluções como os métodos *Long-Short Term Memory* (LSTM) e *Graded Recurrent Units* (GRU), e ainda *autoencoders* (AEs) e as suas diversas formas; tem-se também métodos mais baseados em noções de estatística, tais como o *Autoregressive Integrated Moving Average* (ARIMA). Todos os métodos anteriores estão relacionados com séries temporais de uma única variável, pelo que para deteção de anomalias "multivariável" identificam-se, para além de redes neuronais, métodos como *Support Vector Machine* (SVM), *Local Outlier Factor* (LOF) e *Random Forest* (RF). À semelhança dos autores anteriores, Choi *et al.* [28] promoveram uma análise de métodos para deteção de anomalias aplicada especificamente a séries temporais e a várias aplicações ao nível industrial. De entre vários algoritmos referem, por exemplo, a aplicação de modelos híbridos de *autoencoders* com redes LSTM no paradigma da indústria da manufatura, mais propriamente no *Smart Manufacturing*. Ao nível da gestão energética, referem novamente um conjunto muito variado de redes neuronais. Adicionalmente, realizaram um estudo muito focado ao nível de redes neuronais, aplicado a 3 diferentes conjuntos de dados. Como resultados, verificaram que não existe propriamente um determinado tipo de rede capaz de se superiorizar às demais, pelo que dependendo do conjunto de dados houve diferentes algoritmos a apresentar o melhor desempenho.

Já Schmidl *et al.* [27] realizaram um estudo mais completo, no qual analisaram o desempenho de 71 algoritmos de deteção de anomalias em função quer da qualidade dos resultados, quer do tempo e memória utilizada no processamento. Foram utilizados vários conjuntos de dados (mais precisamente 976 séries temporais), e também alguns conjuntos de dados de treino para métodos supervisionados e semi-supervisionados. Em termos de tempo de processamento, verificou-se que métodos supervisionados e semi-supervisionados, regra geral, são os mais lentos, o que se deve principalmente à maior necessidade de treino dos mesmos. Dentro destes métodos encontra-se o ARIMA (referido acima). Comparando os algoritmos em termos de desempenho, verifica-se que:

- Algoritmos focados em séries temporais com uma variável apresentam desempenhos ligeiramente melhores face aos algoritmos multi-variável;
- Apesar dos algoritmos supervisionados estarem dotados de uma fase de treino e possuírem algum conhecimento prévio de anomalias, não apresentaram propriamente maiores capacidades de deteção de anomalias face a métodos semi-supervisionados ou não supervisionados;
- Muito poucas implementações apresentaram de facto elevada robustez e eficácia na deteção de anomalias, sendo de destacar os métodos *K-Nearest Neighbours* (KNN), *k-means* e o LOF;
- De salientar por fim que vários dos métodos referidos por Cook *et al.* [26], tais como LSTM, RF ou ARIMA, apresentam dos melhores resultados globais de todos os algoritmos comparados.

Posto isto, pode-se concluir que há um conjunto muito variado de algoritmos que podem ser usados para a deteção de anomalias, desde métodos como o KNN, SVM até redes neuronais e *autoencoders*. No Apêndice A é conduzida uma explicação mais detalhada relativa ao funcionamento de um conjunto de métodos de deteção de anomalias habitualmente utilizados. Já na secção seguinte é efetuada uma breve comparação relativamente a estes métodos, expondo-se as suas vantagens e desvantagens.

Comparação dos métodos de deteção de anomalias

Na secção anterior foi feita uma introdução relativa aos métodos de deteção de anomalias mais comuns e que geralmente revelam melhores desempenhos. Adicionalmente, no Apêndice A são apresentadas explicações mais detalhadas no que toca aos princípios de funcionamento de alguns dos algoritmos referidos e cuja utilização foi ponderada. Posto isto na Tabela 2.2 é apresentada uma comparação global de um conjunto desses algoritmos muito utilizados na tarefa de deteção de anomalias.

Olhando para as diferentes opções, tem-se que algoritmos como o SVM, o *K-means clustering* e o RF se revelam eficazes à utilização em *datasets* (conjuntos de dados) com um número elevado de *features* (dimensões), sendo uma das suas principais vantagens. Adicionalmente, estes algoritmos possuem um número relativamente baixo de hiperparâmetros, tornando a sua implementação não muito complexa. Outros, como o LOF ou o KNN, apresentam uma maior simplicidade e facilidade de implementação, possuindo um baixo custo computacional assim como poucos hiperparâmetros a manipular. Já as redes neuronais e os *autoencoders* possuem um maior número de hiperparâmetros a manipular, tais como o número de camadas, de nodos por camada, funções de ativação, entre outros.

Tabela 2.2: Comparação entre algoritmos de deteção de anomalias geralmente utilizados.

Algoritmos	**Classificação**	**Funcionamento geral**	**Custo computacional**	**Nº de parâmetros**
SVM	S	Separação dos dados em classes através da definição de hiperplanos	Elevado	Elevado
LOF	NS	Determinação de densidades de pontos e probabilidades de ocorrência de anomalias	Reduzido	Reduzido
KNN	S	Baseia-se nas distâncias entre os pontos e os seus K vizinhos mais próximos	Médio	Reduzido
K-means	NS	Agrupamento de pontos em *clusters* (conjuntos de pontos com características semelhantes)	Médio	Médio
Redes neuronais	NS	Junção de várias camadas, ligadas entre si, que simulam o processo de raciocínio do cérebro humano	Elevado	Elevado
RF	S	Conjugação de várias árvores de decisão e junção dos resultados de cada uma	Elevado	Médio
AE	NS	Extração de características e reconstrução do *input*	Elevado	Elevado

Legenda: S - supervisionado, NS - não supervisionado

Isso torna a sua implementação um pouco mais complexa. As redes neuronais, apesar do seu custo computacional elevado especialmente durante a fase de treino, apresentam grande eficiência ao nível da obtenção de resultados. Por fim, os *autoencoders* possuem uma grande versatilidade, podendo-se adaptar a um conjunto amplo de aplicações, que vão desde a redução de ruído em imagens até à deteção de anomalias. Adicionalmente, o modo como promovem a deteção de anomalias é também facilmente compreensível, sendo que não é necessário no seu treino "aprenderem" o que é uma anomalia: apenas necessitam de aprender o comportamento normal, e automaticamente determinam o que é uma anomalia.

Em termos das desvantagens, o SVM, apesar da sua aparente simplicidade, acaba por ser um processo bastante lento com elevados tempos de processamento quer ao nível do treino, quer depois ao nível da implementação. Adicionalmente, quando o *dataset* se torna muito grande (para além de muitas dimensões tem também muitas amostras), o SVM torna-se muito ineficaz. O LOF possui especialmente alguma falta de consistência ao nível da definição do que é uma anomalia, uma vez que não há a definição concreta de um valor base a partir do qual um dado ponto constitui uma anomalia. O algoritmo KNN, à semelhança do SVM, acaba por ser lento (implica o cálculo de distâncias para todos os pontos do *dataset* em função dos seus K vizinhos), e a definição do número de vizinhos pode-se por vezes revelar muito complicada. O *K-means* possui problemas ao nível dos resultados e da sua robustez: correndo duas vezes o algoritmo há uma forte possibilidade de se obter resultados distintos. Além disso, possui também uma grande sensibilidade à escala dos valores fornecidos, pelo que procedendo, por exemplo, a uma normalização dos mesmos, os resultados acabam por dar valores completamente distintos. O RF tem especialmente problemas ao nível do custo computacional, que é bastante elevado, e em termos de tempos de treino, dado que constrói um conjunto de árvores de decisão (consultar Apêndice A, Secção A.6) e basicamente cada uma delas tem que fornecer um resultado. Por fim, os *autoencoders* também acabam por ter um custo computacional elevado, mas mais na fase de treino, e acabam por ser bastante

sensíveis aos próprios dados de treino: caso o *dataset* de treino contenha anomalias, os *autoencoders* correm o risco de "aprender" esse como sendo o comportamento normal dos dados. Por consequência, na fase de implementação acabam por não identificar esses tipos de pontos como anomalias.

Concluindo, como foi possível verificar ao longo desta secção há um conjunto de algoritmos, cada um com as suas vantagens e desvantagens, que podem ser aplicados à tarefa de deteção de anomalias. Isso leva a que seja inicialmente complicado saber se dado algoritmo é o mais adequado à aplicação em causa. Posteriormente, já com um algoritmo escolhido, a existência de um conjunto de diferentes hiperparâmetros pode levar a que sejam necessárias um conjunto de iterações até se conseguir otimizar o desempenho do algoritmo relativamente à aplicação em questão. Para facilitar todo este processo, surgiu o AutoML, uma ferramenta que permite a automatização de grande parte do processo de construção e aplicação dos modelos. Na secção seguinte é conduzida uma pequena introdução a este conceito.

2.1.5 AutoML

O AutoML constitui uma tecnologia recente e emergente que permite, como o nome indica, automatizar as tarefas associadas à implementação de modelos de ML, levando à aceleração do processo de construção dos modelos e aumento da sua eficiência [29].

Geralmente, a implementação do ML "convencional" implica um conjunto de passos a realizar sequencialmente: (1) recolha dos dados, (2) processamento dos dados, (3) escolha e construção de um modelo adequado para o problema em questão, (4) treino e teste desse modelo e (5) processo iterativo de análise de resultados com alteração/otimização dos hiperparâmetros. Posteriormente, pode haver a necessidade de se fazer a comparação de outros tipos de modelos, e portanto há nova repetição dos passos (4) e (5). Tudo isto torna o processo bastante exaustivo.

Posto isto, a utilização do AutoML tenta colmatar esta certa ineficiência da implementação "manual" de modelos de ML. Com o AutoML, é possível haver uma automatização ao nível da escolha do modelo, otimização de hiperparâmetros e avaliação do mesmo. Isto permite que o foco esteja muito mais concentrado na análise dos resultados e não tanto em todo o processo de construção e melhoria dos modelos. A Figura 2.4 resume bastante bem a potencialidade do AutoML e os passos que permite automatizar.

Contudo, é relevante salientar que a utilização do AutoML para construir modelos de elevada qualidade exige por si só algum tempo e recursos; certos problemas como a definição das restrições inerentes aos hiperparâmetros a ser explorados pelo AutoML ou os *cold-starts* (processo pode começar por uma combinação fraca de hiperparâmetros e ficar muito tempo preso num modelo indesejável) devem ser tidos sempre em conta, e deve ser sempre efetuada uma cuidada configuração do processo de AutoML [31].

Aliado ao AutoML tem-se também o *meta-learning*, que consiste na utilização de conhecimento prévio relativo à aplicação de diferentes algoritmos em diferentes cenários para a construção de um modelo aplicado ao problema a resolver. Esse modelo acaba por combinar características desses diferentes algoritmos conforme a similaridade que revelam face ao *dataset* atual. O *meta-learning* pode portanto ser usado para complementar a utilização do AutoML, resolvendo, por exemplo, o problema dos *cold starts* referido anteriormente ao encontrar desde logo modelos mais otimizados para o problema em causa.

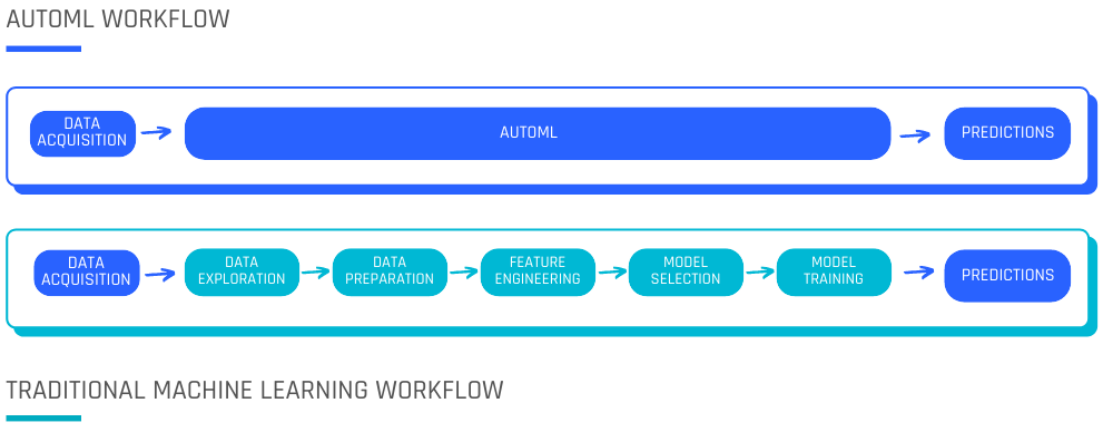


Figura 2.4: Potencialidades do AutoML (retirado de [30]).

Em termos de implementação, à semelhança do ML, também para o AutoML existem um conjunto de bibliotecas Python que auxiliam bastante a sua utilização. Zimmer *et al.* [32], por exemplo, introduzem a utilização da biblioteca Auto-PyTorch para otimização de hiperparâmetros via AutoML de modelos com redes neuronais. Outras bibliotecas Python geralmente empregues são a Auto-SKLearn ou Auto-Keras.

2.2 Levantamento das Tecnologias

Feita uma introdução a alguns dos conceitos mais teóricos, nesta parte são introduzidas algumas tecnologias e ferramentas que permitiram a implementação prática dos conceitos teóricos introduzidos anteriormente.

2.2.1 Cloud2Edge

O Cloud2Edge (C2E) consiste numa combinação de dois projetos desenvolvidos pela Eclipse Foundation [33], o Eclipse Hono [34] e o Eclipse Ditto [35]. Esta junção tem como objetivo agilizar a comunicação entre dispositivos, pelo que através de protocolos como *Hypertext Transfer Protocol* (HTTP), *Message Queuing Telemetry Transport* (MQTT) ou *Advanced Message Queuing Protocol* (AMQP), é possível fazer o registo de dispositivos no Hono e enviar mensagens relativas ao seu estado. Por conseguinte, com recurso ao Ditto podem-se criar DTs dos mesmos. Depois, é possível ter uma aplicação a comunicar diretamente com o Ditto e a analisar o estado dos equipamentos ou, na direção inversa, fazendo através do DT a alteração do estado do equipamento físico. De seguida são apresentadas explicações mais detalhadas relativamente ao Eclipse Hono e Ditto.

Eclipse Hono

O Eclipse Hono [34] providencia uma plataforma *open-source* que permite conectar um número elevadíssimo de dispositivos IoT e interagir com eles independentemente do protocolo de comunicação utilizado. Como se pode observar na Figura 2.5, é possível comunicar com os dispositivos IoT por CoaP (*Constrained Application Protocol*), MQTT, HTTP, entre outros, e com outros serviços superiores através de AMQP ou Kafka.

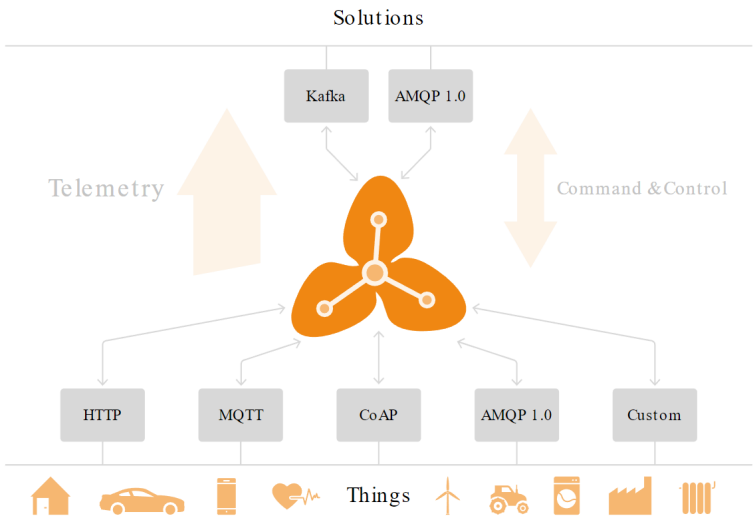


Figura 2.5: Visão geral do funcionamento do Eclipse Hono (retirado de [34]).

Com o Eclipse Hono há a possibilidade de ocorrência de três grandes fluxos de informação (dois deles representados na Figura 2.5):

- **Telemetria** - fluxo maioritariamente unidirecional, que vem dos dispositivos de baixo nível para aplicações e servidores de alto nível. Este pode conter, por exemplo, dados de leituras de sensores;
- **Eventos** - também correspondem a mensagens dos dispositivos para servidores e aplicações de alto nível. São menos frequentes, mas mais importantes que as mensagens de telemetria;
- **Controlo e Comando** - neste caso já se tem um fluxo bidirecional dos dados, com os dispositivos a enviarem e receberem respostas das aplicações e servidores de alto nível. Os dados enviados podem ser relativos a determinadas atualizações de *firmware* ou configurações de parâmetros, por exemplo.

Posto isto, o Eclipse Hono acaba por funcionar como um intermediário - *broker* - que consegue fornecer uma interface comum de comunicação entre dispositivos IoT ao nível físico e aplicações e servidores na *cloud*, que posteriormente podem efetuar o armazenamento ou o tratamento dos dados.

Eclipse Ditto

O Eclipse Ditto [35] trata-se novamente de um *software open-source*, aplicado num contexto IoT, mas neste caso com o intuito de criar e gerir os DTs já abordados na Secção 2.1.3. O seu objetivo não é tanto fornecer uma plataforma IoT totalmente integrada, mas sim fornecer APIs (*Application Programming Interfaces*) que simplificam a utilização de dispositivos conectados via Eclispe Hono [35], por exemplo, tal como se pode observar na Figura 2.6.

Com o Eclipse Ditto, é então possível criar réplicas virtuais de objetos e dispositivos reais. Com estas réplicas, é possível de forma muito mais simplificada aceder a certos aspetos do objeto real num ambiente digital ou até mesmo simular esse próprio

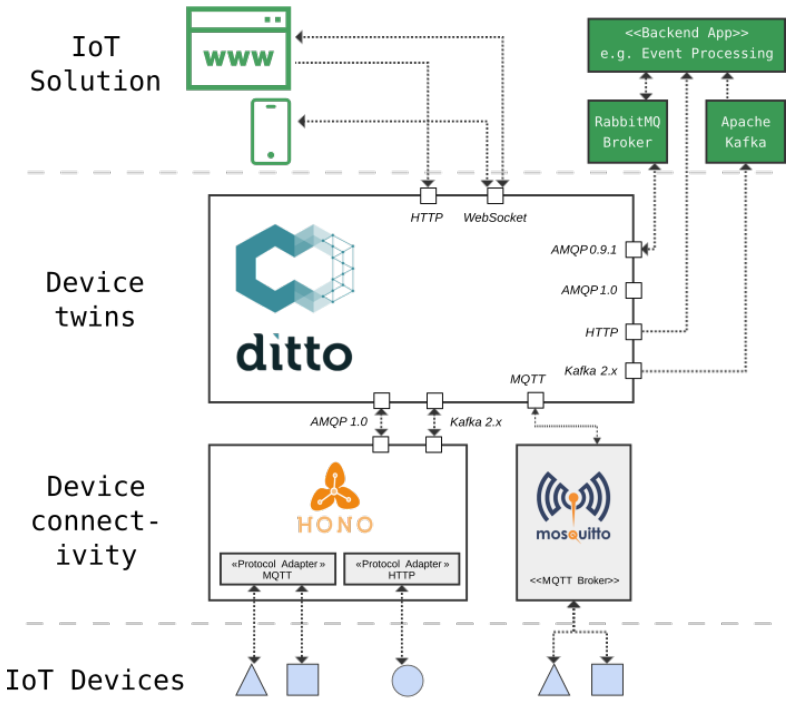


Figura 2.6: Visão geral da aplicabilidade do Eclipse Ditto (retirado de [35]).

objeto. Adicionalmente, a implementação de DTs na indústria permite que através de aplicações para o efeito, seja possível controlar o estado do dispositivo físico recorrendo exclusivamente à sua representação virtual.

Tendo um equipamento industrial conectado ao Hono, este consegue via MQTT, por exemplo, enviar de forma periódica e constante dados relativos às suas variáveis (tais como valores de corrente e tensão). Tendo uma ligação entre Hono e Ditto configurada, de forma automatizada há a atualização do estado do DT associado ao equipamento. A partir daqui, serviços secundários podem ser utilizados para captar estas atualizações do DT, fazer processamento de dados e enviar comandos de volta para a réplica digital, no fluxo inverso. Por fim, é possível através desses comandos controlar o dispositivo físico.

2.2.2 Nexeed MES

O Nexeed MES [36] corresponde ao sistema de execução e manufatura (MES) desenvolvido e implementado pela Bosch.

Segundo Shojaeinasab et al. [37], o conceito de *Manufacturing Execution System* (MES) surgiu em seguimento do conceito de *Enterprise Resource Planning* (ERP). Um sistema ERP engloba funções relativas ao planeamento da produção, controlo de inventário ou contabilidade, funções muito mais ao nível administrativo. Isso leva a que o sistema revele algumas incapacidades na gestão em tempo real da informação que vem do chão de fábrica. Posto isto, o MES surge como forma de fazer esta conexão entre o chão de fábrica e a camada superior onde se insere o ERP, permitindo que haja uma troca contínua de dados desde um nível mais baixo da produção até operações de controlo de qualidade e gestão da produção [37], [38]. Dentro das diversas funções de um MES, Jaskó et al. [38] destacam as seguintes:

- **Recolha de dados** - contínua recolha de dados ao nível do chão de fábrica, relativos aos equipamentos e à produção, e armazenamento dos mesmos em bases de dados para o efeito;
- **Monitorização da produção** - isto envolve a monitorização de todo o processo da produção, desde que a matéria-prima entra na fábrica até que o produto é finalizado. Envolve o registo dos lotes, encomendas, equipamentos, e o acompanhamento de cada produto ao longo de todo o seu processo de produção;
- **Avaliação de desempenho** - através da monitorização da produção e da recolha dos dados pode ser feito um processamento desses dados de modo a avaliar o desempenho de determinado processo produtivo, assim como de um dado equipamento;
- **Visualização da informação** - a existência de uma plataforma de visualização torna-se extremamente útil para verificar as tendências da produção, sendo aplicável quer a um nível mais elevado para analistas de dados, quer também ao nível do chão de fábrica;
- **Outras funções** - depois, há um conjunto de outras funções mais ligadas já aos ERPs, tais como gestão de recursos, calendarização e planeamento da produção, que irão trabalhar em conjunto com todas as outras referidas acima.

Olhando agora para a plataforma desenvolvida pela Bosch, o Nexeed MES, como se pode ver na Figura 2.7, é possível identificar estas funções e muitas mais a si associadas.

É por fim de salientar que as vantagens da implementação desta plataforma ao nível do chão de fábrica Bosch são notórias. Em Homburg, houve uma redução de 40% no consumo de energia e poupanças na ordem dos 800 mil euros em ar comprimido para operações de manutenção através da utilização do Nexeed MES [39].

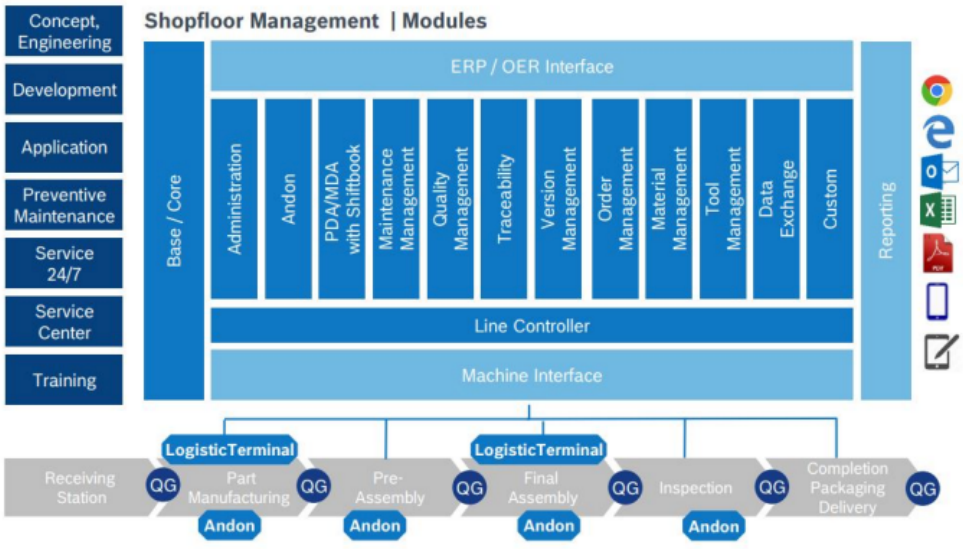


Figura 2.7: Potencialidades da plataforma do Nexeed MES (retirado de [36]).

2.2.3 InfluxDB

O InfluxDB corresponde a uma base de dados temporal, *open-source*, a qual é desenvolvida pela InfluxData [40]. Tratando-se de uma base de dados temporal está otimizada para gerir, em tempo real, milhares ou até milhões de pontos por segundo, providenciando assim uma elevada capacidade de recolha de dados em tempo real [41], [42]. Esta capacidade das bases de dados temporais, onde se enquadra o InfluxDB, tem-se refletido no aumento significativo das suas implementações em aplicações ligadas ao IoT e à Indústria 4.0. Inclusivamente, dentro das bases temporais, para sistemas de gestão e armazenamento de dados, o InfluxDB tem sido mesmo a base de dados mais empregue, como se pode ver na Figura 2.8 [43].

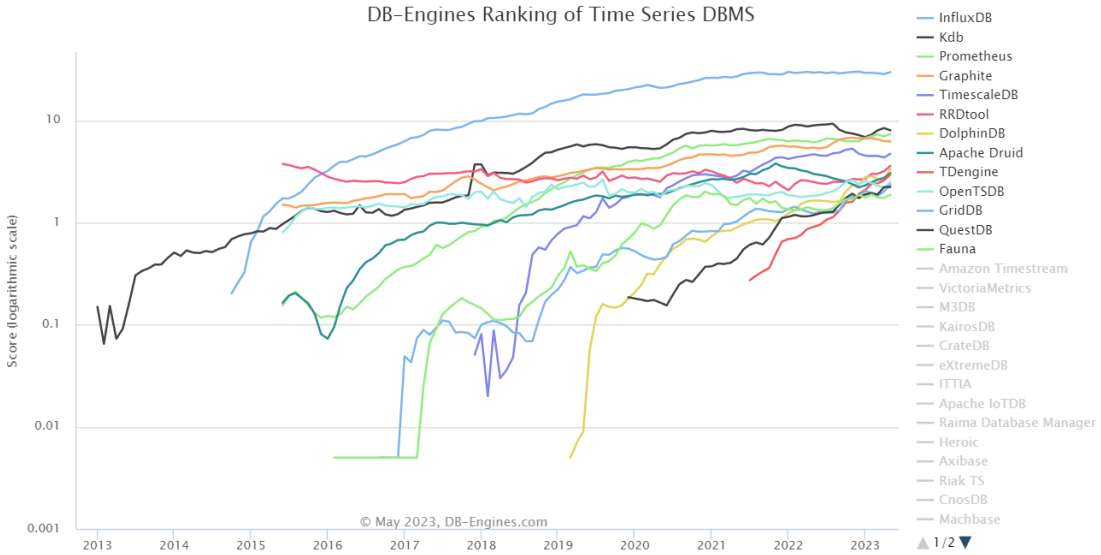


Figura 2.8: Evolução da popularidade de um conjunto de diferentes bases de dados temporais nos últimos anos (retirado de [43]).

Olhando de uma forma muito sucinta para a forma como os dados são organizados ao nível desta base de dados, na Tabela 2.3 são apresentadas as principais diferenças face a uma base de dados relacional, do tipo SQL.

Tabela 2.3: Comparação entre a organização de uma base de dados SQL e o InfluxDB

Bases SQL	**InfluxDB**
Tabelas	*Measurements*
Colunas (indexadas)	*Tags*
Colunas (não indexadas)	*Fields*
Linhas	*Points*

Antes de mais, no InfluxDB cada base de dados tem o nome de *bucket*, sendo que nele são introduzidos um conjunto de dados. Dentro destes dados, são possíveis identificar *measurements*, equivalentes às tabelas numa base SQL. Cada *measurement* possui *fields* que equivalem a colunas não indexadas numa base SQL, e *tags* que definem um conjunto

de campos e parâmetros associados à *measurement* a monitorizar. Por fim, associados a um determinado ponto temporal, têm-se os *points*, o equivalente a linhas numa tabela SQL, que possuem para determinada *measurement* todas as *tags* e *fields* injetadas num dado ponto temporal [44].

2.2.4 Grafana

O Grafana corresponde a mais um software *open-source*, neste caso desenvolvido pela organização Grafana Labs [45]. Corresponde a uma plataforma de visualização e análise de dados, sendo capaz de se conectar a uma fonte de dados temporais (como o InfluxDB) e traduzir esses dados sob a forma de gráficos e tabelas de elevada interatividade.

O Grafana possui um conjunto de potencialidades, sendo possível retirar dados específicos de uma determinada fonte, visualizá-los, definir alertas e ainda fazer algum processamento dos mesmos. É possível construir mapas de calor, histogramas, gráficos de barras, mapas geográficos entre muitas outras formas de visualização [46]. Para complementar a parte da visualização, o Grafana oferece também uma componente alarmística, pelo que não só é possível a visualização de alertas nas interfaces como também conectar o Grafana ao email, e enviar notificações desses alertas para os utilizadores. Também em termos de aquisição de dados, o Grafana oferece a possibilidade de conexão a um conjunto de diversas fontes, tais como o InfluxDB.

Todas estas características têm levado a que, nos últimos anos, tenha havido um conjunto de grandes entidades a adotar esta plataforma para os seus casos de estudo, de entre as quais o Ebay, a Microsoft ou a Tripadvisor [47].

2.2.5 Docker

O Docker corresponde a uma plataforma de desenvolvimento e partilha de aplicações que permite desenvolver código e realizar testes de uma forma muito mais rápida e eficiente [48]. Com este *software* é possível compactar uma determinada aplicação e corrê-la num ambiente completamente isolado, denominado *container*. Os *containers* são extremamente leves para o *host* do computador, e possuem toda a informação necessária para que de facto se possa executar uma aplicação num determinado computador, independentemente do que está instalado no mesmo. Como se pode ver na Figura 2.9, num computador é possível também correr um conjunto de diferentes *containers*, que podem ter quer aplicações completamente distintas, quer a mesma aplicação a correr, mas em ambientes isolados.

Em termos de arquitetura, os *containers* Docker são basicamente uma instância executável das imagens Docker. Cada imagem possui um conjunto de instruções, reunidas num ficheiro denominado "*Dockerfile*", as quais fornecem informações sobre a forma como deverá correr o respetivo *container*. Cada *Dockerfile* é composto por um conjunto de instruções, as camadas, pelo que sempre que se quer correr determinado *container*, estas instruções são executadas uma a uma. Caso não haja alterações a este ficheiro base, não há necessidade de reconstruir a imagem, sendo que isso é uma das características que torna os *containers* Docker tão leves e rápidos.

Adicionalmente, sendo o Docker uma aplicação *open-source* há disponíveis através da sua plataforma milhares de diferentes imagens, criadas por outros utilizadores. Assim, muitas das vezes não há qualquer necessidade de construir as imagens de raiz, sendo elas já disponibilizadas gratuitamente [50].

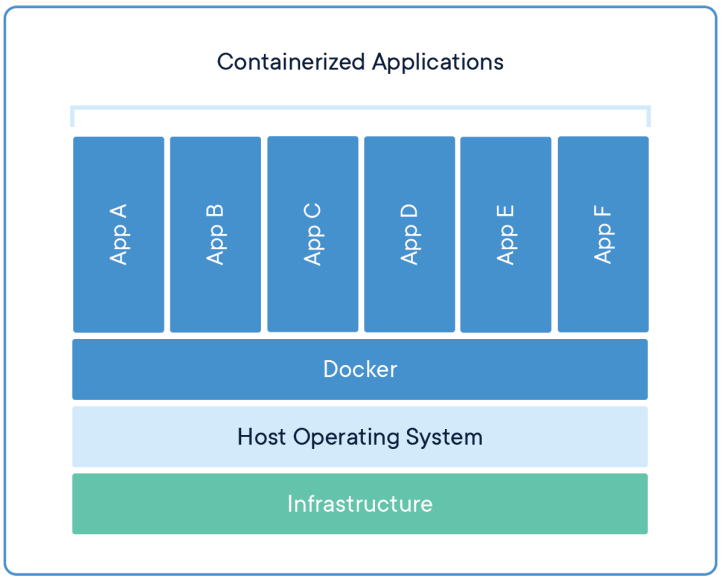


Figura 2.9: Funcionamento dos *containers* Docker (retirado de [49]).

2.3 Trabalhos Relacionados

Por fim, nesta secção são apresentados dois breves resumos relativos a duas dissertações realizadas anteriormente, no âmbito do Projeto *Augmanity*, e que apresentaram relevância para o trabalho em questão: uma relacionada mais com a parte da aquisição, recolha e organização dos dados [6], e outra com a parte do tratamento dos mesmos sob a forma de algoritmos de deteção de anomalias [7].

2.3.1 Serviços Web para Monitorização de Consumos Energéticos em Chão de Fábrica, com Base nas Plataformas Eclipse IoT: Bosch e SCoT

O objetivo desta dissertação [6] consistiu no desenvolvimento de um sistema ciber-físico capaz de fazer a recolha autónoma de dados provenientes do chão de fábrica, guardá-los e processá-los em servidores na *cloud*, e por fim ter a capacidade de os apresentar em interfaces Web para a visualização dos mesmos. Para atingir este objetivo, foi criada uma arquitetura hierarquizada como se pode constatar na Figura 2.10.

Como se pode ver na Figura 2.10, o autor propôs uma arquitetura dividida nos três níveis fundamentais já referidos na Secção 2.1.1: físico, digital e de utilizador. No nível físico constam todos os dispositivos e equipamentos do chão de fábrica, desde sensores até PLCs. Há depois *gateways* que tratam de fazer uma recolha da informação proveniente do chão de fábrica, e fazem a ponte entre a camada física e digital. Ao nível digital, os dados são injetados no Nexeed MES, plataforma desenvolvida e implementada pela Bosch. Depois, há um conjunto de *softwares* já abordados ao longo desta secção (tais como o C2E) os quais fazem a organização dos dados, e um conjunto de *scripts* em Python que permitem a transmissão de dados até ao armazenamento. Por fim, tem-se o nível do utilizador, o qual pode comunicar com o Grafana (*software* de visualização e monitorização dos dados).

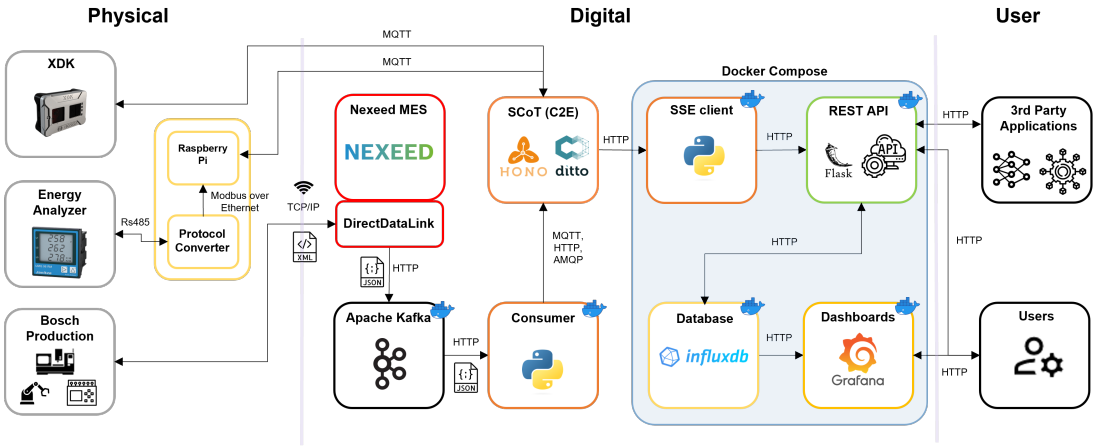


Figura 2.10: Solução genérica proposta por Matos [6].

Os resultados desta arquitetura revelaram-se bastante positivos, sendo que foi possível não só estudar a implementação a nível simulado, como também ao nível do chão de fábrica Bosch. Foi provado que a transição entre nível físico e digital era bem conseguida, analisando dados diretamente no Nexeed MES; por sua vez, todas as interações no nível digital funcionaram corretamente, desde a recolha de dados no MES até à sua introdução nas bases de dados; por fim, no que toca à interação com o utilizador, a API desenvolvida revelou-se capaz de responder com sucesso a todos os pedidos realizados pelo utilizador.

2.3.2 Agente Inteligente para Gestão de Eletricidade

No caso desta dissertação [7], o trabalho focou-se muito mais na parte do tratamento e processamento dos dados. O seu objetivo foi encontrar uma arquitetura, baseada em ML, que conseguisse apresentar previsões e detetar anomalias relativas aos consumos energéticos de um compressor situado no chão de fábrica Bosch. A arquitetura proposta é apresentada na Figura 2.11.

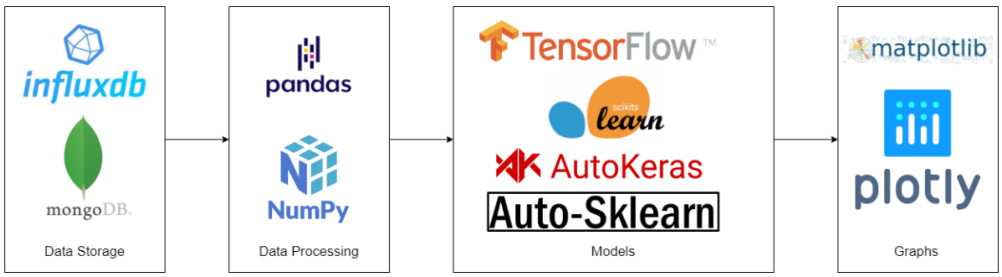


Figura 2.11: Arquitetura proposta por Oliveira [7].

Começando pela recolha de dados, à semelhança do trabalho desenvolvido por Matos [6] recorreu-se à plataforma C2E para promover uma mais fácil conectividade entre a camada física e a camada digital. A partir dos DTs, propôs-se um serviço que permitia a conexão a uma base de dados (inicialmente MongoDB, posteriormente InfluxDB), e posteriormente a conexão das bases de dados ao Grafana para se ter uma forma de monitorizar e visualizar os dados. No que toca ao agente capaz de implementar os modelos de

ML, como se pode ver pela Figura 2.11 este começa com o pré-processamento dos dados recolhidos a partir das bases de dados. Para o treino e implementação dos modelos foi usado um conjunto variado de bibliotecas, pelo que se começou pela construção de modelos base mais simples até se chegar à implementação do AutoML. O foco fundamental deste trabalho foi estabelecer uma base de comparação ao nível de diferentes modelos de ML, com a utilização de ferramentas de AutoML.

Os resultados deste trabalho foram puramente teóricos, não tendo sido possível a implementação prática da arquitetura proposta e com os modelos construidos. Foram feitas comparações em termos de tempos de execução, erros de previsões e um conjunto de indicadores utilizados para análise de desempenho dos modelos. A utilização do AutoML, apesar de revelar tempos de processamento superiores, permitiu a construção dos melhores modelos, dando indicações positivas das vantagens em termos de desempenho que a implementação do AutoML pode trazer.

2.3.3 Apreciação dos Trabalhos Anteriores

Estes dois trabalhos, realizados no âmbito do Projeto Augmanity, acabam por se complementar à luz deste projeto desenvolvido.

Por um lado, é possível retirar muita informação útil relativa à arquitetura de recolha e organização dos dados no trabalho realizado por Matos [6]. Tem-se já uma arquitetura base muito sólida com resultados práticos palpáveis, da qual foram possíveis tirar ideias para a realização deste trabalho, desde a utilização do C2E como plataforma de gestão da informação e dos DTs até à utilização de *software open-source* como o InfluxDB ou o Grafana para armazenamento e visualização dos dados.

Por outro lado, tem-se o trabalho realizado por Oliveira [7], no qual já houve um estudo bastante aprofundado ao nível da utilização de um conjunto de diferentes algoritmos para promover a deteção de anomalias nos consumos energéticos. Tem-se uma arquitetura base de recolha de dados muito semelhante à implementada por Matos [6], e posteriormente já se tem também documentados um conjunto de resultados úteis, como, por exemplo, as vantagens da utilização de AutoML, apesar do elevado custo computacional que implica.

Posto isto, o trabalho desenvolvido acaba por ser uma união destes dois trabalhos de naturezas distintas: tem-se uma arquitetura de recolha e disponibilização de dados do chão de fábrica, a qual foi construída para posterior aplicação no caso de estudo Bosch relativa a um compressor industrial com o intuito de fazer uma monitorização do equipamento. Depois, tem-se o desenvolvimento de um agente inteligente, implementado na arquitetura construída, que trata de realizar operações de previsão e deteção de anomalias, onde Oliveira [7] já realizou um importante trabalho de pesquisa. Adicionalmente, o agente inteligente fornece diferentes formas de interação com o utilizador, fruto do processamento dos dados por si conduzido.

Concluindo, o agente inteligente construído no âmbito deste projeto implementa um conjunto de modelos de ML e realiza operações de previsão e deteção de anomalias. A sua implementação enquadra-se numa arquitetura desenvolvida para conseguir recolher constantemente dados do chão de fábrica, com os resultados do processamento a serem reportados e disponibilizados para o utilizador de várias formas distintas.

Capítulo 3

Solução Concetual Proposta

Nesta secção é apresentada e descrita detalhadamente a arquitetura concetual proposta para atingir o objetivo principal do projeto, assim como cumprir com os requisitos definidos na Secção 1.3. Esta arquitetura está apresentada na Figura 3.1.

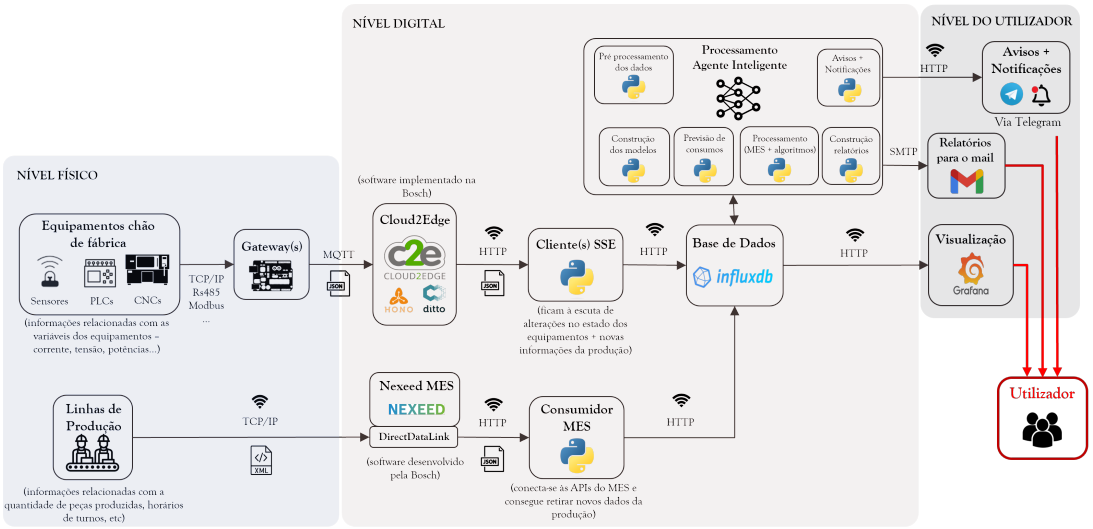


Figura 3.1: Arquitetura concetual proposta.

A arquitetura proposta está dividida em três níveis fundamentais: físico, digital e de utilizador. O nível físico é composto por todos os dispositivos e equipamentos do chão de fábrica, desde pequenos sensores e dispositivos IoT até equipamentos de maior dimensão tais como PLCs. Já o nível digital engloba um conjunto de *softwares*, muitos deles já descritos no Capítulo 2, que tratam de fazer a recolha dos dados do nível físico, sua organização e armazenamento. Também neste nível se tem um conjunto de aplicações associadas ao agente inteligente, desenvolvidas com recurso à linguagem de programação Python, que permitem o processamento dos dados. Por fim, o nível do utilizador corresponde a todas as interações que ocorrem entre o utilizador e a arquitetura, desde a parte da monitorização e visualização dos dados até às notificações que recebem via agente inteligente. Nas secções seguintes são explicadas estas três partes fundamentais da arquitetura proposta e as suas interações.

3.1 Nível Físico

Começando pelo nível físico, como já dito anteriormente, este é composto por todos os dispositivos e equipamentos que existem no interior do chão de fábrica. Estes estão dotados de sensores e detetores capazes de enviar informações e dados para a camada digital da arquitetura, na qual serão armazenados e processados. A arquitetura referente à camada física e aos fluxos de dados existentes na mesma é apresentada na Figura 3.2.

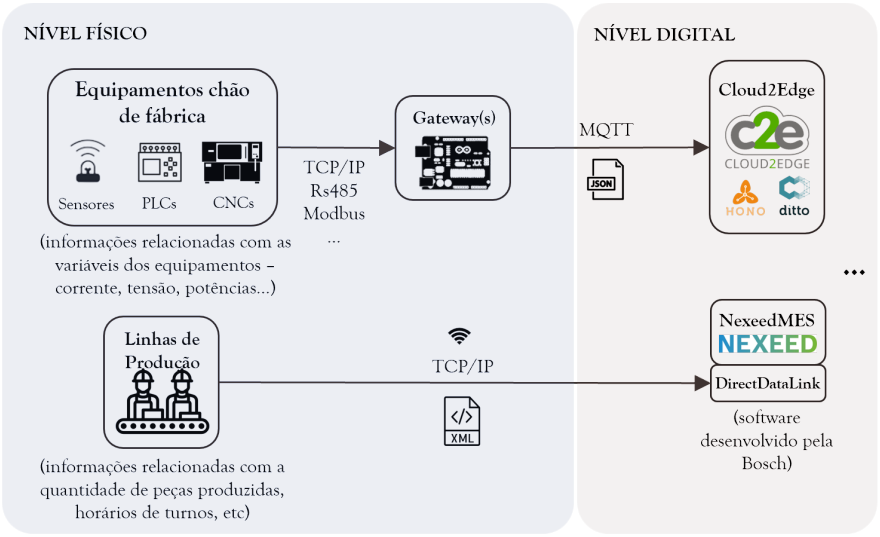


Figura 3.2: Camada física da arquitetura proposta.

Como se pode verificar na Figura 3.2, dentro deste nível há dois fluxos de dados distintos. Apesar destes dados poderem ser da mesma fonte (*i.e.*, do mesmo equipamento), constituem informações diferentes e por esse motivo entram em partes distintas do nível digital da arquitetura. Nas seguintes secções são apresentados com mais detalhe cada um destes fluxos de dados.

3.1.1 Dados dos Equipamentos

Sendo toda a arquitetura desenvolvida em torno da recolha, organização e processamento de dados provenientes dos equipamentos, torna-se clara a importância que estes possuem, constituindo a fonte destes mesmos dados.

Como se pode ver pela Figura 3.2, estes dados são gerados num conjunto muito variado de equipamentos, desde equipamentos industriais de maior porte como PLCs até pequenos dispositivos de sensorização e deteção. Há portanto um conjunto vasto de variáveis a ser monitorizadas: consumos energéticos, tensões, correntes, valores de temperatura, humidade, entre outros. Assim, existe a necessidade de transferir esses dados para a camada digital da arquitetura para posterior processamento.

Como há uma grande diversidade de equipamentos, há também uma grande variedade de protocolos de comunicação. Daí que a utilização de *gateways* seja fundamental. Os *gateways* funcionam como os dispositivos físicos intermediários que fazem a ponte entre o nível físico e o digital da arquitetura, tratando-se, por exemplo, de microcontroladores. Estes conseguem agregar a informação proveniente do chão de fábrica sob a forma dos

mais variados protocolos de comunicação (MQTT, TCP/IP, Rs485, etc.), e comunicá-la sob a forma de um único protocolo (MQTT por exemplo) para as plataformas digitais. Através do uso de *gateways*, é possível ter uma comunicação entre nível físico e digital muito mais simplificada e com total abstração face aos protocolos de comunicação existentes no chão de fábrica.

3.1.2 Dados da Produção

Se num dos lados da arquitetura se tem o envio constante de informações relativas aos equipamentos e às suas variáveis, tais como consumos de corrente ou tensão, do outro lado tem-se o fluxo de informação relacionado com a capacidade produtiva.

Estes dados dizem respeito a parâmetros como a duração dos turnos, número de peças produzidas numa dada estação ou número de peças defeituosas. Tudo isto corresponde a informação mais relacionada com os trabalhadores e com a capacidade produtiva, não constituindo propriamente variáveis inerentes aos equipamentos. Esta informação é enviada diretamente para uma aplicação desenvolvida e já implementada pela Bosch, o Nexeed MES, constituindo assim um novo fluxo de dados a ser enviado para a camada digital da arquitetura.

É de salientar que esta informação da produção possui também uma elevada importância. O processamento destes dados pode fornecer uma maior riqueza e versatilidade ao agente inteligente, uma vez que poderá levar à identificação de tendências comuns entre os comportamentos de trabalhadores numa linha de produção e os consumos energéticos dos equipamentos dessas linhas. Por exemplo, não havendo processamento dos dados do MES, um determinado pico horário de consumo de um equipamento poderia ser visto como anómalo à luz dos algoritmos. Contudo, processando também informação do MES, se durante esse período houvesse um intensificar da produção de peças nesse equipamento, este pico de consumo poderia já não representar uma anomalia, mas sim apenas um período onde a capacidade produtiva desse equipamento foi superior.

3.2 Nível Digital

Após a camada física, tem-se a camada digital. Esta corresponde à camada intermédia onde ocorrem todas as operações em *background*, desde a agregação dos dados provenientes do chão de fábrica, seu armazenamento e processamento, até que estes ficam disponíveis ao utilizador na última camada da arquitetura. Na Figura 3.3 está representado o fluxo e a organização da informação dentro da camada digital.

Como se verifica na Figura 3.3, a informação proveniente da camada física sob a forma de um fluxo duplo de dados é injetada em duas plataformas distintas, C2E e Nexeed MES, a partir das quais é agregada em *scripts* desenvolvidos em Python que consomem esses dados. O envio da informação é realizado via HTTP sob a forma de documentos em formato JSON. Os consumidores de dados, sempre que recebem novas informações, processam-nas e enviam-nas para armazenamento na base de dados temporal InfluxDB. Por fim, os dados seguem um duplo caminho para processamento via agente inteligente e visualização com recurso a interfaces gráficas construídas em Grafana.

Nas secções seguintes são apresentados mais detalhadamente os diferentes componentes presentes nesta camada digital.

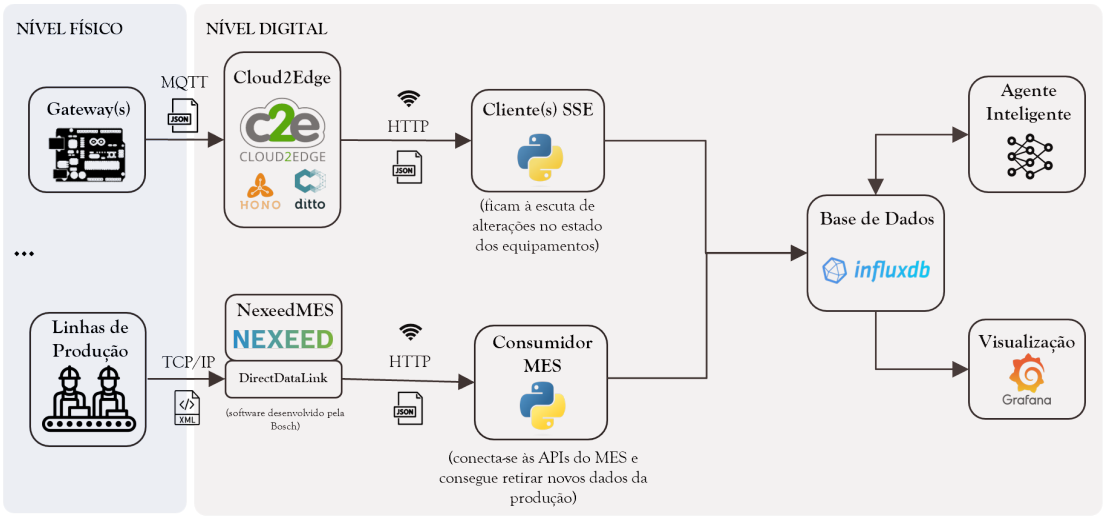


Figura 3.3: Camada digital da arquitetura proposta.

3.2.1 C2E - Cloud2Edge

No Capítulo 2, Secção 2.2.1, já foi feita uma explicação detalhada do que é esta plataforma do C2E e as vantagens que a sua implementação pode trazer. Adicionalmente, esta plataforma, já implementada na Bosch Termotecnologia em Aveiro, possui também uma implementação atualmente em funcionamento no Departamento de Engenharia Mecânica da Universidade de Aveiro, onde muito do trabalho foi de facto desenvolvido. Assim, a sua utilização tornou-se facilitada, especialmente durante a fase de testes e desenvolvimento da solução.

Como se pode ver na Figura 3.3, esta plataforma é responsável pela receção dos dados provenientes de equipamentos no chão de fábrica. Estes são primeiramente inseridos no Eclipse Hono e de seguida remetidos para o Eclipse Ditto. Começando pelo Eclipse Hono, como já explicado na Secção 2.2.1, este acaba por funcionar como o *broker* entre o nível físico e a plataforma digital do Eclipse Ditto. Cada equipamento monitorizado é inicialmente registado no Eclipse Hono. Após o registo, com a definição de um identificador para esse equipamento, o *gateway* a si associado envia a informação via MQTT para o Eclipse Hono de forma contínua.

Por sua vez, entre o Eclipse Hono e o Eclipse Ditto há uma série de possibilidades para se estabelecer a ligação e trocar informação, como os protocolos AMQP e MQTT, via Kafka, entre outros. Após se estabelecer um determinado tipo de conexão entre o Hono e o Ditto, e de se fazer também o registo do equipamento, gera-se o seu DT (mais informação na Secção 2.1.3). A partir daí, toda a informação enviada para o Eclipse Hono é automaticamente remetida para o Eclipse Ditto, e o DT do equipamento é continuamente atualizado. Com a conexão do C2E estabelecida aos *gateways*, é possível usar a nova informação que chega para processamento e eventualmente atuar no sentido inverso: alterando os valores do DT, é possível com algumas configurações modificar o estado do equipamento físico associado.

Mais informação relativa ao registo de equipamentos no Hono e no Ditto pode ser encontrada na documentação disponibilizada pela Eclipse Foundation [51] e no Apêndice B.

3.2.2 Nexeed MES

Paralelamente ao fluxo de dados dos equipamentos, como referido na Secção 3.1, tem-se o fluxo de dados da produção que é remetido para o Nexeed MES. O Nexeed MES, como o próprio nome indica, constitui um *Manufacturing Execution System*. De uma forma muito resumida, um MES corresponde a um sistema capaz de conduzir a monitorização, registo e o próprio controlo de todos os processos industriais que ocorrem ao nível do chão de fábrica desde que a matéria-prima chega até que se tem, à saída, os produtos finalizados. Sendo assim, os dados que se conseguem obter ao nível do MES são de naturezas completamente distintas face aos dados injetados pelos equipamentos. Dependendo dos postos e das linhas de trabalho conseguem-se obter informações variadas, desde a utilização e consumos de materiais, capacidades de produção de equipamentos e linhas, até ao próprio controlo de qualidade.

Adquirindo toda esta informação, e apesar das naturezas muito divergentes face à informação recolhida no C2E, é possível juntá-las e estabelecer relações, por exemplo, entre o número de peças produzidas por dado equipamento e o consumo energético que ele teve ao longo de um turno. Isso pode fornecer um maior valor aos algoritmos construídos para promover a deteção de anomalias ao nível dos consumos elétricos das máquinas.

3.2.3 Agregação dos Dados

Como se pode ver na Figura 3.3, após a recolha e organização dos dados em duas plataformas digitais, a sua agregação é realizada através de dois blocos correspondentes a *scripts* Python.

Começando pela informação proveniente dos equipamentos. Após a conexão dos equipamentos físicos ao C2E e criação das suas réplicas digitais, estas conseguem enviar informação de forma contínua para a plataforma. Com recurso a um cliente SSE (*Server Sent Events*) é possível detetar quando nova informação chega. O SSE consiste numa tecnologia que permite conectar um servidor a um cliente, sendo que o esquema de funcionamento de uma conexão SSE encontra-se demonstrado na Figura 3.4.

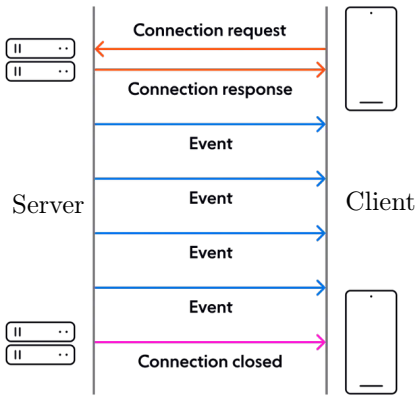


Figura 3.4: Diagrama de conexões entre servidor e cliente com recurso a SSE (adaptado de [52]).

Como se pode ver na Figura 3.4, após se estabelecer uma ligação HTTP entre cliente e servidor, sempre que o servidor recebe novos dados, automaticamente gera um evento no qual envia essa informação para o cliente. No caso da arquitetura proposta esta informação é enviada sob a forma de documentos JSON para um cliente desenvolvido em Python. Esse cliente, por sua vez, filtra a informação relevante e injeta-a na base de dados.

Os dados provenientes da produção possuem também um destino semelhante, mas não com um cliente SSE. Enquanto o Ditto possui apenas uma API, o Nexeed MES tem várias, cada uma para um módulo distinto (tem-se, por exemplo, uma API dedicada somente ao controlo dos turnos, outra dedicada à monitorização das peças produzidas, entre outras). Contudo, nenhuma dessas APIs consegue gerar eventos SSE. Para o consumidor em Python receber a informação do MES, deve ser ele a enviar pedidos às APIs. As respostas chegam-lhe sob a forma de documentos JSON e de forma semelhante ao que acontece com o cliente SSE, a informação mais relevante é filtrada e enviada para a base de dados.

3.2.4 Armazenamento dos Dados

Após ser feita a recolha dos dados, primeiro passando pelas plataformas C2E e Nexeed MES, tem-se o armazenamento desses dados. Na Secção 2.1.2 já foi feita uma introdução relativa às bases de dados, quais os dois principais tipos e as suas vantagens e desvantagens.

As bases de dados relacionais, apesar de serem capazes de lidar até certo ponto com dados temporais, possuem fraca escalabilidade e algumas dificuldades em lidar com grandes quantidades de informação, o que num contexto de Indústria 4.0 e de *Big Data* acaba por ser inevitável. Por outro lado, as bases de dados não relacionais oferecem já uma grande escalabilidade e capacidade de gestão de grandes fluxos de dados.

Dentro das bases de dados não relacionais, para lidar com grandes quantidades de dados de diferentes formatos em tempo real as bases de dados temporais revelam-se as mais indicadas [41], [42]. Daí que na arquitetura conceptualizada se tenha proposto uma base de dados desse tipo, o InfluxDB, a qual permite uma fácil interação com aplicações externas tais como o *software* de visualização escolhido, o Grafana. Adicionalmente, esta base de dados tem inúmeras implementações em casos de estudo na Bosch Termotecnologia Aveiro, facilitando também a sua integração nesta arquitetura.

3.2.5 Processamento dos Dados

Este é o bloco da arquitetura onde o agente inteligente possui grande relevância. A sua implementação envolve um conjunto variado de funções. As principais dimensões associadas ao agente inteligente são as seguintes:

- **Recolha dos dados** - para ser possível a construção dos algoritmos, são necessários um conjunto de dados reais, devidamente pré-processados. Esses permitirão o processo de treino dos modelos. Assim, é necessário que seja realizado este trabalho de recolha e tratamento de dados;
- **Construção dos modelos** - a partir dos dados recolhidos, é feita a construção dos modelos. Esta engloba a implementação dos algoritmos de ML, mais especificamente *autoencoders*, em conjunto com uma ferramenta de otimização, o AutoML;

- **Implementação dos modelos** - após finalizado o processo de construção, os modelos estão prontos a ser implementados. A partir de um dado número de valores prévios, são capazes de fazer uma previsão futura para diversas variáveis. Esta implementação exige uma constante comunicação com a base de dados, para poderem ser recolhidos os dados anteriores e enviadas as previsões futuras;
- **Deteção de anomalias** - o processo de deteção de anomalias promove uma comparação entre previsões e respetivo valor real verificado. Caso o valor previsto seja muito díspar relativamente ao respetivo valor real, o ponto é visto como uma anomalia. Dependendo depois do comportamento de cada variável, a diferença a partir da qual se tem uma anomalia é diferente;
- **Geração de relatórios** - uma mais-valia associada a este agente inteligente é também a geração de relatórios, quer sejam relatórios horários, diários ou por turnos. Nestes relatórios podem constar as mais variadas informações, desde as anomalias detetadas ao longo do último turno, consumos médios dos equipamentos, gráficos do comportamento das variáveis e também informações relativamente ao Nexeed MES. Estes relatórios, automaticamente enviados para os trabalhadores, engenheiros de produção, etc., podem dar *inputs* importantes relativamente a oportunidades de melhoria ao nível do comportamento dos equipamentos e da produção;
- **Envio de notificações** - enquanto os relatórios estão mais relacionados com informações enviadas periodicamente, também se pode implementar esta dimensão das notificações do agente inteligente para uma monitorização mais constante. Ao se detetar uma anomalia num equipamento ou na produção, o agente é responsável por enviar de forma automatizada uma notificação (via uma aplicação como o Telegram) para um responsável de manutenção, para que este rapidamente possa atuar sobre esse equipamento e normalizar a situação o mais rápido possível.

3.2.6 Visualização dos Dados

Por fim, ao nível da camada digital resta referir a plataforma de visualização dos dados, neste caso o Grafana. Esta interface gráfica está conectada à base de dados, e recebe constantemente dados relativos ao estado dos equipamentos, ao estado da produção e muitas outras informações resultantes do processamento do agente inteligente.

Com a interface gráfica do Grafana, um qualquer utilizador pode ter acesso a um conjunto de gráficos e informações em tempo real relativamente ao que está de facto a acontecer no chão de fábrica. De entre as várias possibilidades de visualização, tem-se:

- **Construção de gráficos** - há a possibilidade de fazer a visualização dos dados sob um conjunto muito variado de gráficos e tabelas. Podem-se construir gráficos de séries temporais onde há uma comparação gráfica entre os valores reais e previstos, tabelas onde se vê, para cada registo temporal, o valor real, previsão associada e erro da previsão, entre outros;
- **Componente alarmística** - também no Grafana há a possibilidade de integrar a alarmística. Paralelamente às notificações que o agente inteligente pode enviar, também através da interface do Grafana é possível definir alarmes, que podem estar associados exatamente às tais anomalias (*i.e.*, valores muito díspares entre previsões e realidade) ou então a possíveis anomalias ao nível da produção, tais

como a paragem súbita de produção de peças numa determinada estação ou um número elevado de peças defeituosas.

Na Figura 3.5 é apresentado um exemplo da interface gráfica desenvolvida para o projeto. Nela podem-se verificar as possibilidades de visualização mencionadas acima.

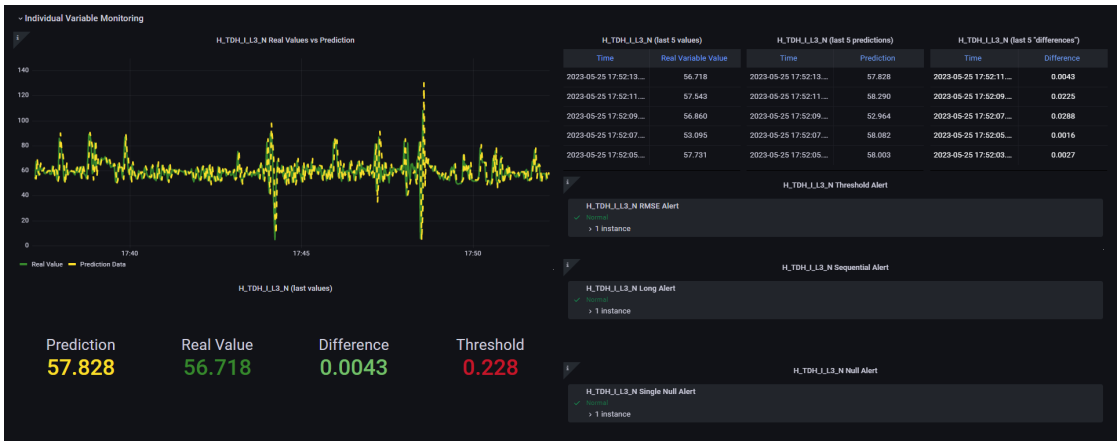


Figura 3.5: Exemplo da interface gráfica desenvolvida e implementada.

3.3 Nível do Utilizador

A última camada da arquitetura corresponde ao nível do utilizador. Este é o nível onde o utilizador irá de facto interagir com toda a arquitetura, segundo várias formas distintas, definidas ao longo da camada intermédia digital que faz a conexão desde o chão de fábrica até este nível superior. Na Figura 3.6 estão apresentadas as várias interações.

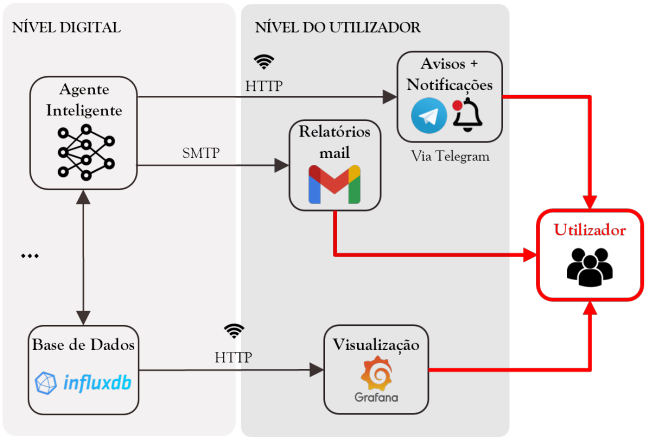


Figura 3.6: Interações entre a camada digital e do utilizador.

Como já foi sendo exposto ao longo da Secção 3.2, e como se pode ver na Figura 3.6, há um conjunto de interações de diferentes naturezas do utilizador com a restante arquitetura:

- **Visualização gráfica dos dados** - através da interface Grafana, o utilizador tem a capacidade de monitorizar tudo o que se passa ao nível do chão de fábrica. Pode identificar tendências, ter acesso a um conjunto de alarmes e também visualizar o estado atual dos equipamentos e da produção, tudo isso através de uma interface rápida de visualização de dados;
- **Receção de notificações** - o agente inteligente, como já referido anteriormente, tem a capacidade de gerar notificações via Telegram para o utilizador. Essas notificações permitem ao utilizador rapidamente identificar o problema e onde deverá atuar ao nível dos equipamentos no chão de fábrica;
- **Receção de relatórios** - neste caso de forma mais periódica. O agente inteligente constrói relatórios que envia por email para o utilizador. Estes contêm toda a informação relativa aos acontecimentos mais relevantes da última hora ou turno. Estes relatórios constituem também um registo histórico, e podem ser posteriormente utilizados para um novo processamento de dados, com análise, por exemplo, de tendências que ocorram durante certos períodos de certos turnos.

3.4 Visão Geral

Como se pode ver, a arquitetura proposta possui já um nível de complexidade relativamente elevado. Por um lado tem-se toda uma arquitetura que recolhe os dados do chão de fábrica e os disponibiliza segundo plataformas digitais. E depois tem-se o agente inteligente, que está dotado de um conjunto de diferentes funcionalidades que deverão funcionar em harmonia.

Num primeiro nível associado ao chão de fábrica há uma recolha de dados de duas fontes distintas: uma ligada aos equipamentos propriamente ditos, com dados relativos aos seus consumos energéticos, e outra associada a dados da própria produção e relacionados com a capacidade produtiva das linhas.

Passando para a camada digital, esta arquitetura faz uso de dois *softwares* já implementados na Bosch Termotecnologia Aveiro, o C2E e o Nexeed MES, estando portanto adaptada às necessidades atuais da empresa. Após injeção dos dados nessas plataformas os dados são alvo de um pré-processamento, com filtragem da informação mais importante, armazenados numa base de dados para o efeito, e finalmente processados pelo agente inteligente. Este agente possui um conjunto de "tarefas" que deve desempenhar, desde operações de previsão e deteção de anomalias, geração contínua de novos modelos adaptados às tendências mais recentes das variáveis até à geração dos relatórios e notificações. Adicionalmente, todas estas operações devem funcionar em harmonia e em tempo real, para promover um processamento o mais eficaz e rápido possível.

Por fim, tem-se a camada associada ao utilizador. Esta representa todas as formas que o utilizador tem de interagir com a arquitetura. Com as interfaces gráficas o utilizador pode facilmente monitorizar os eventos que ocorrem no chão de fábrica. A receção de emails podem-lhe fornecer resumos de como correu determinado turno. Por fim, a receção de notificações via agente inteligente poderá permitir uma mais rápida intervenção caso algo corra menos bem.

No capítulo seguinte são demonstrados, de uma forma detalhada, todos os passos da implementação da arquitetura proposta sob a forma de diferentes blocos, os quais foram sucessivamente aumentando de complexidade.

Intentionally blank page.

Capítulo 4

Implementação da Solução

Este capítulo é dedicado à exposição de como foi implementada a arquitetura proposta no Capítulo 3. Esta implementação foi realizada por blocos, que seguiram a estrutura e a própria sequência lógica dos dados ao longo da arquitetura. Todos estes blocos foram sendo validados a nível simulado, com o trabalho a ser desenvolvido no Departamento de Engenharia Mecânica da Universidade de Aveiro. Todo este trabalho culminou por fim na implementação da arquitetura a um caso de estudo prático, na Bosch Termotecnologia Aveiro.

É de salientar que, devido a certas dificuldades que serão mencionadas ao longo desta secção, não foi possível fazer a implementação do Nexeed MES na arquitetura proposta. Sendo assim, nesta secção não é abordada essa parte.

4.1 Interface entre os Dispositivos e a Plataforma C2E

Numa primeira fase da implementação foi necessário trabalhar na passagem da informação do nível físico para o digital, através da plataforma C2E. Assim, desenvolveu-se numa primeira instância um *script* em Python que simulava os dados provenientes de um dispositivo IoT. Posteriormente, usaram-se dois microcontroladores ESP32 para simular o envio de dados já provenientes de dispositivos físicos para a plataforma digital.

4.1.1 Utilização de um Simulador de Dados

A primeira parte da arquitetura implementada, a um nível totalmente digital e simulado, encontra-se na Figura 4.1.

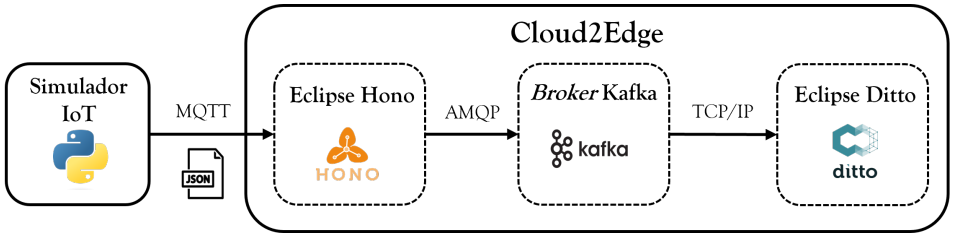


Figura 4.1: Esquema do fluxo de dados entre dispositivos (neste caso simulados) e a plataforma C2E.

Como se pode ver na Figura 4.1, criou-se um *script* em Python para fazer a simulação da injeção de dados no C2E. Para que essa injeção de dados pudesse ocorrer houve a necessidade inicial de fazer o registo de um dispositivo de testes na plataforma (a forma como esse registo deve ser efetuado é apresentada detalhadamente no Apêndice B). Neste registo houve a definição do nome a atribuir ao dispositivo, assim como de algumas credenciais que permitiram o envio da informação via MQTT para o C2E.

Após o registo do dispositivo na plataforma digital e criação do seu DT foi possível simular a injeção de dados através do *script* construído. Este *script* permitia estabelecer uma conexão MQTT com o Hono (usando as devidas credenciais criadas para o dispositivo); a partir do momento em que a conexão era estabelecida, simulavam-se valores aleatórios de temperatura e humidade, que eram periodicamente injetados nessa plataforma. Para que essa informação pudesse chegar devidamente ao Hono, era necessário gerar uma mensagem MQTT com um tópico devidamente formatado, e um corpo sob a forma de um JSON com uma sintaxe muito específica, como se pode ver no Código 4.1.

```
Topico_MQTT = "telemetry/<tenant>/<namespace>:<device_name>";
JSON = {
    "topic": "<namespace>/<device_name>/things/twin/commands/modify",
    "headers": {},
    "path": "/features",
    "value": {
        "temperature": {
            "properties": {
                "value": 15}
            }
        "humidity": {
            "properties": {
                "value": 84}
            }
        }
    }
```

Código 4.1: Tópico MQTT e JSON a enviar para o Eclipse Hono.

Olhando para o Código 4.1, há alguns pontos a salientar:

- Todos os parâmetros que se encontram entre "<>" correspondem a parâmetros definidos aquando o registo do dispositivo no C2E, correspondendo essencialmente às suas credenciais e identificadores (mais informação sobre estes parâmetros no Apêndice B);
- Existem dois tópicos distintos: tem-se o tópico MQTT onde a nova informação é publicada e tem-se o tópico da mensagem JSON. Este último define um comando que o Hono remete para o Ditto, indicando-lhe que está a enviar novos dados para atualização do DT. Alterando o tópico do JSON é possível definir outros pedidos ao Ditto. No Código 4.2 apresenta-se o JSON a enviar para consulta do estado do DT.

```
JSON = {
    "topic": "<namespace>/<device_name>/things/twin/commands/retrieve",
    "headers": {},
    "path": "/attributes",
}
```

Código 4.2: JSON a enviar para se analisar o estado atual do DT do dispositivo.

Analisando a restante arquitetura apresentada na Figura 4.1, após a receção dos dados via MQTT do simulador, o Eclipse Hono remetia-os para um *broker* Kafka. Por fim, com o Eclipse Ditto a subscrever aos tópicos do Kafka, sempre que havia alguma alteração no estado das variáveis (ou seja, novos valores simulados a ser injetados), automaticamente o DT associado era atualizado. É de salientar que, como referido na Secção 3.2.1, a conexão entre o Hono e Ditto pode ser configurada de diversas formas, desde MQTT, AMQP até à utilização de um *broker* Kafka. Utilizou-se o Kafka pelo facto de, nas versões mais recentes do C2E, esta ser a conexão estipulada por defeito.

4.1.2 Utilização de Dispositivos Físicos

Após a validação do funcionamento desta parte da arquitetura a um nível totalmente simulado e digital, substituiu-se o simulador em Python pela utilização de dispositivos físicos reais, dois microcontroladores ESP32. A Figura 4.2 representa um diagrama de todos os passos conduzidos para colocar em funcionamento esta parte da arquitetura.

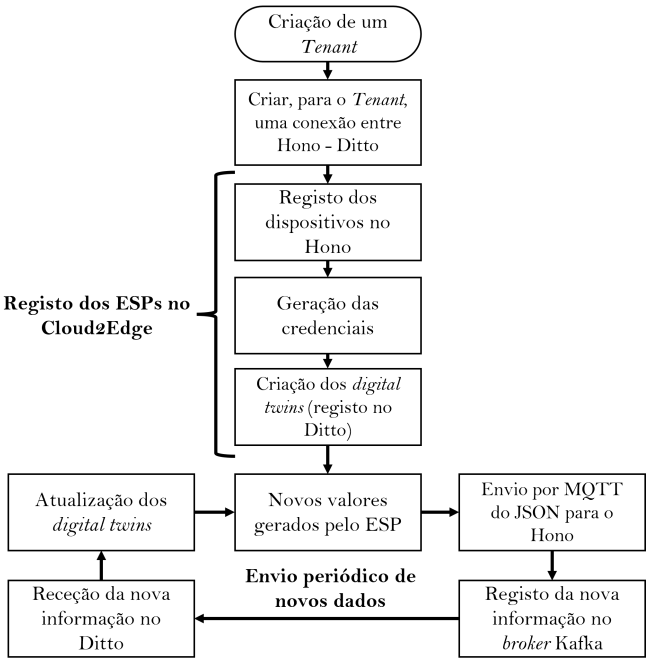


Figura 4.2: Fluxograma com os passos necessários para registar dispositivos e enviar informação periodicamente para o C2E.

Começando pelos ESPs, estes foram programados com recurso ao ambiente de desenvolvimento Arduino IDE [53]. Neles foram inseridos um conjunto de dados reais relativos ao caso de estudo do compressor da Bosch Termotecnologia Aveiro onde Oliveira [7] trabalhou previamente, e onde este trabalho foi implementado. Estes dados correspondiam a leituras reais relativas a algumas variáveis (potência elétrica, tensão, etc.) medidas no compressor do chão de fábrica Bosch. Posto isto, o código inserido nos ESPs fez o mesmo que o código do simulador previamente implementado: cada ESP conectava-se inicialmente via MQTT ao Eclipse Hono, e após a conexão estar devidamente estabelecida enviava periodicamente novos valores para o C2E.

Para que este envio pudesse ser efetuado, como já discriminado na Secção 4.1.1, primeiro foi necessário fazer um registo dos dispositivos na plataforma digital. Como se pode ver na Figura 4.2, este registo segue uma sequência lógica: caso não haja um *tenant* criado, deve-se proceder à criação do mesmo (um *tenant* corresponde a uma entidade que reúne um conjunto de dispositivos). De seguida, deve ser estabelecida, para esse *tenant*, uma ligação Hono - Ditto. Só após isto é que é possível registar os dispositivos: primeiro na plataforma do Hono, com a geração das suas credenciais (importantes para estabelecer a conexão MQTT com o Hono) e depois no Ditto, com a criação das réplicas digitais de cada dispositivo.

Após todo este processo, os ESPs foram capazes de enviar informações para o Hono através de documentos JSON com o formato já especificado anteriormente. Essas informações eram remetidas para o Ditto, e os DTs de cada ESP atualizados. Acedendo ao URL "`http://<DITTO_API_IP>:<DITTO_API_PORT_HTTP>/api/2/things/<namespace>:<device_name>`" num computador conectado à mesma rede onde o C2E corre é possível consultar o estado do DT de um dado equipamento. A Figura 4.3 serve de exemplo para mostrar o estado do DT do dispositivo com o nome "ESP_32", *namespace* "av101" (mais informação pode ser encontrada na documentação oficial do Eclipse Ditto [51]).

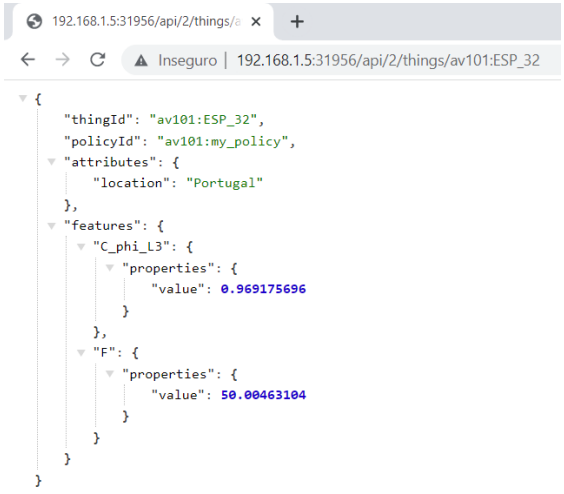


Figura 4.3: Exemplo da consulta do estado de um DT de um dos ESPs, neste caso com o *namespace* "av101" e o nome "ESP_32".

4.2 Base de Dados e Plataforma de Visualização

Após uma primeira implementação onde já se conseguiu realizar a ponte entre o nível físico (sob a forma de ESPs) e o nível digital da arquitetura (representado pelo C2E), aumentou-se a sua complexidade. Acrescentou-se, após o C2E, a capacidade de visualização e armazenamento dos dados com recurso ao Grafana e ao InfluxDB, respetivamente. Na Figura 4.4 é demonstrado o esquema representativo dessa arquitetura.

Na implementação prévia já se tinha a conexão físico-digital, com os ESPs a representarem dispositivos físicos que injetavam periodicamente dados na plataforma digital do C2E. Para esta implementação, introduziu-se na arquitetura um cliente SSE (*Server*

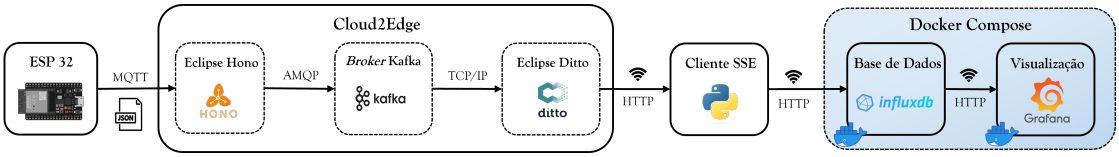


Figura 4.4: Arquitetura representativa da implementação do armazenamento e visualização dos dados.

Sent Events) desenvolvido em Python que veio permitir a inclusão das capacidades de armazenamento e visualização na arquitetura. O cliente SSE estabelecia uma conexão HTTP com o servidor do Ditto, e sempre que novos dados chegavam ao Ditto, eram automaticamente remetidos para o cliente. No mesmo *script* em Python foi feita uma conexão à base de dados, para a qual as novas informações eram posteriormente remetidas para armazenamento. Finalmente, tendo o Grafana conectado à base de dados, os dados puderam ser visualizados sob a forma de gráficos e tabelas interativas. É de salientar que para o teste desta parte da arquitetura (e todos os outros testes que ocorreram no Departamento de Engenharia Mecânica da Universidade de Aveiro), o InfluxDB e o Grafana estavam contidos num *Docker-Compose*. Isso veio facilitar a implementação, não sendo necessária a instalação destes *softwares*.

4.2.1 Cliente SSE

Como exposto acima, desenvolveu-se um cliente SSE em Python com o objetivo de se conectar à API HTTP do Eclipse Ditto. Após o estabelecimento de uma conexão HTTP entre as duas entidades, sempre que havia uma alteração ao nível do estado do DT, a API notificava o cliente das alterações que haviam ocorrido num dado dispositivo. Assim, sempre que ocorresse uma alteração, era gerado um novo evento e enviado para o cliente SSE um JSON com o formato especificado no Código 4.3.

```
{
  "thingId": "av101:ESP_32",
  "features": {
        "C_phi_L3": {
            "properties": {
                "value": 0.93885463
                }
            }
        "F": {
            "properties": {
                "value": 49.9355648
                }
            }
        }
}
```

Código 4.3: JSON recebido pelo cliente SSE.

Neste caso, houve a alteração no estado das variáveis "C_phi_L3" e "F", relativas ao dispositivo registado no Ditto com o nome "ESP_32" com o *namespace* "av101". Para cada dispositivo registado no C2E, sempre que era atualizado o estado do DT, um JSON com exatamente este formato era gerado e enviado via HTTP para o cliente SSE. No *script* Python do cliente havia um processamento e filtragem da informação, que por sua

vez era enviada para a base de dados. Para isso, também nesse *script* houve a criação de um cliente InfluxDB.

4.2.2 Armazenamento - InfluxDB

Como já exposto na Secção 3.2.4 a base de dados escolhida foi o InfluxDB, uma base de dados temporal capaz de lidar com uma grande quantidade de dados em tempo real. Adicionalmente, na Secção 2.2.3 foi feita uma introdução ao InfluxDB e à forma da organização dos dados.

Como referido nessa secção, o InfluxDB está organizado em *buckets*, dentro dos quais a informação é armazenada. Cada vez que se pretende injetar novos dados são enviados *points*, os quais possuem *measurements*, *tags*, *fields* e *timestamps*. A Figura 4.5 expõe a organização do *bucket* InfluxDB usado nesta fase da implementação.

_time	_measurement	_value	DataType	Equipment
2023-04-20 14:33:47 GMT+1	U_L1_N	239,28	Real Data	ESP_32_2
2023-04-20 14:33:52 GMT+1	U_L1_N	239,15	Real Data	ESP_32_2
2023-04-20 14:33:57 GMT+1	U_L1_N	238,59	Real Data	ESP_32_2
2023-04-20 14:34:02 GMT+1	U_L1_N	238,58	Real Data	ESP_32_2

Figura 4.5: *Screenshot* da organização geral do *bucket* InfluxDB.

Olhando para cada uma das colunas, tem-se:

- "_time" - guarda o instante temporal onde chegou toda a informação;
- "_measurement" - representa as variáveis que eram injetadas a partir dos ESPs (neste caso tem-se a variável "U_L1_N");
- "_value" - guarda o valor da medição;
- "DataType" - corresponde a uma das *tags* criadas manualmente. Neste caso, apenas serve para dizer que a medição é um valor real. Nas secções seguintes, após a implementação do agente inteligente, foi usada como base esta estrutura e foram criados novos parâmetros associados ao "DataType", relativos a previsões e erros de previsão;
- "Equipment" - permite identificar qual o dispositivo que enviou a informação relativa à medição. No caso desta implementação, tem-se dois ESPs registados, pelo que nesta coluna aparecia ou "ESP_32" ou "ESP_32_2".

Por fim, no Código 4.4 é apresentada a *query* InfluxDB enviada para injetar novos dados na base de dados.

```
msg_to_send = influxdb_client.Point(measurement) \
    .tag("Equipment", equipment) \
    .tag("DataType", "Real Data") \
    .field("value", value) \
    .time(datetime.utcnow(), influxdb_client.WritePrecision.NS)
```

Código 4.4: *Query* InfluxDB a enviar.

4.2.3 Visualização - Grafana

Para promover a monitorização dos dados e do fluxo de informação ao longo da arquitetura utilizou-se o Grafana, uma plataforma *open-source* que possibilita a construção de interfaces gráficas interativas, realizar algum processamento de dados e ainda implementar a componente alarmística.

A sua conexão à base de dados é feita de forma bastante simples: o Grafana possui um conjunto de diferentes fontes de dados, de entre elas o InfluxDB. É apenas necessário configurar no Grafana o InfluxDB como sendo a sua fonte de dados, indicando alguns parâmetros de autenticação e definindo o *bucket* a monitorizar. Na Figura 4.6 é apresentada a *dashboard* criada para monitorizar os dados injetados pelos dois ESPs.

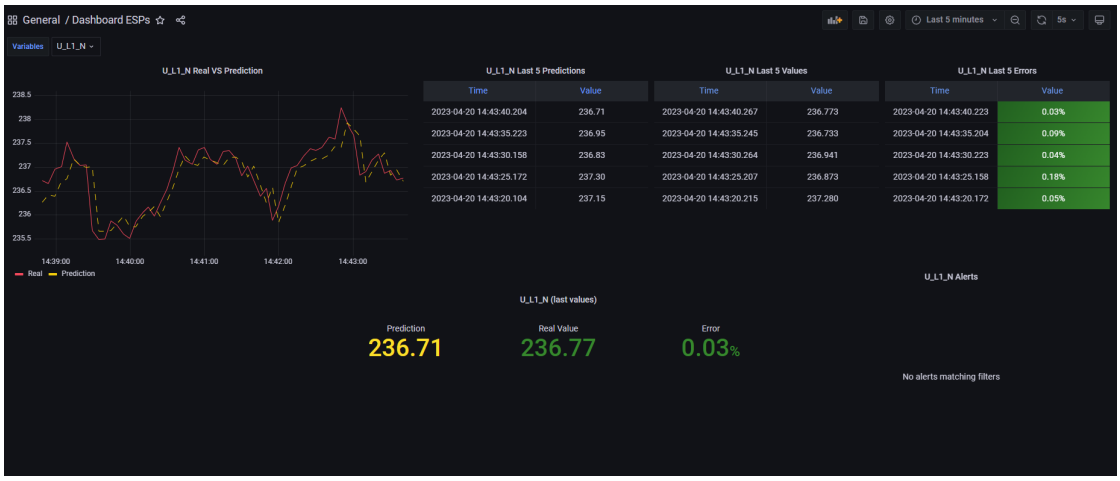


Figura 4.6: *Dashboard* desenvolvida em Grafana para monitorização dos dados enviados pelos ESPs.

Como se pode ver na Figura 4.6, a *dashboard* criada permite fazer uma monitorização gráfica do histórico de valores de cada variável a ser injetada pelos ESPs. Adicionalmente, criou-se uma tabela com o propósito de permitir uma visualização mais rápida dos últimos valores de cada variável. Posteriormente, após a implementação do agente inteligente, complementou-se esta interface gráfica com mais algumas funcionalidades, nomeadamente a visualização dos erros da previsão, gráficos com a comparação entre valores reais e previstos, e ainda alarmes quando os erros eram excessivamente elevados.

4.3 *Intelligent Assistant*

Após o desenvolvimento de grande parte da arquitetura de recolha e organização dos dados, o foco do trabalho concentrou-se no desenvolvimento do agente inteligente. Este envolveu um conjunto muito grande de funcionalidades e necessidades, desde a construção dos modelos de ML, a sua aplicação de modo a realizar previsões e posterior deteção de anomalias, tudo isto relacionado com o processamento dos dados. Adicionalmente, também envolveu um conjunto variado de diferentes interações com o utilizador.

Tal como aconteceu na restante parte da arquitetura, à exceção do código dos ESPs desenvolvido em Arduino IDE, para o tratamento dos dados e desenvolvimento dos modelos de ML o código foi todo desenvolvido em linguagem Python. Houve portanto a

necessidade de recorrer a um conjunto de diferentes bibliotecas. Para o tratamento dos dados, que engloba várias tarefas, desde geração de vetores e matrizes, cálculo de certos parâmetros, leitura de ficheiros, entre outras, foram usadas as bibliotecas Numpy 1.22.4 e Pandas 1.3.5. A análise gráfica dos resultados, nomeadamente numa primeira fase de testes, foi conseguida recorrendo à biblioteca Matplotlib 3.5.1. Por fim, a construção dos modelos e a utilização do AutoML implicou a conjugação de várias bibliotecas, sendo de destacar o Tensorflow 2.10.0, biblioteca dedicada à construção de redes neuronais, o scikit-learn 1.2.1, mais usado para o cálculo de métricas e pós-processamento dos resultados dos modelos, e por fim a biblioteca KerasTuner 1.3.0 para promover a implementação do AutoML.

As secções seguintes abordam com mais detalhe todos os processos presentes no desenvolvimento do agente inteligente, desde a escolha inicial do algoritmo até à forma como estes realizam previsões e detetam anomalias, e ainda as suas formas de interação com o utilizador.

4.3.1 Escolha do Algoritmo - *Autoencoders*

Na Secção 2.1.4 foi abordada a questão da deteção de anomalias, com uma exposição detalhada ao nível dos tipos de anomalias que podem ocorrer e quais os principais algoritmos usados para as detetar. Olhando para os diferentes algoritmos, suas vantagens e desvantagens, optou-se pela utilização dos *autoencoders*. De uma forma muito breve, os *autoencoders* correspondem a uma derivação das redes neuronais, sendo que a sua arquitetura base é apresentada na Figura 4.7.

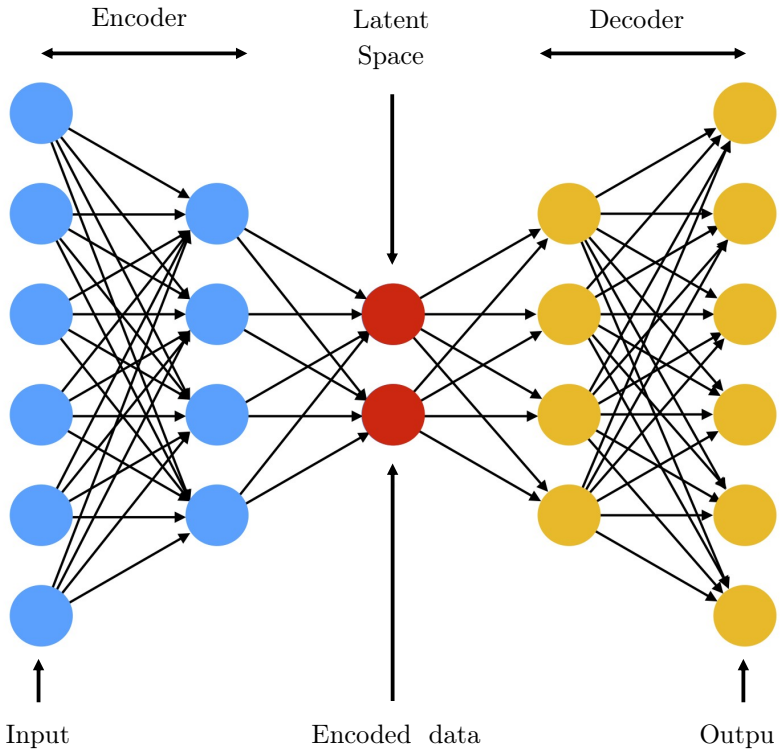


Figura 4.7: Arquitetura base de um *autoencoder* (adaptado de [54]).

São compostos por três elementos fundamentais - *encoder*, *decoder* e espaço latente. Tendo em conta a dimensão dos dados de entrada, o objetivo do *encoder* é fazer uma extração das características mais importantes desses dados, de modo a obter uma representação latente com o menor número de dimensões possível. Posteriormente, o *decoder* trata de fazer a reconstrução desses dados até ao estado inicial. O *autoencoder* é tão melhor quanto mais próximos os dados de entrada são dos dados de saída (mais informações relativas aos *autoencoders* podem ser encontradas no Apêndice A, Secção A.7).

Para a aplicação em causa, onde se combina o processo de previsão de variáveis em séries temporais com a tarefa de deteção de anomalias, a implementação dos *autoencoders* possui algumas vantagens:

- Mesmo havendo a possibilidade de se ter certas variáveis a prever dependentes de outras, o *autoencoder* apenas necessita de ter como *input* a variável a prever. Assim, não há a necessidade de analisar o comportamento de um conjunto de variáveis para prever apenas uma;
- Manipulando a estrutura e dimensão das camadas, facilmente se pode definir a quantidade de pontos temporais a prever e a quantidade de pontos temporais prévios necessários, havendo bastante flexibilidade nesse sentido;
- Apesar dos *autoencoders* serem usados para tarefas de regressão (*i.e.*, previsão de séries temporais), facilmente é possível aplicá-los a tarefas de classificação, como é o caso da deteção de anomalias: olhando para o comportamento dos dados, basta definir um limite (manualmente ou através de um pequeno processamento dos dados) para a anomalia. Qualquer ponto que se afaste mais do que o definido no limite será considerado uma anomalia.

Contudo, é importante salientar que a implementação deste algoritmo exige algumas precauções. Uma vez que o funcionamento do algoritmo implica fazer uma reconstrução do comportamento normal da variável de entrada para "aprender" o seu comportamento, é importante garantir que os dados alimentados para o treino do *autoencoder* estão isentos de anomalias. Caso contrário, o treino do *autoencoder* com dados anómalos pode levar a que o modelo codifique como comportamento normal a existência de anomalias. Por consequência, o modelo perde eficácia ao nível da sua deteção.

4.3.2 Treino e Construção dos Modelos

Como já mencionado na Secção 3.2.5, o processo de construção dos modelos foi realizado com o auxílio do AutoML (*Automated Machine Learning*). Na Figura 4.8 é apresentado um fluxograma com os passos a seguir para a construção de um modelo de previsão de uma determinada variável.

O processo de construção dos modelos dividiu-se num conjunto de fases essenciais. Primeiro foi necessário definir uma arquitetura base do *autoencoder*, a partir da qual o AutoML iria proceder à construção dos modelos. Definindo os hiperparâmetros de iteração do AutoML, e recorrendo a um pré-processamento dos dados de treino, foi possível fazer a implementação do AutoML segundo um processo iterativo de otimização dos hiperparâmetros para fazer a construção dos modelos. De seguida é dada uma explicação mais detalhada relativa a cada passo fundamental deste processo de construção.

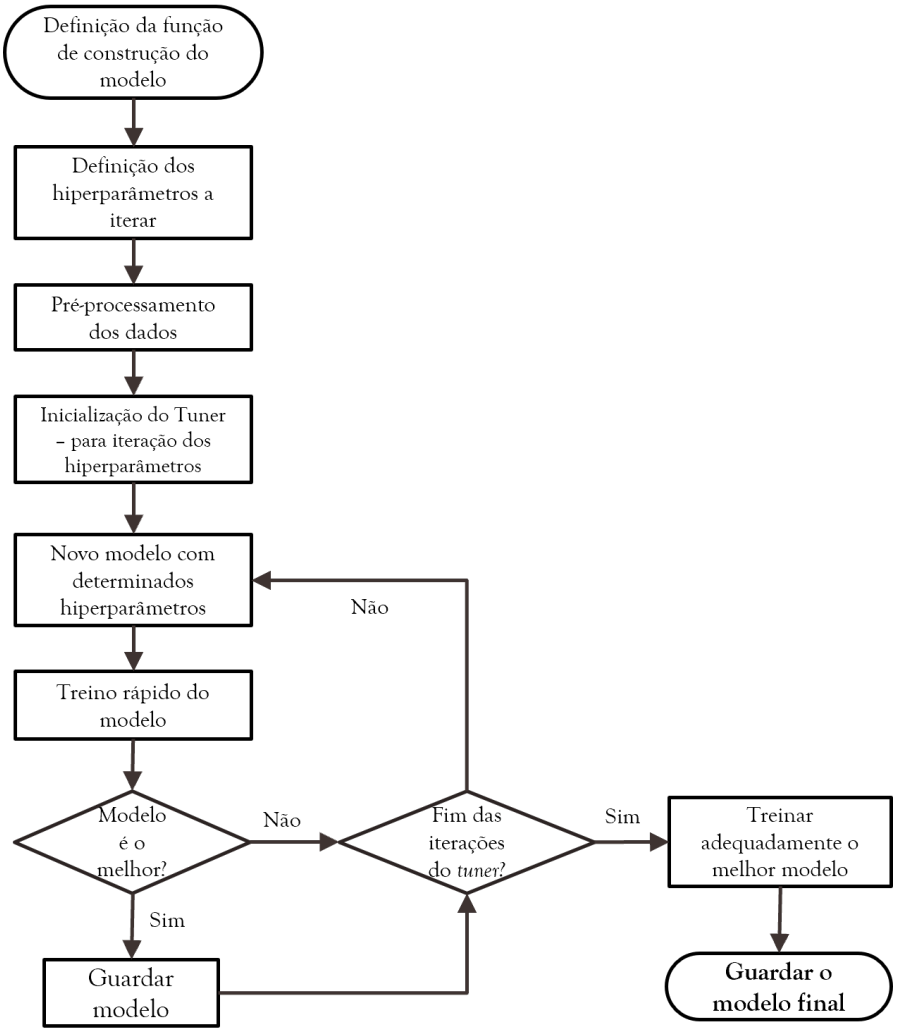


Figura 4.8: Fluxograma representativo do processo de construção de um modelo para previsão.

Arquitetura do *Autoencoder*

A utilização do AutoML, para além de permitir a construção de um modelo mais otimizado através da refinação dos parâmetros, veio também trazer uma maior versatilidade ao nível da própria arquitetura do *autoencoder*. Isso porque não houve uma definição fixa do número de camadas nem da dimensão das mesmas, sendo que para cada variável, o *tuner* (instrumento de otimização do AutoML) realizou um processo exaustivo de procura pela combinação de parâmetros que melhor retratava o seu comportamento. No entanto, foi necessário estabelecer uma estrutura base para os *autoencoders*, a qual se encontra representada na Figura 4.9.

Como se pode visualizar na Figura 4.9, o modelo construido possui duas camadas iniciais LSTM, com as demais a serem camadas densas, tratando-se portanto de um *autoencoder* autorregressivo. As camadas LSTM introduzem no *autoencoder* a capacidade de identificação de dependências temporais na série temporal fornecida. Isso porque

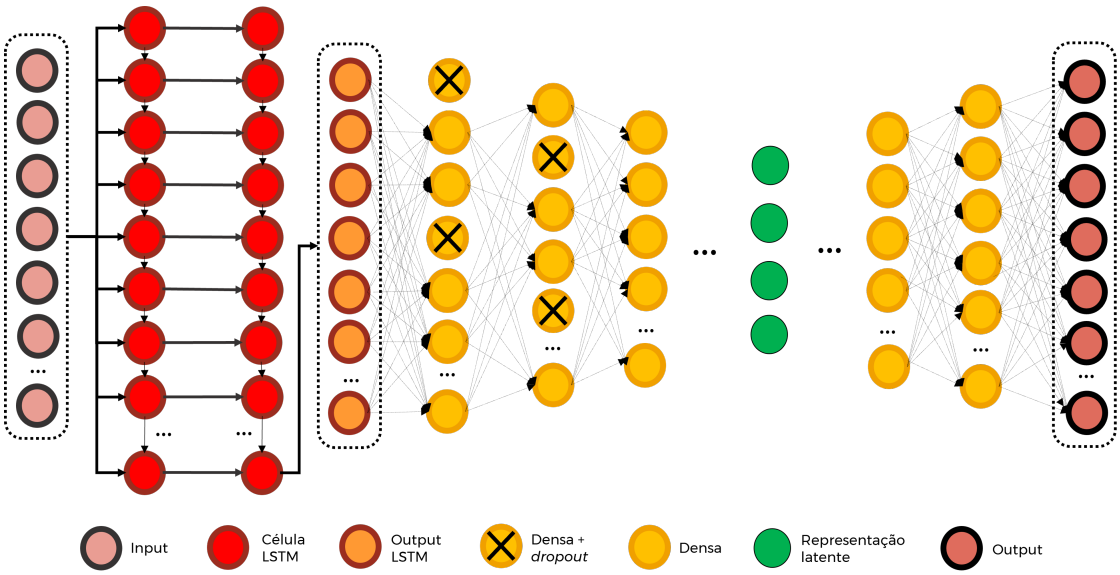


Figura 4.9: Arquitetura base dos *autoencoders* construídos.

têm capacidade de memória, sendo capazes de guardar informações relativas a pontos e espaços temporais anteriores, propagando-as para pontos temporais seguintes que apresentem semelhanças. Por outro lado, as restantes camadas densas são responsáveis pela redução dimensional dos dados até à sua representação latente, no *encoder*, e posterior descompressão novamente até ao seu estado inicial.

A primeira camada LSTM recebe como *input* um vetor com comprimento y, sendo que $y \in \mathbb{Z}^+$, representando o número de pontos temporais prévios necessários para realizar uma previsão futura. Cada célula desta camada processará este vetor de dimensão y, colocando à saída também um vetor da mesma dimensão. Assim sendo, ter-se-á como *output* um conjunto de N vetores de dimensão y, com N a representar o número de células da camada inicial LSTM. Já na segunda camada LSTM, apenas a última célula retornará o *output*, que volta basicamente a ser um vetor com comprimento y.

A partir deste *output*, há um conjunto de camadas densas que tratam de fazer a compressão dimensional inicial no *encoder* até ao estado de representação latente com posterior expansão dimensional. No final, o *output* do *autoencoder* será o tal vetor de comprimento y, cujos valores devem ser o mais próximos possível do vetor inicial que entrou no *autoencoder*.

Definição dos Parâmetros a Iterar

Partindo da arquitetura base apresentada acima, utilizou-se como já referido o AutoML com recurso à biblioteca Python KerasTuner. Esta permitiu a implementação do *tuner*, objeto de otimização que permitiu realizar várias iterações onde explorou diferentes valores para vários hiperparâmetros. Os intervalos de exploração dos parâmetros encontram-se expostos na Tabela 4.1. Já a explicação de cada hiperparâmetro é apresentada de seguida:

- **Dimensão da camada inicial LSTM** - o *tuner* testa ao longo das iterações um conjunto de diferentes valores inteiros, que representam o número de células da

camada inicial LSTM;

- **Número de camadas** - também associado a um valor inteiro, define o número de camadas (segunda LSTM mais as densas do *encoder*) até ao ponto da representação latente, inclusive;
- ***Dropout* entre camadas** - dependendo do número de camadas, é definido um vetor constituído por um conjunto de números reais entre os valores de 1.1 e 1.9. Cada um destes números representará a "queda" do número de células entre camadas consecutivas (por exemplo, começando na camada n, e definindo um *dropout* d para a camada seguinte $n+1$, a dimensão da camada seguinte será dada pelo valor inteiro que resulta da divisão $x_{n+1} = x_n/d$);
- **Taxa de aprendizagem** - de entre um conjunto de valores fixos, o *tuner* escolhe um deles para definir a taxa de aprendizagem do modelo. Esta taxa representa a velocidade com que o modelo converge para os seus parâmetros finais;
- ***Dropout*** - pode ser definido como um valor real entre 0 e 1. Neste caso, entre as duas primeiras camadas densas no *encoder* haverá um *dropout*. Este tenta impedir que as células da camada acabem por se tornar demasiado dependentes dos dados de entrada. Este não deve ser confundido com o *dropout* entre camadas referido acima.

Tabela 4.1: Parâmetros otimizados com recurso ao AutoML.

Dimensão camada LSTM inicial	**Número de camadas do encoder**	***Dropout* entre camadas**	***Dropout***	**Taxa de aprendizagem**
30 - 80 (Números Inteiros)	5 - 10 (Números Inteiros)	1.1 - 1.9 (Números Reais)	0 - 1 (Números Reais)	0.01, 0.001, 0.0001

Assim, com recurso ao AutoML, foi possível a partir de uma arquitetura base como a representada na Figura 4.9 construir modelos diferentes e adaptados ao comportamento de cada variável. Cada um acabou por ter um número diferente de camadas, de nodos por camada, etc.

Pré-processamento dos Dados

Numa primeira fase de treino e teste dos algoritmos, os dados utilizados foram já dados reais retirados ao longo de uma semana do caso de estudo do compressor. Este, conectado a um analisador de energia, possui um conjunto de cerca de 60 variáveis a ser constantemente monitorizadas. Contudo, houve apenas a construção de modelos para 10 delas. As variáveis escolhidas surgem no seguimento do trabalho de Oliveira [7].

A sua escolha das variáveis baseou-se no cálculo do *Pearson Correlation Coefficient*, um indicador que avalia o quão duas variáveis estão de facto relacionadas. Este pode ter valores entre -1 e 1, sendo que quanto mais próximos dos extremos, maior a relação entre as duas variáveis. Os sinais apenas indicam a direção da associação, *i.e.*, com o coeficiente positivo ambas as variáveis aumentam no mesmo sentido, e com o coeficiente negativo as variáveis aumentam em sentidos inversos. A aplicação deste coeficiente permitiu que se agrupassem as cerca de 60 variáveis em 10 grupos de variáveis altamente correlacionadas

(coeficientes superiores a 0.8), com as variáveis a prever a refletirem o comportamento global do grupo de variáveis associado. Estas 10 variáveis encontram-se descritas na Tabela 4.2, sendo que a sua nomenclatura surge da documentação do analisador de energia implementado no caso de estudo [55].

Tabela 4.2: Descrição das 10 variáveis utilizadas para o treino e teste dos algoritmos.

Variável	**Descrição**
C_phi_L3	Fator de potência associado à fase 3 da alimentação do compressor
F	Frequência (Hz) da corrente alimentada ao compressor
H_TDH_I_L3_N	Distorção harmónica total na corrente da fase 3
H_TDH_U_L2_N	Distorção harmónica total na tensão da fase 2
I_SUM	Soma vetorial das correntes das três fases a alimentar o compressor
P_SUM	Potência total consumida pelo compressor
ReacEc_L1	Energia reativa capacitiva na primeira fase
ReacEc_L3	Energia reativa capacitiva na terceira fase
RealE_SUM	Soma total da energia a ser consumida pelo equipamento
U_L1_N	Tensão entre a fase 1 e o neutro da alimentação do compressor

Antes dos dados poderem ser alimentados para a construção dos algoritmos, foi necessário submetê-los a um pré-processamento, que envolveu 3 etapas distintas:

1. **Limpeza dos dados** - a capacidade de previsão de um *autoencoder* está muito dependente da qualidade dos dados fornecidos como *input*. Nestes não devem constar anomalias. Num ambiente industrial onde há uma recolha constante de dados, torna-se complicado que haja alguém a olhar para os mesmos e a definir o que é e não é uma anomalia. Assim, utilizou-se uma ferramenta para promover uma suavização dos dados, a EMA (*Exponential Moving Average*). A partir de um conjunto de pontos temporais, a EMA faz o cálculo de uma média dos valores desses pontos, dando maior ênfase aos pontos mais recentes. Com a EMA é possível suavizar a curva comportamental de determinada variável, tentando quer evitar que haja valores muito extremos, quer também que haja demasiado ruído ao longo dos dados. Assim, é possível suavizar dadas anomalias que eventualmente os dados provenientes do equipamento possam trazer. A Figura 4.10 representa, para uma das variáveis de treino, a diferença entre os dados antes e depois da suavização.

2. **Normalização dos dados** - após a suavização procedeu-se a uma normalização dos dados. Esta normalização serviu apenas para compactar os dados num intervalo de valores mais curto (entre 0 e 1). Certas variáveis possuíam uma grande magnitude ao nível dos seus valores, com dados, por exemplo, a variar entre 80 mil e 120 mil, o que tornava mais complicada a construção e implementação dos modelos;

3. **Segmentação da série temporal** - para fornecer os dados ao *autoencoder* e posteriormente obter previsões, foi necessária uma segmentação da série temporal. Como se pode ver na Figura 4.11, esta divisão consistiu na construção de duas estruturas distintas: a primeira (X) um conjunto de vetores, cada um com a dimensão igual ao número de pontos temporais a prever; a segunda (y) um vetor único, com dimensão

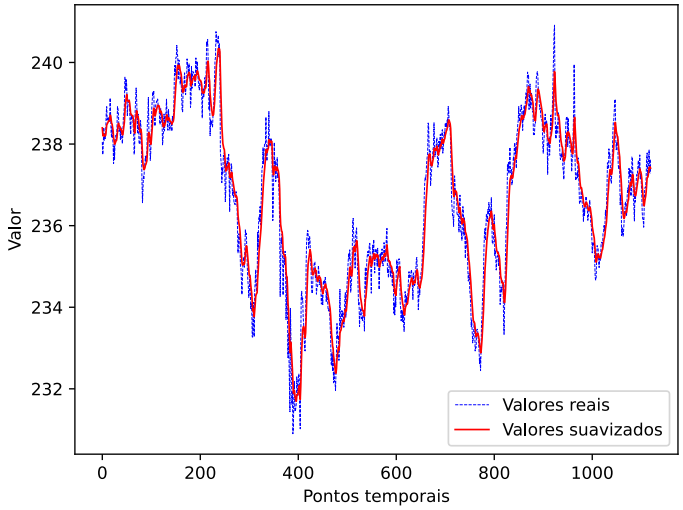


Figura 4.10: Comparação de parte dos dados antes e depois da suavização com recurso à EMA (variável U_L1_N)

igual ao número de vetores da primeira estrutura. Assumindo que são necessários 5 pontos temporais para fazer uma previsão: o primeiro vetor de X consistirá nos 5 primeiros valores dos dados, com o primeiro elemento de y a ser o ponto 6 da série; depois, o segundo vetor de X terá os elementos 2 a 6 da série temporal, e o segundo elemento de y será o ponto 7 da série. Desta forma, é sempre possível obter à saída o valor de uma previsão relativa a um determinado número de pontos temporais prévios.

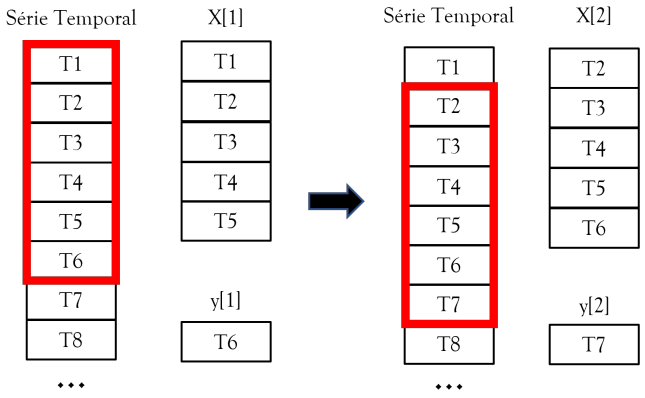


Figura 4.11: Esquema simplificado da divisão da série temporal.

Processo Final de Treino e Construção

Após todo o processo de definição do modelo base, definição dos parâmetros a otimizar via *tuner* e do pré-processamento dos dados, foi possível realizar a construção dos modelos.

Esta construção, como exposto previamente no fluxograma da Figura 4.8, dividiu-se em duas fases fundamentais:

- Um primeiro processo iterativo no qual o *tuner*, a partir de um conjunto de parâmetros definidos pelo utilizador (função objetivo para o processo de otimização do modelo, número de iterações por cada modelo analisado, entre outros), realizava um processo iterativo de busca dos melhores hiperparâmetros associados ao modelo. Este processo envolveu a construção, iteração a iteração, de um modelo com determinados hiperparâmetros e um treino rápido desse modelo. No fim do treino, era guardado o valor correspondente à função objetivo definida (que se pretendia minimizar), sendo esse o valor que determinava o melhor modelo;
- Após completa a busca do *tuner* pelo melhor conjunto de hiperparâmetros para o modelo em questão, o melhor modelo obtido era treinado de forma mais exaustiva, de forma a minimizar o melhor possível a função de custo definida. No final deste treino é que se tinha o modelo final definido para dada variável que era guardado sob a forma de um ficheiro, para posteriormente ser implementado nas previsões.

Para o treino e comparação dos algoritmos, a função de custo definida e usada foi o RMSE (*Root Mean Squared Error*), apresentado na Equação 4.1. Este indicador dá uma noção da "distância" média, para todos os pontos temporais, entre o valor real e respetiva previsão. Assim, quanto melhor o modelo, menor será o RMSE, e mais próximas as previsões estão da realidade.

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(y_{\text{prev},i} - y_{\text{real},i})^2}{N}} \tag{4.1}$$

Posto isto, no processo de otimização dos hiperparâmetros via *tuner* são realizados treinos rápidos em cada iteração, pelo que o melhor modelo resultante desse processo é aquele que revela o menor RMSE. Posteriormente, esse modelo é alvo de um treino mais extenso no qual se tenta reduzir ainda mais o RMSE. Estes processos são repetidos para todas as variáveis, e no final obtém-se um conjunto de modelos cada um otimizado conforme o comportamento específico da sua variável. Na Figura 4.12 são apresentadas duas das estruturas obtidas para dois modelos distintos.

Como se pode constatar na Figura 4.12, a partir da mesma arquitetura base apresentada anteriormente foi possível obter dois modelos totalmente distintos. Um deles consegue atingir uma representação comprimida dos dados através de um vetor de 9 dimensões (Figura 4.12a), enquanto o outro necessita de um vetor já com 16 dimensões (Figura 4.12b); há também um número de células por camada distinto, refletindo diferentes valores do *dropout* entre células, e um número de camadas que varia. Contudo, deve-se salientar que a arquitetura base continua patente nos resultados: 2 camadas iniciais LSTM, com um vetor à saída de iguais dimensões ao que entra na primeira camada LSTM (neste caso com 30 pontos temporais prévios).

4.3.3 Previsões e Deteção de Anomalias

Com os modelos para cada variável construidos, foi possível fazer a sua implementação para as tarefas de previsão e deteção de anomalias. Estes dois processos foram realizados

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 30, 80)	26240
lstm_3 (LSTM)	(None, 43)	21328
dense_7 (Dense)	(None, 23)	1012
dropout_2 (Dropout)	(None, 23)	0
dense_8 (Dense)	(None, 12)	288
dropout_3 (Dropout)	(None, 12)	0
dense_9 (Dense)	(None, 10)	130
dense_10 (Dense)	(None, 9)	99
dense_11 (Dense)	(None, 9)	90
dense_12 (Dense)	(None, 10)	100
dense_13 (Dense)	(None, 12)	132
dense_14 (Dense)	(None, 23)	299
dense_15 (Dense)	(None, 30)	720

(a) Variável H_TDH_I_L3_N

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 30, 80)	26240
lstm_3 (LSTM)	(None, 72)	44064
dense_21 (Dense)	(None, 65)	4745
dropout_2 (Dropout)	(None, 65)	0
dense_22 (Dense)	(None, 59)	3894
dropout_3 (Dropout)	(None, 59)	0
dense_23 (Dense)	(None, 31)	1860
dense_24 (Dense)	(None, 16)	512
dense_25 (Dense)	(None, 16)	272
dense_26 (Dense)	(None, 30)	510

(b) Variável C_phi_L3

Figura 4.12: Comparação entre a arquitetura dos modelos construídos para duas variáveis distintas (imagens resultantes da aplicação de uma função da biblioteca Tensorflow).

simultaneamente num único *script* em Python, sendo que apenas foi necessário definir a velocidade com que os dados eram injetados na base de dados (ou na plataforma C2E). Na Figura 4.13 é apresentado um fluxograma relativo ao funcionamento destes processos.

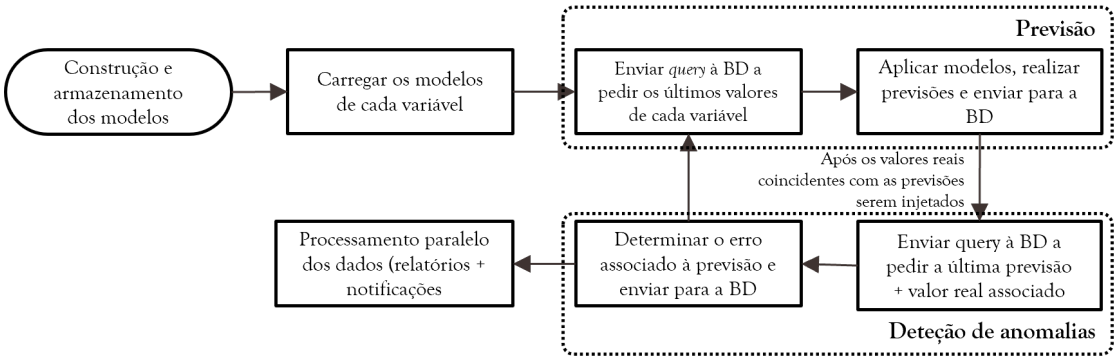


Figura 4.13: Fluxograma relativo ao processo cíclico de previsão e deteção de anomalias.

Após o processo de construção num *script* dedicado, os modelos são guardados num diretório específico. Para que estes possam ser implementados nas tarefas de previsão, é necessário carregá-los a partir desse diretório. Após outras inicializações, tais como a conexão ao *bucket* do InfluxDB onde a informação real é armazenada, entra-se num processo repetitivo de previsões e deteção de anomalias. Quando há uma nova injeção de dados no InfluxDB, é enviada uma *query* onde para cada variável são retirados os últimos N valores necessários para se realizar uma previsão. Depois, realizam-se as previsões futuras, que são enviadas de volta para a base de dados. Quando o novo grupo de dados reais associado às previsões efetuadas chega, conduz-se o processo de deteção de anomalias.

No que toca a este processo, em vez de se definir um erro fixo entre previsão e valor real

que determinava o que era e não era anomalia, aplicaram-se limites dinâmicos associados a cada variável. Como já se referiu anteriormente, associado a cada modelo surge um valor de RMSE, que reflete o desvio médio das previsões efetuadas (num conjunto de dados de teste) face aos valores reais. Definindo, por exemplo, um valor $\alpha > 1$, tem-se que uma anomalia é qualquer conjunto de pontos tal que se verifique:

$$|y_{\text{prev}} - y_{\text{real}}| > \alpha \times \text{RMSE} \tag{4.2}$$

ou seja, qualquer par previsão e valor real cuja diferença se superiorize α vezes ao RMSE do respetivo modelo da variável. Deste modo, o processo de deteção de anomalias tornou-se mais dinâmico: cada modelo possuía um RMSE associado, e cada variável um limite a definir o que era uma anomalia dependente da qualidade do seu modelo.

Posto isto, após se realizarem as previsões e se esperar pelos valores reais associados é feita uma análise no *script* Python a estes dois valores. Essa análise implica então a determinação do erro, dado pelo RMSE, e envio do mesmo para a base de dados. Após isto dá-se por finalizado um ciclo de previsão e deteção de anomalias e um novo ciclo é iniciado. Tal como se pode ver na Figura 4.13, é importante referir que paralelamente ocorre um processamento mais profundo dos dados, com a geração de relatórios, envio de notificações, entre outros.

4.3.4 Geração dos Relatórios

Como exposto na secção anterior, a implementação do agente inteligente implicou um processo repetitivo de recolha de novos dados, geração de previsões e posterior deteção de anomalias sobre essas previsões. Paralelamente a este processo ocorria, como se pode ver na Figura 4.13, um processamento local dos dados com a geração de relatórios que continham o resultado da deteção de anomalias. Estes relatórios, como explicado no Capítulo 3, eram posteriormente enviados automaticamente via email para o utilizador.

Estes relatórios são de extrema importância, pois permitem ao utilizador receber periodicamente um resumo das ocorrências anómalas no chão de fábrica ao longo da última hora ou do último turno. Guardando um histórico destes relatórios, pode ser possível a identificação de certas tendências que poderão levar a que o utilizador consiga encontrar pontos de melhoria e eventualmente otimizar os consumos energéticos.

Sempre que há um novo ciclo de previsões e deteção de anomalias, são calculadas as diferenças entre os valores reais e as previsões associadas. Comparando para cada variável a diferença com o limite definido a partir do RMSE, classificam-se os pontos como normais ou anómalos. Todos os pontos que sejam anomalias (diferença é superior ao limite) são adicionados a um ficheiro Excel, que no fim de uma hora ou de um turno é enviado por email para o utilizador. Neste email, para além do relatório, tem-se também alguma informação relevante relativa aos acontecimentos prévios. Um exemplo do email pode ser visualizado na Figura 4.14, e a estrutura do ficheiro Excel na Figura 4.15.

O email enviado (Figura 4.14) contém informações relativas ao número de anomalias que ocorreram na última hora ou turno, o número de previsões por variável consideradas anómalas e ainda a severidade das anomalias. A definição da sua severidade é baseada no erro já apresentado na Equação 4.2. Tendo o tal α base a partir do qual um ponto pode ser considerado anómalo, podem-se definir depois outros valores β que determinam a severidade da mesma. Na Equação 4.3 é apresentado matematicamente o que foi dito.

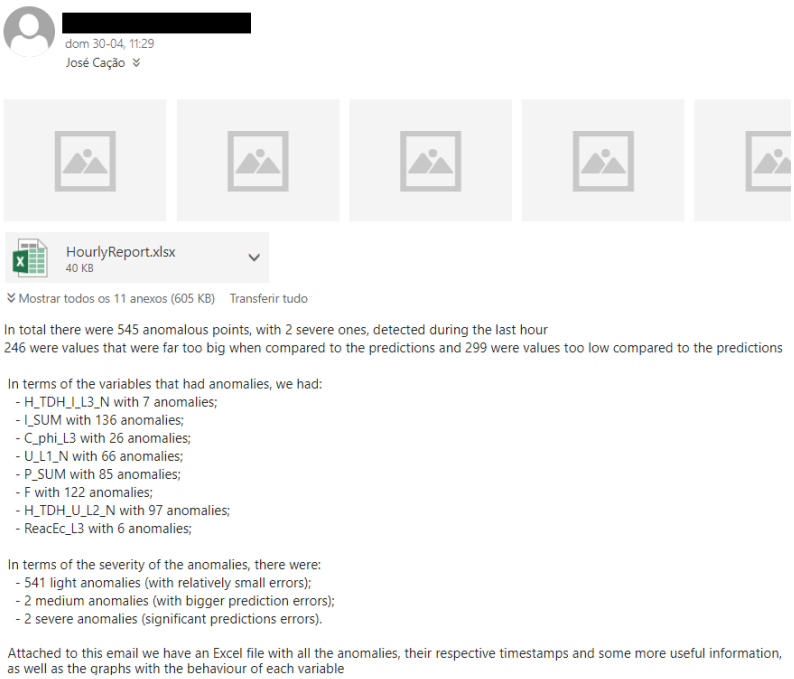


Figura 4.14: Exemplo de um email enviado pelo agente inteligente, neste caso com informações relativas à última hora de medições e previsões.

Timestamp	Predicted Value	Real Value	Norm Difference	Thresholds	Severity	Notes	Variable
13:17:45 - 04/29/2023	197650595,8	197840945	0,00428	0,000278354	Severe	The real value was to high when compared with the prediction given	RealE_SUM
13:17:45 - 04/29/2023	1,002023404	0,681025552	0,23461	0,179911589	Light	The real value was far to low than the prediction given	I_SUM
13:17:52 - 04/29/2023	197840363,7	197857332	0,00038	0,000278354	Light	The real value was to high when compared with the prediction given	RealE_SUM
13:17:52 - 04/29/2023	0,643830657	0,90021927	0,18739	0,179911589	Light	The real value was to high when compared with the prediction given	I_SUM
13:17:54 - 04/29/2023	197801741,7	197863720,3	0,00139	0,000278354	Light	The real value was to high when compared with the prediction given	RealE_SUM
13:17:54 - 04/29/2023	49,99444085	50,01987048	0,10012	0,085353607	Light	The real value was to high when compared with the prediction given	F
13:17:54 - 04/29/2023	5,723654135	7,600216623	0,282	0,135938957	Light	The real value was to high when compared with the prediction given	H_TDH_U_L2_N
13:18:00 - 04/29/2023	197915137,8	197875417,6	0,00089	0,000278354	Light	The real value was far to low than the prediction given	RealE_SUM
13:18:00 - 04/29/2023	17626,82586	89449,37641	0,54116	0,283434652	Light	The real value was to high when compared with the prediction given	P_SUM
13:18:00 - 04/29/2023	197774516,7	197881411,6	0,0024	0,000278354	Medium	The real value was to high when compared with the prediction given	RealE_SUM
13:18:00 - 04/29/2023	0,383692525	0,648931767	0,19386	0,179911589	Light	The real value was to high when compared with the prediction given	I_SUM
13:18:05 - 04/29/2023	50,00236852	49,97350676	0,11363	0,085353607	Light	The real value was far to low than the prediction given	F

Figura 4.15: Parte da folha Excel gerada pelo agente inteligente, onde são guardadas um conjunto de informações relativas à deteção de anomalias.

$$\begin{cases} \text{Anomalia "Leve":} & \alpha \times \text{RMSE} < \text{Erro} < \beta_1 \times \text{RMSE},\ \beta_1 > \alpha \\ \text{Anomalia "Média":} & \beta_1 \times \text{RMSE} < \text{Erro} < \beta_2 \times \text{RMSE},\ \beta_2 > \beta_1 \\ \text{Anomalia "Severa":} & \beta_2 \times \text{RMSE} < \text{Erro} < \beta_3 \times \text{RMSE},\ \beta_3 > \beta_2 \end{cases} \tag{4.3}$$

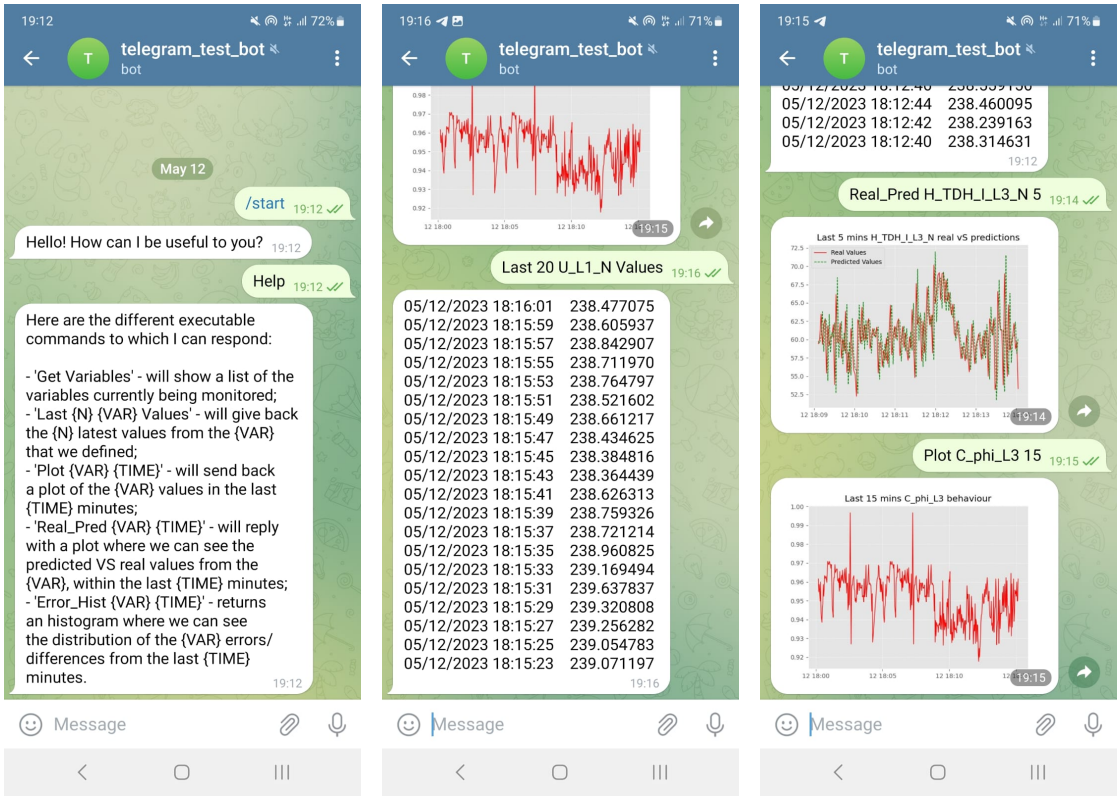
Com este parâmetro a ser analisado, é depois possível ter uma maior flexibilidade ao nível das notificações, com avisos de diferentes magnitudes: pode-se, por exemplo, apenas considerar necessário o envio de notificações quando as anomalias têm elevada severidade, e definir-se alertas de diferentes naturezas baseados nestes limites definidos β_1, β_2 e β_3.

Juntamente com este processamento dos dados, são ainda enviados um conjunto de gráficos que refletem as diferenças entre as previsões e os valores reais, e o tal relatório em Excel que surge na Figura 4.15.

4.3.5 Envio de Notificações

Anteriormente já foram apresentadas duas formas essenciais de interação com o utilizador: a interface gráfica desenvolvida em Grafana, que permite uma rápida e fácil visualização dos dados; e a geração e envio dos relatórios, mencionada na secção anterior como sendo uma forma de manter um histórico com um conjunto de informação processada. Falta então a dimensão do envio de notificações de forma automatizada.

Para se conseguir realizar este tipo de interação, utilizou-se a aplicação Telegram [56]. Esta permite, de uma forma muito simples, criar *bots* que após configurados, podem realizar um conjunto de operações de forma automatizada. Depois, utilizando algumas bibliotecas disponíveis em Python, é possível fazer a configuração dos *bots* para que estes consigam quer processar informação e enviá-la para o utilizador, quer mesmo enviar automaticamente notificações via Telegram. Nas Figuras 4.16 e 4.17 são expostas estas duas funcionalidades dos *bots* (mais informação relativa à criação e configuração dos *bots* pode ser encontrada no Apêndice C).



(a) Consulta dos possíveis comandos

(b) Consulta dos últimos valores de uma variável

(c) Análise gráfica de resultados

Figura 4.16: Demonstração das potencialidades do *bot* Telegram desenvolvido.

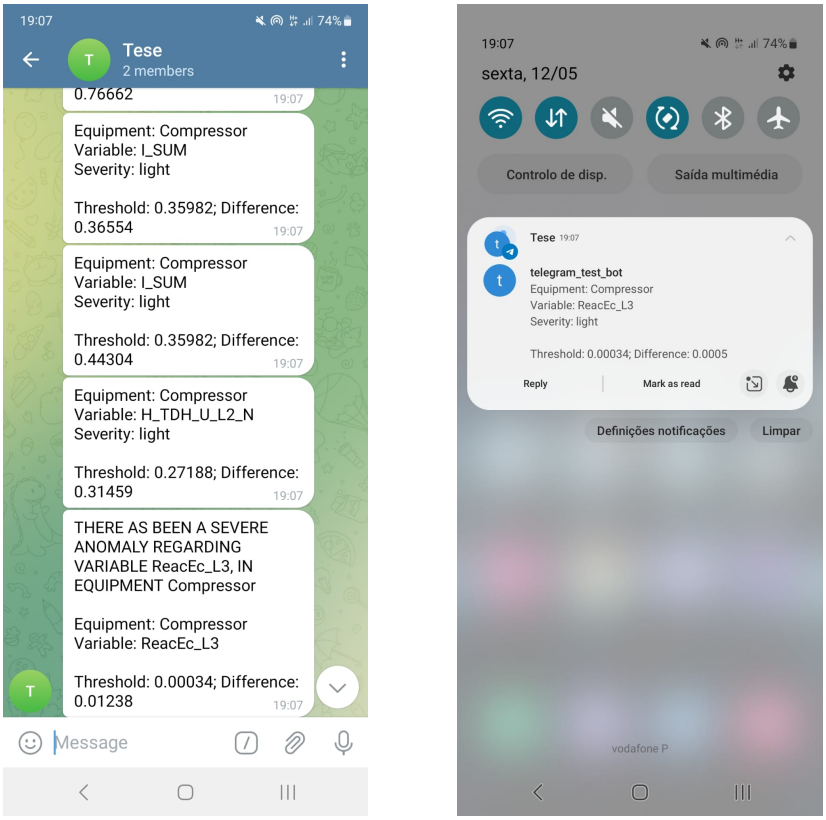


Figura 4.17: Notificações automatizadas do *bot* Telegram.

De seguida são explicadas as duas dimensões associadas ao *bot* Telegram criado:

1. **Consulta de dados históricos** - esta primeira dimensão implica que o utilizador envie comandos para o *bot*. Reconhecendo-os, o *bot* envia a informação pedida para o utilizador. Essa informação pode envolver, por exemplo, o histórico das últimas leituras associadas a determinada variável do caso de estudo, gráficos de comparação entre as previsões e os valores reais, informações relativas aos erros de previsão, entre outras. Assim, mesmo não tendo acesso à interface de visualização gráfica construída em Grafana, é possível ir ao telemóvel, inicializar uma conversação com o *bot* e enviar-lhe pedidos. Exemplos desses pedidos podem-se encontrar na Figura 4.16.
2. **Envio de notificações em tempo real** - esta segunda potencialidade já implica um envio automatizado de notificações por parte do *bot* sem que haja a necessidade do utilizador fazer um pedido. Neste caso, é aproveitado o processamento realizado na deteção de anomalias. Ao se detetar uma anomalia numa dada variável, é enviada de imediato uma notificação para o utilizador, informando-o em tempo real que poderá estar a ocorrer algum problema com o equipamento a ser monitorizado. Desse modo, o utilizador, analisando a própria mensagem enviada, pode decidir logo se pretende atuar sobre o equipamento ou não. Exemplos destas notificações podem ser encontrados na Figura 4.17, com o nome das variáveis que tiveram possíveis anomalias, assim como os valores associados aos erros a serem mostrados.

Posteriormente, é possível explorar notificações de outros tipos com este *bot*, tais como o envio de informações associadas à produção, do próprio relatório via Telegram, entre outras. Mais detalhes relativos à utilização do *bot* Telegram podem ser encontrados no Apêndice C.

4.4 Implementação do Caso de Estudo Bosch

Após a construção da arquitetura bloco a bloco, e validação final da arquitetura concetual em ambiente simulado, foi então realizada a implementação da arquitetura ao nível de um caso de estudo real nas instalações da Bosch Termotecnologia Aveiro. Este caso de estudo envolveu a monitorização e processamento das variáveis associadas aos consumos de um compressor atualmente a funcionar nas instalações da empresa, cujos dados históricos das variáveis foram já sendo utilizados, especialmente para o treino de algoritmos e simulação da injeção real de dados.

A arquitetura final que foi possível implementar ao nível do chão de fábrica Bosch é apresentada na Figura 4.18.

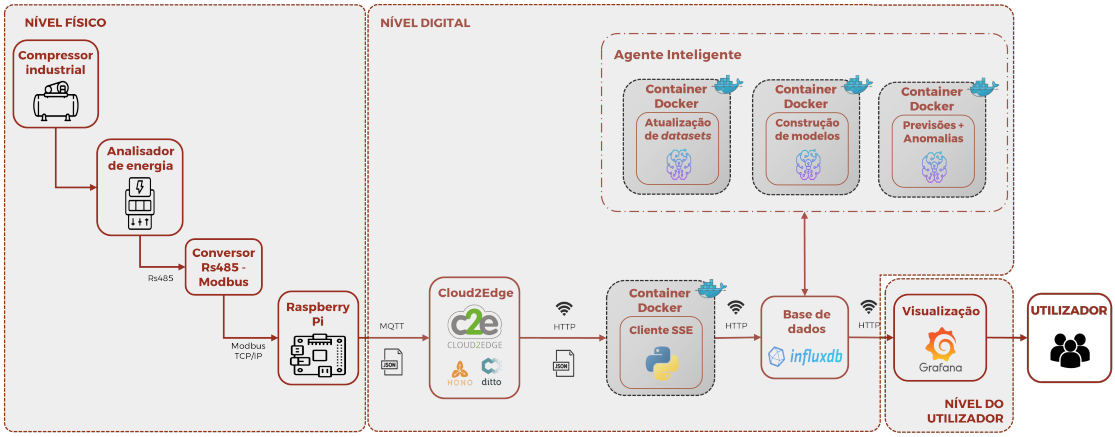


Figura 4.18: Arquitetura final implementada em chão de fábrica Bosch.

Olhando para esta arquitetura, são de salientar alguns pontos-chave:

- Toda a parte de recolha de dados ao nível do compressor, nomeadamente os diferentes dispositivos intermédios entre compressor e *gateway*, e a própria programação do *gateway* foram já feitos anteriormente, não tendo sido necessário proceder a uma programação exaustiva destes equipamentos;
- Devido a algumas incompatibilidades entre o servidor do caso de estudo e as bibliotecas de ML usadas, houve a separação de serviços em duas máquinas. O servidor, máquina local, possuía o cliente SSE, InfluxDB e o Grafana. Já uma máquina externa corria os serviços do agente inteligente;
- Tendo em conta que a rede 5G usada no caso de estudo é fechada e não tem conexão com o exterior, não foi possível implementar os serviços de envio de notificações e emails. Isso porque não era possível ser feita a conexão com os servidores do Telegram e Gmail, respetivamente;

- Para além de fechada, a rede 5G utilizada pelo caso de estudo é isolada da rede produtiva. Assim, não foi possível integrar o Nexeed MES na arquitetura.

De seguida são apresentados alguns pontos relevantes associados à implementação, e diferenças face à arquitetura que foi sendo desenvolvida em ambiente simulado.

4.4.1 Recolha de Dados do Compressor

Comparando desde logo a arquitetura concetual proposta no Capítulo 3 (Figura 3.1) com a arquitetura implementada (Figura 4.18), notam-se claramente algumas diferenças. Isso porque a própria empresa já tinha implementados um conjunto de interfaces e equipamentos, à luz de trabalhos anteriores desenvolvidos [5]–[7], que permitiam uma recolha dos dados provenientes do caso de estudo do compressor.

Posto isto, as informações relativas aos consumos energéticos do compressor são recolhidas por um analisador de energia, o Janitza UMG 96rm [55]. Como o analisador de energia apenas é capaz de enviar os dados via protocolo Rs485, a informação é primeiro direcionada para um equipamento intermédio, o Lumel PD8 [57], capaz de realizar a conversão de mensagens Rs485 para o protocolo Modbus TCP/IP. Por conseguinte, através deste último protocolo a informação é remetida para um Raspberry Pi 3b+ [58] a funcionar como o *gateway*. Este último equipamento, via MQTT, é responsável por fazer a conexão à camada digital, com a informação a ser enviada para o Eclipse Hono.

4.4.2 Implementação da Camada Digital e do Utilizador

No caso da camada digital, pode-se dizer que as arquiteturas concetual desenvolvida e implementada são no seu global muito similares. A principal mudança foi a forma de implementação: inicialmente esperava-se que toda a implementação fosse realizada num único servidor, com todos os serviços desenvolvidos e *softwares* utilizados a correr no mesmo; devido às dificuldades inerentes aos serviços que implementam os modelos de ML, houve uma separação em duas máquinas. Como se pode ver na Figura 4.18, tem-se todos os softwares externos - C2E, Grafana e InfluxDB - implementados no servidor juntamente com o cliente SSE, responsável pela ponte entre C2E e armazenamento. Numa máquina externa tem-se todos os serviços ligados ao agente inteligente e que requerem a utilização de bibliotecas de ML, responsáveis pela construção e atualização dos modelos, atualização dos dados de treino e ainda das tarefas de previsão e deteção de anomalias.

No entanto, apesar dos serviços funcionarem em máquinas separadas, há um ponto comum aos mesmos, a base de dados: os dados injetados na base de dados pelo serviço SSE a correr no servidor devem ser remetidos para o serviço do agente inteligente, na máquina externa. Consequentemente, de forma periódica, o serviço de atualização dos dados deve recolher novos dados também do InfluxDB, e alimentá-los ao construtor de modelos para que este consiga atualizar os mesmos de acordo com potenciais novas tendências. Para isto ser possível, foi necessário conectar ambas as máquinas à mesma sub-rede (sendo o IP do servidor 192.168.17.5, associar à máquina externa um IP do tipo 192.168.17.XXX). Depois, bastou configurar as credenciais do InfluxDB da máquina local na máquina externa. Com isto, foi então possível ter os serviços das duas máquinas a funcionar em simultâneo, com os dados reais, das previsões e das anomalias a serem processados pelo agente inteligente, enviados para a base de dados e visualizados no Grafana.

Contudo, como já referido acima, a implementação das funcionalidades do assistente na íntegra não foi possível. Não houve portanto a capacidade de testar o envio de notificações e de emails para o utilizador. Isso porque a rede 5G que o caso de estudo utiliza corresponde a uma *Non Public Network* (NPN). Isso significa que a rede é fechada, privada, e que não há possibilidades de se conectar com servidores externos, tais como o do Telegram (para envio das notificações) e o do Gmail (para envio dos emails). Deste modo, a única dimensão de interação com o utilizador foi mesmo a visualização dos dados e geração de alarmes através de uma interface em Grafana. Adicionalmente, apesar dos relatórios não serem enviados por email, foi sendo realizado o seu armazenamento numa pasta para o efeito.

Na última secção deste capítulo é exposto um pequeno resumo com a organização dos serviços e dos diretórios, nomeadamente ao nível daqueles associados ao agente inteligente na máquina externa.

4.4.3 Organização dos Serviços do Agente Inteligente

Como já mencionado na Secção 4.4.2, foi necessária a construção de um *Docker-Compose* para poder implementar os serviços associados ao agente inteligente. Para isso, construiu-se um diretório no qual constavam um conjunto de ficheiros e pastas, cuja organização é exposta na Figura 4.19. Adicionalmente, na Tabela 4.3 são apresentados os serviços e as pastas partilhadas que estes utilizam.

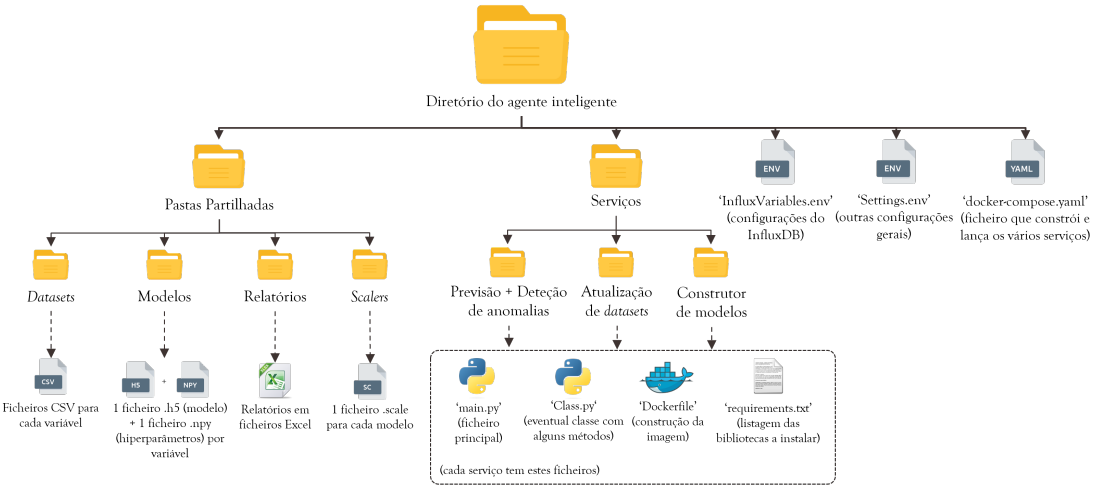


Figura 4.19: Organização geral do diretório relativo ao agente inteligente.

Tabela 4.3: Acesso de cada serviço às várias pastas partilhadas.

	Datasets	**Modelos**	**Relatórios**	***Scalers***
Previsão + Deteção de Anomalias	-	R	W	R
Atualização de *Datasets*	W	-	-	-
Construtor de Modelos	R	W	-	W

Legenda: W - serviços conseguem escrever/colocar novos ficheiros nas pastas; R - serviços conseguem ler/usar informações dos ficheiros nas pastas.

Olhando para a Figura 4.19, no agente inteligente podem-se distinguir dois diretórios fundamentais: as pastas partilhadas e os serviços. Nas pastas partilhadas constam quatro subdiretórios, sendo que como se pode visualizar na Tabela 4.3, diferentes serviços acedem de diferentes formas aos mesmos: por exemplo, o serviço de construção de modelos retira informação que consta nos ficheiros CSV integrados na pasta "*Datasets*", e gera ficheiros que são armazenados nas pastas "Modelos" e "*Scalers*". No que toca a cada um destes subdiretórios, tem-se:

- "*Datasets*" - aqui são armazenados, segundo a forma de ficheiros CSV, registos históricos relativos à última semana de recolha de dados para todas as variáveis a ser monitorizadas e processadas. Estes dados são usados essencialmente na construção e treino dos modelos;
- "Modelos" - nesta pasta, para cada variável, armazenam-se dois tipos de ficheiros: um ficheiro com a extensão ".h5", que corresponde ao modelo já treinado, e um ficheiro do tipo ".npy", com informações relativas a parâmetros usados ao longo do treino. Estes ficheiros são necessários na previsão de deteção de anomalias, de modo a que os modelos possam ser devidamente carregados;
- "Relatórios" - contêm os relatórios horários em formato Excel, gerados pelo serviço de previsão e deteção de anomalias;
- "*Scalers*" - neste caso possuem ficheiros com a extensão ".scale", usados na normalização das variáveis. São gerados pelo construtor de modelos e utilizados pelo serviço de previsão e deteção de anomalias.

Relativamente ao diretório dos serviços, este possui os três serviços que dão origem a três *containers* distintos aquando a implementação do *Docker-Compose*. Como se pode ver na Figura 4.19, cada subdiretório associado a um serviço possui os seguintes ficheiros:

- "main.py" - ficheiro principal, com o código necessário para realizar as devidas tarefas;
- "Class.py" - engloba um conjunto de funções utilizadas no ficheiro "main.py". Implementado essencialmente para simplificar ao máximo o ficheiro principal;
- "requirements.txt" - ficheiro de texto que contém todas as bibliotecas Python, assim como as respetivas versões, necessárias para correr o código do respetivo serviço;
- "Dockerfile" - ficheiro que contém as instruções que permitem construir a imagem e correr o serviço devidamente. Este ficheiro inclui, entre outros, comandos para instalar as bibliotecas contidas no ficheiro "requirements.txt" e para correr o código presente no "main.py".

No que toca aos restantes ficheiros, tem-se dois ficheiros com a extensão ".env", um que contém as configurações devidas para a máquina externa conseguir comunicar com o InfluxDB a correr no servidor, e outro com um conjunto de valores fixos, configurações, etc., usados nos vários serviços implementados. Por fim, tem-se o ficheiro "docker-compose.yaml", o qual é responsável por orquestrar a construção das imagens, associar cada serviço às respetivas pastas utilizadas, e ainda lançar os mesmos sob a forma de *containers* Docker.

Capítulo 5

Análise dos Resultados

Este capítulo dedica-se à exposição dos resultados relativos à implementação da arquitetura desenvolvida. Estes estão divididos em duas partes fundamentais: uma mais ligada aos modelos de *Machine Learning* aplicados pelo agente inteligente, onde é analisado o desempenho dos modelos no que toca às tarefas de previsão e deteção de anomalias e analisada a capacidade de construção dos modelos com recurso a AutoML, e outra relacionada com a própria implementação da arquitetura no chão de fábrica Bosch.

5.1 Análise dos Modelos Implementados

Como foi sendo exposto ao longo do documento, foram desenvolvidos modelos de *autoencoders*, adaptados a cada variável a monitorizar, com o intuito de fazer a previsão do ponto temporal futuro e posteriormente uma deteção de anomalias em tempo real. Assim, havendo identificadas estas duas tarefas distintas, promoveu-se uma avaliação dos modelos quanto às suas capacidades de previsão e de deteção de anomalias. De referir novamente que, apesar de haver uma monitorização de cerca de 60 variáveis ao nível do compressor, apenas 10 foram estudadas com recurso aos algoritmos, no seguimento do trabalho desenvolvido por Oliveira [7]. É ainda de salientar que, pegando num conjunto de dados retirados do chão de fábrica relativos às variáveis a prever, houve uma separação dos mesmos de tal modo que 70% serviram para fazer o treino dos modelos, 20% serviram como validação e os 10% restantes foram então utilizados para fazer os testes.

Nas secções seguintes apresentam-se as metodologias e os resultados dos testes realizados. Adicionalmente, no Apêndice D apresentam-se alguns resultados complementares da análise aos modelos, que incluem gráficos para todas as variáveis das previsões face aos valores reais, e ainda tabelas de comparação relativas à deteção de anomalias.

5.1.1 Resultados das Previsões

Para analisar a capacidade de previsão dos modelos foram utilizados um conjunto de diferentes indicadores relevantes:

- **MAE - *Mean Absolute Error*** - uma das métricas geralmente usadas em modelos regressivos, como é o caso. O MAE (Equação 5.1) representa o erro médio do valor absoluto das diferenças entre valor real e valor previsto pelo modelo. Quanto maior

o MAE, pior é a capacidade de previsão do modelo;

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_{\text{prev},i} - y_{\text{real},i}| \tag{5.1}$$

- **MSE - *Mean Squared Error*** - o MSE (Equação 5.2) é bastante semelhante ao MAE, mas ao invés de fazer o cálculo do módulo das diferenças, acaba por calcular a média das diferenças entre previsões e valores reais ao quadrado. Esta diferença ao quadrado leva a que maiores erros possuam maiores influências no resultado final, algo que o MAE não pondera. À semelhança do MAE, quanto maior o MSE, pior é a capacidade de previsão do modelo;

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_{\text{prev},i} - y_{\text{real},i})^2 \tag{5.2}$$

- r^2 **- Coeficiente de Determinação** - Este indicador (Equação 5.3), de uma forma muito resumida, expressa o quão a variância de uma variável explica a variância da outra. Possui valores entre 0 e 1, pelo que quanto maior o r^2, melhor é o modelo.

$$r^2 = 1 - \frac{\sum_{i=1}^{n}(y_{\text{real},i} - \overline{y})^2}{\sum_{i=1}^{n}(y_{\text{real},i} - y_{\text{prev},i})^2},\ \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_{\text{real},i} \tag{5.3}$$

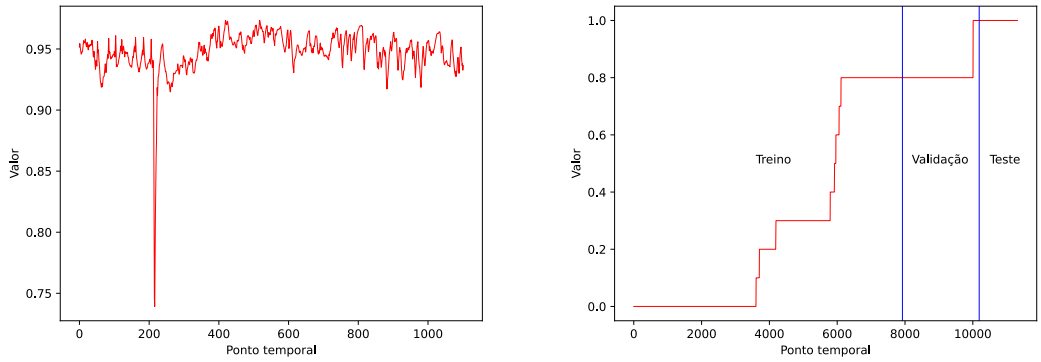
Posto isto, na Tabela 5.1 são apresentados os resultados destas métricas para cada uma das variáveis previstas.

Tabela 5.1: Indicadores obtidos na avaliação dos modelos preditivos.

Variável	**MAE**	**MSE**	r^2
C_phi_L3	0.011791	0.000164	0.447049
F	0.011179	0.000205	0.948757
H_TDH_I_L3_N	0.008170	0.000126	0.866547
H_TDH_U_L2_N	0.013182	0.000298	0.977937
I_SUM	0.015562	0.000452	0.955313
P_SUM	0.020964	0.000678	0.949120
ReacEc_L1	2.34e-05	5.49e-10	-1.49e19
ReacEc_L3	2.27e-05	5.16e-10	-4.14e20
RealE_SUM	2.20e-06	1.02e-11	1.000000
U_L1_N	0.007084	8.86e-05	0.996960

Ao nível do MAE e MSE, todas as variáveis apresentam valores muito baixos, o que é um indicador da qualidade das previsões efetuadas pelos modelos ao longo de todo o conjunto de dados de teste. Porém, para os valores de r^2 há já algumas diferenças. 7 das 10 variáveis apresentam excelentes resultados ao nível do coeficiente de determinação, com valores muito próximos de 1, revelando claramente que para o conjunto de dados de teste o modelo consegue expressar e avaliar corretamente a variação de uma variável em função da outra. Já para a variável C_phi_L3 há uma descida considerável do

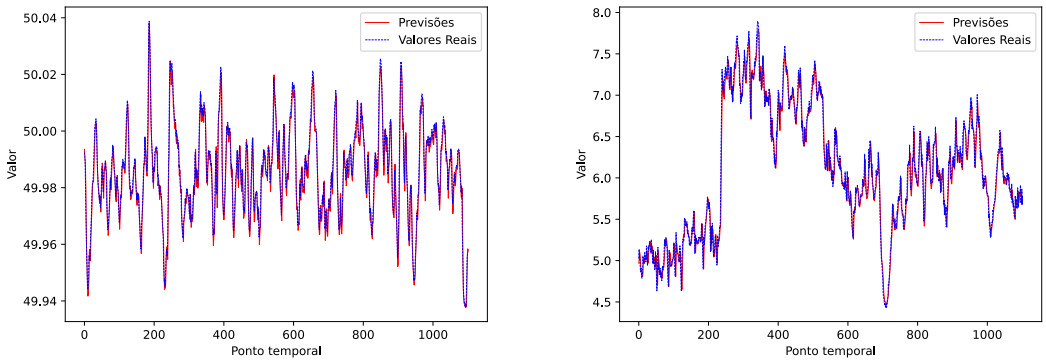
r^2. A razão para esta discrepância face às demais é rapidamente explicada de forma gráfica. Como se pode ver na Figura 5.1a, há um ponto no conjunto de dados de teste que claramente se afasta dos demais (mesmo após suavização) e afeta a qualidade dos resultados, contrastando assim com o MAE e MSE desta variável. Isto só prova o que já foi dito acerca dos *autoencoders*, e da importância de se ter um conjunto de dados "limpo", sem anomalias, para treinar e testar os modelos de forma adequada. Quanto às variáveis ReacEc_L1 e ReacEc_L3, verificam-se valores de r^2 extremos e completamente fora da escala. Isso deve-se ao facto dos seus dados de teste terem praticamente um valor constante, como se pode ver para o caso da variável ReacEc_L1 na Figura 5.1b. Isso leva a uma variância extremamente reduzida, com estes valores de r^2 extremos.



(a) Variável C_phi_L3 (dados de teste) (b) Variável ReacEc_L1 (divisão dos dados)

Figura 5.1: Gráficos de duas das variáveis com piores desempenhos em termos de r^2.

Por fim, para concluir a análise do desempenho dos modelos relativamente às previsões, na Figura 5.2 pode-se visualizar graficamente a excelente capacidade de previsão dos modelos, onde para duas das variáveis são apresentados gráficos com as previsões face aos valores reais no conjunto de dados de teste (no Apêndice D, Secção D.1, podem-se encontrar os gráficos para todas as variáveis).



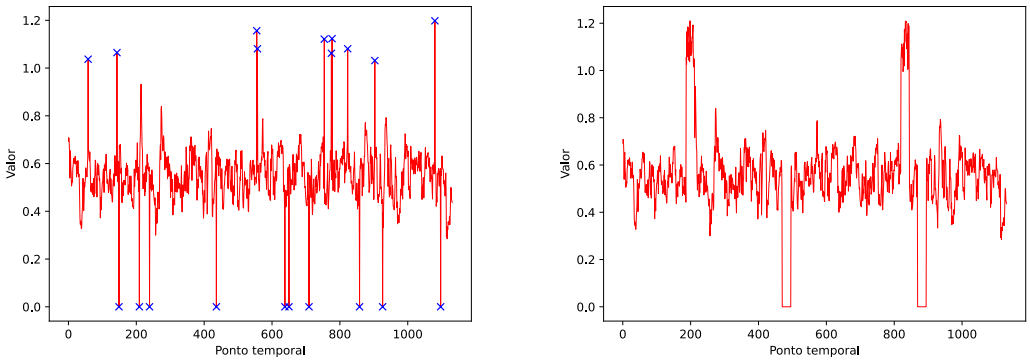
(a) Variável F (b) Variável H_TDH_U_L2_N

Figura 5.2: Comparação, para dois dos modelos, das previsões face aos valores reais.

5.1.2 Resultados da Deteção de Anomalias

Ao nível da deteção de anomalias, como já foi exposto na Secção 2.1.4, podem existir anomalias de diferentes naturezas. Do ponto de vista da análise de consumos energéticos e outras variáveis relacionadas num equipamento, é de elevada importância conseguir fazer a distinção entre elas: pode-se ter uma anomalia pontual associada ou a uma quebra instantânea ou um pico de consumo do equipamento, que rapidamente volta ao seu estado normal, ou então uma anomalia sequencial. Esta última pode estar relacionada com períodos prolongados de défice energético ou consumos anormalmente elevados. É então importante testar a capacidade e versatilidade dos modelos no que toca à identificação destes dois tipos de anomalias.

Para a realização destes testes, treinaram-se e validaram-se os modelos com os tais 70% e 20% do conjunto de dados (referidos no início desta secção), e nos restantes 10% introduziram-se manualmente um conjunto de anomalias, pontuais e sequenciais, como se pode ver na Figura 5.3.



(a) Anomalias pontuais (marcadas com um X) (b) Anomalias sequenciais

Figura 5.3: Exemplos com os dois tipos de anomalias aplicadas a um conjunto de dados de teste.

No caso da Figura 5.3a tem-se um conjunto de anomalias pontuais, as quais tanto podem ser zeros (quebras momentâneas de energia) ou valores excessivamente altos (picos de consumo). Por outro lado, na Figura 5.3b, tem-se um conjunto de pontos consecutivos, de diferentes naturezas e situados em partes distintas dos dados, a simularem anomalias sequenciais.

Para avaliar os resultados dos modelos segundo indicadores quantitativos, foram usadas matrizes de confusão (*confusion matrices*), cuja representação esquemática base se encontra na Figura 5.4, e o MCC (*Matthew's Correlation Coefficient*) cuja fórmula está apresentada na Equação 5.4.

$$\mathrm{MCC} = \frac{\mathrm{TN} \times \mathrm{TP} - \mathrm{FN} \times \mathrm{FP}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}} \tag{5.4}$$

Uma matriz de confusão constitui uma das principais formas de avaliar problemas de classificação, como é o caso da deteção de anomalias. Comparando os pontos do conjunto

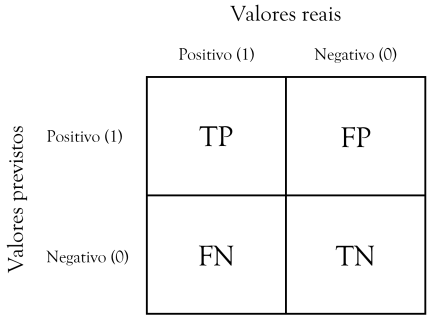


Figura 5.4: Estrutura base de uma matriz de confusão.

de dados de teste definidos como normais (os zeros) com os pontos considerados anomalias (os uns) pelo modelo classificativo, define 4 grandezas fundamentais:

- **TP -** ***True Positives*** - correspondem aos pontos anómalos que o modelo classificou corretamente como anomalias, representados no 1.º quadrante da matriz;
- **FP -** ***False Positives*** - correspondem aos pontos normais que o modelo classificou incorretamente como anómalos, representados no 2º quadrante da matriz;
- **FN -** ***False Negatives*** - correspondem aos pontos anómalos que o modelo classificou incorretamente como normais, representados no 3º quadrante da matriz;
- **TN -** ***True Negatives*** - correspondem aos pontos normais que o modelo classificou corretamente como normais, representados no 4º quadrante da matriz.

Através da definição destes 4 valores, é possível proceder ao cálculo de um indicador quantitativo mais geral, o MCC. Este permite avaliar a capacidade global do modelo identificar corretamente anomalias e pontos normais, ponderando na sua fórmula as 4 grandezas definidas na matriz de confusão. Um MCC próximo de 0 revela que as previsões do modelo e consequente deteção de anomalias ocorreram de forma quase aleatória, ao passo que valores próximos de 1 revelam excelentes capacidades do modelo para prever e detetar anomalias.

Para se proceder à definição do que era e não era anomalia, como foi mencionado na Secção 4.3.3, foi usado o RMSE da validação do treino dos modelos multiplicado por um valor α. Para obter os melhores resultados e desempenhos possíveis ao nível da deteção de anomalias, procedeu-se para cada variável a um processo de busca pelo valor de α que melhor se adequava e que maximizava o MCC. Um fluxograma com a descrição do processo iterativo está representado na Figura 5.5.

Após a definição do conjunto de dados de treino e de teste do modelo, entra-se num processo iterativo principal. Ao longo de cada iteração principal, novas anomalias são definidas (*i.e.*, anomalias em diferentes pontos do conjunto de dados de teste). Por sua vez, por cada iteração principal tem-se também outro processo iterativo interno, no qual são estudados valores para α e de onde sai o melhor valor para cada iteração principal. No fim, após armazenados os valores de α para cada iteração principal, é então escolhido o α final que será usado para detetar as anomalias.

Nas Tabelas 5.2 e 5.3 são apresentados, para cada modelo, os indicadores obtidos para a deteção de anomalias pontuais e sequenciais, respetivamente, que incluem todas as grandezas associadas à matriz de confusão e que permitem o posterior cálculo do MCC.

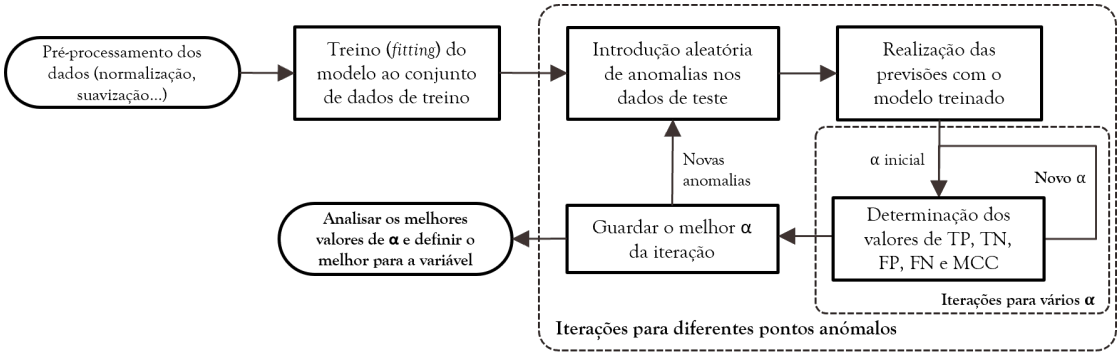


Figura 5.5: Fluxograma do processo de escolha do α para deteção de anomalias.

Tabela 5.2: Resultados obtidos para a deteção de anomalias pontuais.

Variável	α	**TP**	**TN**	**FP**	**FN**	**MCC**
C_phi_L3	0.624	94	946	56	4	0.7531
F	1.771	98	1000	2	2	0.9780
H_TDH_I_Ł3_N	2.153	96	1000	2	4	0.9668
H_TDH_U_L2_N	1.764	100	993	9	0	0.9535
I_SUM	1.989	91	992	10	9	0.8960
P_SUM	1.014	98	939	63	2	0.7459
ReacEc_L1	2.761	98	912	90	2	0.6798
ReacEc_L3	8.577	97	910	92	3	0.6692
RealE_SUM	5.796	98	833	169	2	0.5439
U_L1_N	0.864	92	937	65	8	0.7027

Tabela 5.3: Resultados obtidos para a deteção de anomalias sequenciais.

Variável	α	**TP**	**TN**	**FP**	**FN**	**MCC**
C_phi_L3	2.551	92	994	8	8	0.9120
F	1.312	99	997	5	1	0.9678
H_TDH_I_L3_N	1.486	100	993	9	0	0.9535
H_TDH_U_L2_N	1.776	98	993	9	2	0.9420
I_SUM	1.428	90	1001	2	9	0.9377
P_SUM	2.881	89	1002	0	11	0.9383
ReacEc_L1	0.010	85	984	29	4	0.8289
ReacEc_L3	0.310	73	989	13	27	0.7678
RealE_SUM	0.392	75	970	32	25	0.6966
U_L1_N	0.674	88	991	16	7	0.8741

Começando pelos resultados relativos às anomalias pontuais, apresentados na Tabela 5.2. Como se pode verificar, os resultados são na sua globalidade satisfatórios, com o MCC para todas as variáveis estudadas a ser superior a 0.5. Variáveis como a F ou H_TDH_I_L3_N possuem um valor de MCC muito próximo de 1, com um número de TPs e TNs muito elevado, e muito poucas falhas ao nível da classificação dos pontos. Inclusivamente, para a variável H_TDH_I_L3_N verifica-se que o modelo conseguiu com sucesso identificar os 100 pontos anómalos colocados manualmente. Isso indica

uma excelente capacidade dos modelos realizarem previsões e identificarem corretamente diferenças relacionadas com anomalias.

Por outro lado, variáveis como a ReacEc_L3 ou a RealE_SUM apresentam já resultados menos satisfatórios de MCC, muito devido ao facto da existência de um número elevado de FPs. De facto, analisando todas as variáveis, verifica-se que regra geral há muitos mais FPs do que FNs. Apesar de corresponderem a falhas ao nível da classificação, é preferível que isto aconteça do que o inverso. Pensando já na implementação em ambiente industrial, é melhor o agente inteligente gerar mais avisos (mesmo que as anomalias identificadas sejam pontos normais), ou seja, gerar mais FPs, do que considerar pontos anómalos como sendo normais e ter mais FNs.

Passando para a análise ao nível das anomalias sequenciais (Tabela 5.3), verifica-se à semelhança das pontuais resultados para o MCC em muitas das variáveis acima dos 0.9, com grande parte dos modelos a revelarem uma boa capacidade para identificarem períodos consecutivos de anomalias, e uma elevada taxa de TPs e TNs face a FPs e FNs. Apenas três modelos (ReacEc_L1, ReacEc_L3 e RealE_SUM) revelam valores inferiores deste indicador, sendo que neste caso já há um equilíbrio ao nível dos FNs e FPs, o que é mais preocupante relativamente ao que se verificou com as anomalias pontuais. Todavia, de um modo geral verifica-se novamente uma boa capacidade na deteção de anomalias, neste caso sequenciais.

Por fim, é importante dar ênfase também aos valores de α resultantes do processo de otimização exposto na Figura 5.5. Analisando as Tabelas 5.2 e 5.3, verifica-se que para cada variável os valores de α variam ainda consideravelmente (no caso da ReacEc_L3, o mais extremo, tem-se um α de cerca de 8.6 para as pontuais e de 0.3 para as contínuas). Isto significa que os limites a partir dos quais o modelo considera um ponto como uma anomalia ($\alpha \times$ RMSE) variam bastante. Torna-se então necessária, no caso de variáveis onde isto se verifica, a escolha ponderada e adequada de um valor de α que consiga garantir um bom desempenho para os dois tipos de anomalias.

De um modo geral, é possível concluir que, tendo em conta os pormenores referidos acima, para além das boas capacidades dos modelos construidos ao nível da realização de previsões, também a metodologia usada para a deteção de anomalias através das previsões se revela adequada.

5.1.3 Resultados da Utilização de AutoML

Como foi sendo referido ao longo do documento, a construção dos modelos preditivos foi realizada com o auxílio do AutoML. Assim, para cada variável definiram-se um conjunto de parâmetros a iterar, e a partir de uma arquitetura base houve a construção automatizada de modelos adaptados ao comportamento de cada variável. Isso resultou em modelos com diferentes características (número de camadas, nodos por camada, etc.).

Para validar o contributo que o AutoML trouxe à qualidade dos modelos em termos das tarefas preditivas e de deteção de anomalias, foram realizadas comparações com duas bases distintas: uma regressão linear, o modelo regressivo mais simples, e um *autoencoder* com uma arquitetura pré-definida. Este possui 2 camadas LSTM iniciais e 5 camadas densas no *encoder*, com um número de nodos de 48, 32, 24, 18, 14, 12 e 10. A comparação das previsões baseou-se ao nível do MSE dos vários modelos, com cada um a fazer previsões sobre os mesmos dados de teste. Já a comparação da capacidade de deteção de anomalias foi feita analisando os valores do MCC sobre o mesmo conjunto de

dados de teste com as mesmas anomalias introduzidas. Adicionalmente, para os modelos de regressão linear e *autoencoder* fixo, usou-se um α constante e igual a 1.5. Para os modelos construídos com AutoML já se procedeu também à otimização do valor de α como exposto na secção anterior. No que toca à deteção de anomalias, no Apêndice A, Secção D.2 são apresentados resultados mais detalhados para todos os tipos de modelos aplicados a todas as variáveis.

Os resultados das comparações são apresentados nas Tabelas 5.4 e 5.5 para as previsões e deteção de anomalias, respetivamente. É de salientar que para 3 das variáveis (ReacEc_L1, ReacEc_L3 e RealE_SUM) não são demonstrados resultados ao nível da tabela das previsões. Devido ao seu comportamento próximo do linear, uma regressão linear resultou num melhor modelo em comparação com o melhor *autoencoder* obtido com AutoML.

Tabela 5.4: Comparação dos valores de MSE para os vários modelos preditivos.

Variável	**AE (AutoML)**	**Regressão linear**		**AE (fixo)**	
		MSE	Melhoria	MSE	Melhoria
C_phi_L3	0.000164	0.000386	-46.4%	0.000207	-20.8%
F	0.000205	0.002522	-89.7%	0.000260	-21.2%
H_TDH_I_L3_N	0.000126	0.001730	-87.7%	0.000212	-40.6%
H_TDH_U_L2_N	0.000298	0.005050	-92.3%	0.000391	-23.8%
I_SUM	0.000452	0.008532	-93.4%	0.000516	-12.4%
P_SUM	0.000678	0.013203	-95.0%	0.000657	+3.2%
U_L1_N	8.86e-05	0.001127	-81.4%	0.000210	-57.8%

Tabela 5.5: Comparação dos valores de MCC obtidos para os vários modelos, relativos às anomalias pontuais. De salientar que para os dois modelos base se usou um α fixo de 1.5.

Variável	**AE (AutoML)**		**Regressão Linear**		**AE (fixo)**	
	α	**MCC**	**MCC**	**Melhoria**	**MCC**	**Melhoria**
C_phi_L3	0.6920	0.7617	0.6758	+2.4%	0.6458	+17.9%
F	2.0711	0.8148	0.6940	+17.4%	0.7431	+9.6%
H_TDH_I_Ł3_N	2.0711	0.9354	0.6998	+33.7%	0.7337	+27.5%
H_TDH_U_L2_N	1.8088	0.8870	0.8816	+0.6%	0.7531	+17.8%
I_SUM	2.1536	0.9153	0.7554	+21.2%	0.6290	+41.4%
P_SUM	1.2167	0.7711	0.6848	+12.6%	0.7525	+2.5%
ReacEc_L1 *	1.6064	0.6732	0.6732	+0.0%	0.3171	+112.3%
ReacEc_L3 *	0.9544	0.6503	0.6168	+5.4%	0.3976	+63.6%
RealE_SUM *	0.6995	0.5205	0.0188	+3620.7%	0.2930	+77.6%
U_L1_N	0.9844	0.8184	0.5873	+39.3%	0.8180	+0.0%

* Regressão linear com melhor desempenho face ao melhor modelo obtido com AutoML

Analisando os resultados das previsões (Tabela 5.4), verifica-se desde logo que quer os modelos obtidos recorrendo a AutoML quer mesmo o *autoencoder* padrão conseguem melhorias significativas ao nível das previsões face ao modelo regressivo mais simples, a regressão linear. Os modelos construídos com AutoML atingem na sua grande maioria reduções do MSE na ordem dos 90%, provando assim as suas vantagens e capacidades

ao nível de tarefas de regressão. Olhando agora para a comparação face ao *autoencoder* fixo, verificam-se também melhorias em certos casos significativas: na variável H_TDH_I_L3_N há uma redução à volta dos 40% no MSE, e na U_L1_N de cerca de 60%. Contudo, verifica-se que para a variável P_SUM, o modelo iterado revela desempenhos ligeiramente piores em termos do MSE. Isso pode dever-se ao facto do número de iterações do AutoML não ter sido o suficiente para otimizar ao máximo o modelo.

Estas melhorias na capacidade de previsão também se refletem na capacidade de deteção de anomalias (Tabela 5.5). Como se pode verificar, em todas as variáveis os modelos otimizados demonstram um valor de MCC mais elevado. Isso deve-se quer ao facto dos modelos terem melhores capacidades de previsão, quer ao processo de otimização do valor de α. Exemplos claros da influência do valor de α são os das variáveis com comportamentos mais lineares, ReacEc_L1, ReacEc_L3 e RealE_SUM. Como já referido anteriormente, a construção com recurso ao AutoML levou a que os modelos finais fossem de regressões lineares, e não de *autoencoders* como os restantes. Daí que comparando as regressões lineares com os *autoencoders*, os últimos revelem desempenhos piores ao nível do MCC. No caso da variável ReacEc_L1, como o α otimizado é muito próximo de 1.5, entre o modelo ótimo e a regressão linear o MCC é o mesmo. Por outro lado, nas outras duas variáveis, a otimização do α revela claras melhorias, nomeadamente na variável RealE_SUM. Neste último caso, com o modelo e α otimizados, houve um aumento em cerca de 36x do MCC face ao obtido com a regressão linear.

Olhando para o todo, pode-se concluir que para as previsões, a utilização de *autoencoders* traz já melhorias ao nível da capacidade preditiva dos modelos face a modelos simples de regressões lineares. Por sua vez, a utilização do AutoML consegue extrair ainda melhores desempenhos através da construção de modelos adaptados ao comportamento de cada variável. Isso reflete-se também ao nível da deteção de anomalias, com os modelos otimizados a revelarem também superior capacidade nesse sentido. Por fim, o processo de otimização do valor α também se revela adequado, aumentando ainda mais a capacidade de deteção de anomalias dos modelos já otimizados.

5.2 Implementação da Arquitetura

Ao longo dos Capítulos 3 e 4 foram expostas a arquitetura concetual proposta e os diferentes passos que levaram à validação em ambiente simulado dos vários blocos da mesma. Estes culminaram na sua implementação num caso de estudo Bosch. Esta secção demonstra os resultados obtidos ao nível dessa implementação prática: comprova-se inicialmente a passagem dos dados provenientes do compressor para a plataforma do C2E; depois, a chegada desses dados ao InfluxDB via cliente SSE; e por fim a capacidade do agente inteligente recolher e enviar novos dados também para o InfluxDB, com respetiva monitorização em tempo real através do Grafana.

5.2.1 Resultados da Interface Físico-Digital

Nesta secção é exposta e avaliada a passagem da informação do nível físico da arquitetura para o nível digital. Como já foi referido na Secção 4.4, para promover a recolha de dados do compressor foi utilizado um Raspberry Pi. Este microcontrolador, por sua vez, remetia os dados para a plataforma C2E, enviando um documento JSON devidamente formatado para o Eclipse Hono. Na Figura 5.6 pode ser visualizado o conteúdo da mensagem, a

qual inclui todas as variáveis que o analisador de energia monitoriza, e com a mesma estrutura daquela referida na Secção 4.1.1 aquando a implementação simulada do C2E.

Figura 5.6: Mensagens enviadas pelo *gateway* para o Eclipse Hono.

Após a informação ser recebida pelo Hono é automaticamente remetida para o Ditto, onde há a atualização do DT relativo neste caso ao compressor. Na Figura 5.7 é possível verificar o estado atual do DT do compressor, acedendo ao URL da API do Ditto. Já na Figura 5.8 é possível visualizar a nova informação gerada pelo Ditto.

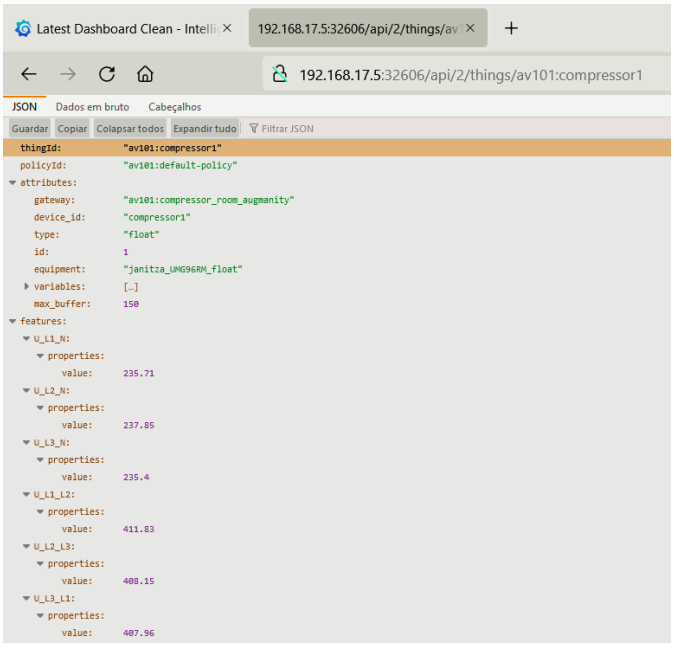


Figura 5.7: *Endpoint* relativo ao estado do DT do compressor.

Após a atualização do estado do *digital twin* do compressor, é necessário que essa informação seja armazenada no InfluxDB. Para a informação poder chegar à base de dados, tem-se um serviço com um cliente SSE desenvolvido em Python implementado no servidor. Sempre que há uma atualização ao nível do DT, a API do Ditto gera um novo evento, sendo enviado via HTTP para o cliente SSE um JSON com toda a nova informação devidamente identificada. Essa informação contém o conteúdo do JSON gerado pelo Ditto, demonstrado na Figura 5.8. Após isso, a informação é toda processada

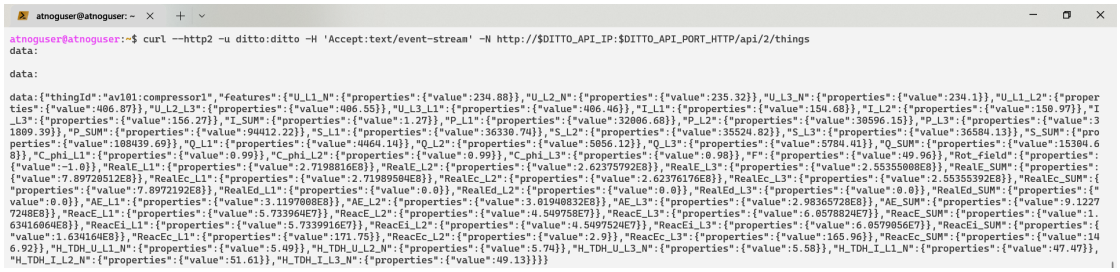


Figura 5.8: *Screenshot* da nova informação notificada pelo Eclipse Ditto.

no mesmo serviço do SSE, e com um cliente InfluxDB inicializado neste *script*, os novos dados são enviados para a base de dados, a qual possui a organização exposta na Figura 5.9.

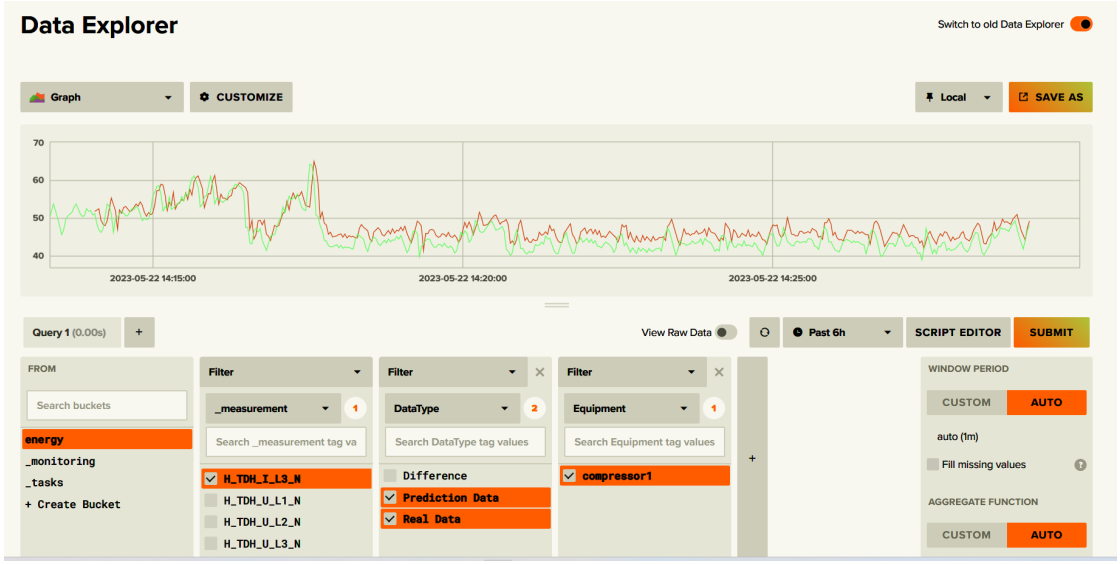


Figura 5.9: Organização da base de dados implementada no caso de estudo.

De um modo muito resumido, cada "_measurement" corresponde exatamente a cada variável a ser monitorizada pelo analisador de energia. Dessas variáveis, 10 são não só monitorizadas como também processadas via algoritmos, e portanto o "DataType" define para essas variáveis os seus valores reais, as previsões efetuadas pelos modelos e ainda a diferença entre esses dois, usada na tarefa de deteção de anomalias. Por fim, no caso de estudo apenas havia um equipamento, o "compressor1", o qual é definido conforme a *tag* "Equipment".

A partir do InfluxDB, os dados são remetidos para o agente inteligente para processamento, que por sua vez também envia nova informação para a base de dados. Simultaneamente, tem-se a conexão com o Grafana, que permite a sua monitorização.

5.2.2 Resultados da Implementação do Agente Inteligente

Com os dados a chegar à base de dados em tempo real, já se torna possível fazer a implementação do agente inteligente, com a geração das previsões e a deteção de anomalias, e

a interface gráfica em Grafana para visualização dos dados.

No caso de estudo do compressor, os dados são recolhidos a cada 2 segundos, e portanto injetados a esta taxa na base de dados. É essencial garantir que o agente inteligente tem a capacidade de, dentro deste curto intervalo de tempo, fazer a deteção de anomalias e ainda as previsões futuras.

Havendo a necessidade de separar os serviços por duas máquinas distintas na implementação prática (como já explicado na Secção 4.4), o InfluxDB e o agente inteligente ficaram em máquinas separadas. Isso levou a que houvesse maiores latências ao nível dos pedidos de leitura e escrita enviados constantemente pelo serviço de previsões e deteção de anomalias. Assim, para garantir que o agente inteligente conseguia realizar ciclos de previsões e deteção de anomalias dentro do tempo limite imposto, foi realizado um pequeno teste. Para 10 novas injeções de dados, foi realizada uma comparação do tempo necessário para realizar um ciclo de deteção de anomalias + previsões num ambiente simulado (todos os serviços a correr na mesma máquina recorrendo a *containers* Docker) e na implementação prática na Bosch (com o serviço do InfluxDB no servidor, mas o agente inteligente a correr na máquina externa). As características da máquina onde o agente inteligente foi executado encontram-se na Tabela 5.6, com os resultados deste teste expostos na Tabela 5.7.

Tabela 5.6: Características base da máquina usada para testes.

Processador	**Base clock speed**	**RAM**	**Armazenamento**	**Gráfica**
AMD Ryzen 7 5800H	3.20 GHz	16.0 GB	SSD 1TB	NVIDIA GeForce RTX 3060

Tabela 5.7: Comparação do desempenho do agente inteligente em ambiente simulado e real.

Injeção	**Simulação**	**Real**
1	0.4989 s	1.4416 s
2	0.4713 s	1.2707 s
3	0.4807 s	1.2958 s
4	0.4607 s	1.3496 s
5	0.4618 s	1.4171 s
6	0.4539 s	1.3037 s
7	0.5849 s	1.4845 s
8	0.4849 s	1.3819 s
9	0.5149 s	1.3225 s
10	0.4453 s	1.3105 s

Como se pode verificar pelos resultados, verifica-se claramente que no nível simulado, onde todos os serviços desde o cliente SSE até à base de dados correm na mesma máquina, os tempos de um ciclo de previsões + deteção de anomalias são muito inferiores. Em média, consegue-se uma redução de cerca de 2.5 a 3 vezes no tempo de execução face à implementação prática, na qual se tinha o serviço do InfluxDB a correr normalmente no servidor, mas os serviços do agente inteligente numa máquina separada. Um dos grandes motivos para esta diferença surge exatamente da necessidade de existência de troca de

informação entre as duas máquinas: é necessário recolher informação da base de dados no início das previsões, e após processamento enviar nova informação de volta para o InfluxDB. Estes resultados também levam a crer que se se implementasse a previsão de todas as variáveis a serem monitorizadas, um ciclo de previsões e deteção de anomalias não conseguiria ser realizado, a um nível prático, no espaço temporal imposto de 2 segundos.

Por fim, resta apenas a demonstração da interface gráfica construída. Como já dito, foi desenvolvida uma interface em Grafana para promover a fácil monitorização das variáveis do compressor, das previsões efetuadas no processamento dos dados e ainda a geração dos alarmes. A interface implementada pode ser visualizada na Figura 5.10.

(a) Interface de monitorização individual das variáveis

(b) Interface de monitorização gráfica das variáveis em simultâneo

Figura 5.10: *Screenshots* da interface gráfica desenvolvida em Grafana.

Com esta interface, é possível ter-se:

- Uma monitorização gráfica das previsões realizadas pelos modelos face aos valores reais que vêm do compressor, que permite também de forma visual fazer uma

inspeção a possíveis anomalias ou diferenças maiores entre previsões e valores reais;

- Uma monitorização sob a forma de tabelas dos últimos valores para as previsões, realidade e ainda diferenças entre os dois;
- Rápida monitorização em tempo real dos últimos valores verificados, assim como a possibilidade de ter sempre presente, para cada variável, o valor limite a partir do qual se tem uma anomalia;
- Uma secção relativa à alarmística, onde se pode visualizar o estado de diversos alarmes: um relativo a uma diferença anormal entre previsão e valor real, outro relativo a pontos consecutivos anómalos (representativa de uma anomalia sequencial) e outro relativo a anomalias cujo valor da variável é nulo;
- Há ainda uma secção em baixo (Figura 5.10b) que permite uma monitorização global do comportamento de todas as variáveis, com gráficos para cada uma delas e tabelas interativas relativas às mesmas.

Idealmente, juntamente com a interface do Grafana, dever-se-ia também ter notificações automatizadas para os utilizadores via Telegram e emails a serem enviados constantemente. Como já referido anteriormente, as características atuais da rede 5G da Bosch, que é fechada, impossibilitaram a validação experimental destas duas formas de interação com o utilizador. Isso porque não é possível estabelecer-se uma conexão com os servidores externos do Telegram e do Gmail para enviar pedidos e receber respostas.

Capítulo 6

Conclusões

Este documento apresenta o trabalho desenvolvido ao longo do semestre no âmbito do Projeto *Augmanity*, e em colaboração com a Bosch Termotecnologia Aveiro. Tendo em linha de conta os objetivos deste projeto, o trabalho envolveu a integração de um agente inteligente numa arquitetura de recolha e monitorização de dados do chão de fábrica. O objetivo final era conseguir implementar o agente inteligente, dotado de algoritmos ML, para realizar previsões e deteção de anomalias ao nível de consumos energéticos. Foi possível, em chão de fábrica Bosch, validar grande parte da arquitetura concetualizada, através da aplicação a um caso de estudo existente relativo a um compressor industrial.

Ao nível físico, recorreu-se a um *gateway* (sob a forma de um Raspberry Pi) para promover uma recolha constante de dados a partir de um compressor industrial. Com um analisador de energia ligado ao compressor, foi possível efetuar uma constante monitorização de cerca de 60 variáveis relativas a consumos de energia. O *gateway*, a receber essas informações, garantiu também a passagem da informação da camada física para a digital, através da conexão ao C2E.

Neste nível digital, o uso do C2E, que integra os serviços do Eclispe Hono e Eclipse Ditto, do InfluxDB e Grafana garantiu que a solução desenvolvida fosse de baixo custo, com todos estes *softwares* a serem *open-source*. Tal como proposto nos objetivos, foi possível recorrer à monitorização do equipamento do caso de estudo com recurso ao seu *Digital Twin*, deixando-se espaço e possibilidades para a criação de mais DTs e monitorização de mais equipamentos. A utilização de *autoencoders* como algoritmos para realizar previsões e detetar anomalias revelou-se acertada, com os modelos a apresentarem indicadores muito positivos em ambas as tarefas. Apesar das dificuldades iniciais encontradas na implementação prática, o agente revelou-se capaz de interagir com a base de dados e restantes serviços de forma adequada, realizando as previsões e deteção de anomalias dentro do intervalo de tempo estipulado. Ao nível da construção dos modelos, verificou-se que o uso de AutoML como ferramenta de otimização garantiu melhorias substanciais ao nível do seu desempenho na realização das previsões. Por fim, a metodologia implementada para a deteção de anomalias, com limites adaptados a cada variável, revelou também bons desempenhos, com capacidades comprovadas de deteção de diferentes tipos de anomalias.

Por fim, ao nível das interações com o utilizador, implementou-se ao nível prático uma interface interativa em Grafana. Nesta, foi possível realizar a monitorização dos dados do compressor, dos dados resultantes do processamento do agente inteligente e ainda a geração automatizada de alguns alarmes. Apesar de apenas validado em ambiente

simulado, também foi possível comprovar a versatilidade do agente, com a implementação de outras possibilidades de interação com o utilizador: geração de relatórios e envio de emails com um resumo dos acontecimentos do último turno, e envio de notificações via Telegram com informações relativas ao processo de deteção de anomalias.

Contudo, apesar de todo o trabalho desenvolvido e testado, são de destacar alguns pontos menos positivos: devido às limitações atuais da rede 5G da Bosch, não foi possível explorar todas as potencialidades do agente inteligente desenvolvido a um nível prático; adicionalmente, os problemas encontrados no servidor do caso de estudo levaram a que houvesse bastante tempo perdido na tentativa da sua resolução. Isso, aliado ao facto da rede 5G do caso de estudo estar atualmente isolada da rede de produção da Bosch, levou a que não houvesse possibilidades de se desenvolver apropriadamente a implementação do Nexeed MES, a qual foi proposta na solução concetual.

Pode-se dizer que, na sua globalidade, a implementação do agente inteligente na arquitetura proposta foi conseguida com sucesso. Porém, o trabalho desenvolvido deixa em aberto um enorme conjunto de possibilidades futuras. Seguem-se algumas propostas para posterior desenvolvimento deste trabalho:

- **Implementação do Nexeed MES**: tal como proposto no Capítulo 3, o processamento conjunto de dados dos equipamentos com dados da produção constitui uma mais-valia, e tornaria o agente inteligente mais apto e conectado ao próprio chão de fábrica. O agente poderia estabelecer certos paralelismos entre fenómenos na produção e fenómenos ao nível dos consumos dos equipamentos, ter maior variedade ao nível dos alarmes, entre outras funcionalidades;
- **Capacidade de atuação nos sistemas físicos**: da mesma forma que a informação vem dos equipamentos físicos, passa pela camada digital intermédia, é processada e finalmente chega ao utilizador, do ponto de vista industrial seria também muito importante ter o fluxo contrário de dados. Quando o agente inteligente detetasse uma anomalia de maior severidade, enviaria essa informação de volta para o C2E. A alteração do estado do DT poderia ser passada de volta para o Hono, e ser transmitida para o equipamento físico. Assim, ter-se-ia uma capacidade autónoma do sistema inteligente de atuar também sobre o nível físico da arquitetura;
- **Aumentar versatilidade dos algoritmos**: neste trabalho, a utilização de *autoencoders* com AutoML foi validada com sucesso. A partir de um grupo de dados prévios, foi possível fazer previsões para o ponto temporal seguinte e promover a deteção de anomalias sobre essas previsões. Com este processo a ser constantemente repetido, sempre que nova informação era injetada havia uma nova previsão futura. Contudo, seria também interessante testar os *autoencoders* em conjunto com o AutoML para realizar operações de previsão a longo prazo. Isso poderia levar a que o agente inteligente implementasse também operações de manutenção preditiva, por exemplo.

Referências

[1] B. Chen e J. Wan, "Emerging Trends of ML-based Intelligent Services for Industrial Internet of Things (IIoT)," em *2019 Computing, Communications and IoT Applications (ComComAp)*, IEEE, out. de 2019. DOI: 10.1109/comcomap46287.2019.9018815.

[2] A. Frankó, G. Hollósi, D. Ficzere e P. Varga, "Applied Machine Learning for IIoT and Smart Production - Methods to Improve Production Quality, Safety and Sustainability," *Sensors*, vol. 22, n.º 23, p. 9148, nov. de 2022. DOI: 10.3390/s22239148.

[3] Bosch Termotecnologia Aveiro, *Work in Harmony | Augmented Humanity*, (acedido a 10 de mar. 2023). URL: https://www.augmanity.pt/.

[4] P. M. B. S. Rendeiro, "Solução de Gestão de Energia para a Internet das Coisas na Indústria," Dissertação de Mestrado, Universidade de Aveiro, 2020.

[5] D. J. Camarneiro, "Internet das Coisas na Indústria, e a conectividade na Bosch," Dissertação de Mestrado, Universidade de Aveiro, 2021.

[6] G. A. V. Matos, "Serviços Web para monitorização de consumos energéticos em chão de fábrica, com base nas plataformas Eclipse IoT: Bosch e SCoT," Dissertação de Mestrado, Universidade de Aveiro, 2022.

[7] P. M. R. Oliveira, "Agente Inteligente para Gestão de Eletricidade," Dissertação de Mestrado, Universidade de Aveiro, 2022.

[8] Rober Bosch GmbH, "Bosch sustainability report 2021," 2021, (acedido a 10 de jan. 2023). URL: https://assets.bosch.com/media/global/sustainability/reporting_and_data/2021/bosch-sustainability-report-2021.pdf.

[9] J. Meira, G. Matos, A. Perdigão et al., "Industrial Internet of Things over 5G: A Practical Implementation," *Sensors*, vol. 23, n.º 11, mai. de 2023. DOI: 10.3390/s23115199.

[10] R. S. Peres, A. D. Rocha, A. Coelho e J. B. Oliveira, "A Highly Flexible, Distributed Data Analysis Framework for Industry 4.0 Manufacturing Systems," em *Service Orientation in Holonic and Multi-Agent Manufacturing*, Springer International Publishing, 2017, pp. 373–381. DOI: 10.1007/978-3-319-51100-9_33.

[11] P. de Arquer Fernández, M. Á. F. Fernández, J. L. C. Candás e P. A. Arboleya, "An IoT open source platform for photovoltaic plants supervision," *International Journal of Electrical Power & Energy Systems*, vol. 125, p. 106 540, fev. de 2021. DOI: 10.1016/j.ijepes.2020.106540.

[12] D. Coelho, D. Costa, E. M. Rocha, D. Almeida e J. P. Santos, "Predictive maintenance on sensorized stamping presses by time series segmentation, anomaly detection, and classification algorithms," *Procedia Computer Science*, vol. 200, pp. 1184–1193, 2022. DOI: 10.1016/j.procs.2022.01.318.

[13] P. Pravin, J. Z. M. Tan, K. S. Yap e Z. Wu, "Hyperparameter optimization strategies for machine learning-based stochastic energy efficient scheduling in cyber-physical production systems," *Digital Chemical Engineering*, vol. 4, p. 100 047, set. de 2022. DOI: 10.1016/j.dche.2022.100047.

[14] DB-Engines, *DB-Engines Ranking - popularity ranking of relational DBMS*, (acedido a 10 de jan. 2023). URL: https://db-engines.com/en/ranking/relational+dbms.

[15] DB-Engines, *DB-Engines Ranking per database model category*, (acedido a 10 de jan. 2023). URL: https://db-engines.com/en/ranking_categories.

[16] A. Gomes, V. Lopes, E. Ribeiro et al., "An Empirical Performance Comparison between MySQL and MongoDB on Analytical Queries in the COMEX Database," em *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*, IEEE, jun. de 2021. DOI: 10.23919/cisti52073.2021.9476623.

[17] C. Gyorödi, R. Gyorödi e R. Sotoc, "A Comparative Study of Relational and Non-Relational Database Models in a Web- Based Application," *International Journal of Advanced Computer Science and Applications*, vol. 6, n.º 11, 2015. DOI: 10.14569/ijacsa.2015.061111.

[18] N. Jatana, S. Puri, M. Ahuja, I. Kathuria e D. Gosain, "A Survey and Comparison of Relational and Non-Relational Database," *International Journal of Engineering Research & Technology (IJERT)*, vol. 1, 6 ago. de 2021, ISSN: 2278-01811.

[19] A. Nayak, A. Poriya e D. Poojary, "Type of NOSQL Databases and its Comparison with Relational Databases," *International Journal of Applied Information Systems (IJAIS)*, vol. 5, 4 mar. de 2013, ISSN: 2249-0868.

[20] MongoDB, *NoSQL Vs SQL Databases | MongoDB*, (acedido a 11 de jan. 2023). URL: https://www.mongodb.com/nosql-explained/nosql-vs-sql.

[21] Y. Li e S. Manoharan, "A performance comparison of SQL and NoSQL databases," em *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, IEEE, ago. de 2013. DOI: 10.1109/pacrim.2013.6625441.

[22] W. Kritzinger, M. Karner, G. Traar, J. Henjes e W. Sihn, "Digital Twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, vol. 51, n.º 11, pp. 1016–1022, 2018. DOI: 10.1016/j.ifacol.2018.08.474.

[23] H. X. Nguyen, R. Trestian, D. To e M. Tatipamula, "Digital Twin for 5G and Beyond," *IEEE Communications Magazine*, vol. 59, n.º 2, pp. 10–15, fev. de 2021. DOI: 10.1109/mcom.001.2000343.

[24] F. Pires, A. Cachada, J. Barbosa, A. P. Moreira e P. Leitao, "Digital Twin in Industry 4.0: Technologies, Applications and Challenges," em *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, IEEE, jul. de 2019. DOI: 10.1109/indin41052.2019.8972134.

[25] K. Shaukat, T. M. Alam, S. Luo et al., "A Review of Time-Series Anomaly Detection Techniques: A Step to Future Perspectives," em *Advances in Intelligent Systems and Computing*, Springer International Publishing, 2021, pp. 865–877. DOI: 10.1007/978-3-030-73100-7_60.

[26] A. A. Cook, G. Misirli e Z. Fan, "Anomaly Detection for IoT Time-Series Data: A Survey," *IEEE Internet of Things Journal*, vol. 7, n.º 7, pp. 6481–6494, jul. de 2020. DOI: 10.1109/jiot.2019.2958185.

[27] S. Schmidl, P. Wenig e T. Papenbrock, "Anomaly detection in time series," *Proceedings of the VLDB Endowment*, vol. 15, n.º 9, pp. 1779–1797, mai. de 2022. DOI: 10.14778/3538598.3538602.

[28] K. Choi, J. Yi, C. Park e S. Yoon, "Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines," *IEEE Access*, vol. 9, pp. 120 043–120 065, 2021. DOI: 10.1109/access.2021.3107975.

[29] A. Alsharef, K. Aggarwal, Sonia, M. Kumar e A. Mishra, "Review of ML and AutoML Solutions to Forecast Time-Series Data," *Archives of Computational Methods in Engineering*, vol. 29, n.º 7, pp. 5297–5311, jun. de 2022. DOI: 10.1007/s11831-022-09765-0.

[30] PI.EXCHANGE, *No-Code AutoML solutions: How can they help you innovate with ML?* (acedido a 2 de mar. 2023). URL: https://www.pi.exchange/blog/why-automl-is-key-to-entrepreneurs-innovating-with-ml.

[31] M. Bahri, F. Salutari, A. Putina e M. Sozio, "AutoML: state of the art with a focus on anomaly detection, challenges, and research directions," *International Journal of Data Science and Analytics*, vol. 14, n.º 2, pp. 113–126, fev. de 2022. DOI: 10.1007/s41060-022-00309-0.

[32] L. Zimmer, M. Lindauer e F. Hutter, "Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, n.º 9, pp. 3079–3090, set. de 2021. DOI: 10.1109/tpami.2021.3067763.

[33] Eclipse Foundation, *The Eclipse Foundation*, (acedido a 13 de fev. 2023). URL: https://www.eclipse.org/org/foundation/.

[34] Eclipse Foundation, *Connect, Command & Control IoT devices: Eclipse Hono*, (acedido a 8 de fev. 2023). URL: https://www.eclipse.org/hono/.

[35] Eclipse Foundation, *Eclipse Ditto - open source framework for digital twins in the IoT*, (acedido a 8 de fev. 2023). URL: https://www.eclipse.org/ditto/.

[36] Bosch, *NEXEED – software and services for production and logistics*, Brochure, (acedido a 8 de abr. 2023), 2018. URL: https://assets.bosch.com/media/en/global/products_and_solutions/connected_products_and_services/industy_40/bosch-connected-industry-brochure.pdf.

[37] A. Shojaeinasab, T. Charter, M. Jalayer et al., "Intelligent manufacturing execution systems: A systematic review," *Journal of Manufacturing Systems*, vol. 62, pp. 503–522, jan. de 2022. DOI: 10.1016/j.jmsy.2022.01.004.

[38] S. Jaskó, A. Skrop, T. Holczinger, T. Chován e J. Abonyi, "Development of manufacturing execution systems in accordance with Industry 4.0 requirements: A review of standard- and ontology-based methodologies and tools," *Computers in Industry*, vol. 123, p. 103 300, dez. de 2020. DOI: 10.1016/j.compind.2020.103300.

[39] Bosch, *Nexeed: Welcome to the Smart Factory | Bosch Global*, (acedido a 8 de abr. 2023). URL: https://www.bosch.com/stories/nexeed-smart-factory/.

[40] Influx Data, *InfluxDB Times Series Data Platform | InfluxData*, (acedido a 8 de abr. 2023). URL: https://www.influxdata.com/.

[41] M. Wang, S. Liu, K. Wei et al., "Design of Performance Benchmark for Time Series Databases," em *2020 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*, IEEE, dez. de 2020. DOI: 10.1109/ispa-bdcloud-socialcom-sustaincom51426.2020.00208.

[42] Y. Hao, X. Qin, Y. Chen et al., "TS-Benchmark: A Benchmark for Time Series Databases," em *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, IEEE, abr. de 2021. DOI: 10.1109/icde51399.2021.00057.

[43] DB-Engines, *DB-Engines Ranking - Trend of Time Series DBMS Popularity*, (acedido a 15 de maio 2023), mai. de 2023. URL: https://db-engines.com/en/ranking/time+series+dbms.

[44] *Comparison to SQL - InfluxData Documentation Archive*, (acedido a 10 de abr. 2023). URL: https://archive.docs.influxdata.com/influxdb/v1.2/concepts/crosswalk/.

[45] Grafana Labs, *Grafana: The open observability platform | Grafana Labs*, (acedido a 8 de abr. 2023). URL: https://grafana.com/.

[46] Shivang, *What Is Grafana? Why Use It? Everything You Should Know About It*, (acedido a 8 de abr. 2023). URL: https://scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/.

[47] Grafana Labs, *Success stories and case studies | Grafana Labs*, (acedido a 8 de abr. 2023). URL: https://grafana.com/success/?product=grafana-cloud.

[48] Docker, *Docker Documentation*, (acedido a 17 de fev. 2023). URL: https://docs.docker.com/.

[49] Docker, *What is a Container?* (acedido a 17 de fev. 2023). URL: https://www.docker.com/resources/what-container/.

[50] Docker, *Docker Hub - Build and Ship any Application Anywhere*, (acedido a 17 de maio 2023). URL: https://hub.docker.com/.

[51] Eclipse Foundation, *Take a tour*, (acedido a 7 de abr. 2023). URL: https://www.eclipse.org/packages/packages/cloud2edge/tour/.

[52] abli, *What Server-Sent Events is - and how and when to implement it*, (acedido a 10 de mar. 2023). URL: https://ably.com/topic/server-sent-events.

[53] Arduino, *Software | Arduino*, (acedido a 10 de abr. 2023). URL: https://www.arduino.cc/en/software.

[54] N. Chatuverdi, *Dimensionality Reduction using an Autoencoder in Python*, (acedido a 2 de fev. 2023), jul. de 2021. URL: https://medium.datadriveninvestor.com/dimensionality-reduction-using-an-autoencoder-in-python-bf540bb3f085.

[55] Janitza, *Power Analyser UMG 96RM*, (acedido a 17 de maio 2023). URL: https://www.janitza.com/files/download/datasheets/UMG-96-RM/janitza-db-umg96rm-en.pdf.

[56] Telegram Messenger, *Telegram*, (acedido a 17 de maio 2023). URL: https://telegram.org/.

[57] Lumel, *Converter of Rs-485/Ethernet interface*, (acedido a 17 de maio 2023). URL: https://www.energometrika.ru/product_files/521/PD8_manual_EN.pdf.

[58] raspberrypi.org, *Raspberry Pi 3 Model B+*, (acedido a 17 de maio 2023). URL: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf.

[59] D. Soni, *Dealing with Imbalanced Classes in Machine Learning*, (acedido a 2 de fev. 2023). URL: https://towardsdatascience.com/dealing-with-imbalanced-classes-in-machine-learning-d43d6fa19d2.

[60] IBM, *What is the k-nearest neighbors algorithm?* (acedido a 2 de fev. 2023). URL: https://www.ibm.com/topics/knn.

[61] IBM, *What are neural networks?* (acedido a 2 de fev. 2023). URL: https://www.ibm.com/topics/neural-networks.

[62] D. Rengasamy, M. Jafari, B. Rothwell, X. Chen e G. P. Figueredo, "Deep Learning with Dynamically Weighted Loss Function for Sensor-Based Prognostics and Health Management," *Sensors*, vol. 20, n.º 3, p. 723, jan. de 2020. DOI: 10.3390/s20030723.

[63] Naveen, *Support Vector Machine algorithm for Machine Learning*, (acedido a 2 de fev. 2023). URL: https://www.nomidl.com/machine-learning/support-vector-machine-algorithm-for-machine-learning/.

[64] E. Y. Boateng, J. Otoo e D. A. Abaye, "Basic Tenets of Classification Algorithms K-Nearest-Neighbor, Support Vector Machine, Random Forest and Neural Network: A Review," *Journal of Data Analysis and Information Processing*, vol. 08, n.º 04, pp. 341–357, 2020. DOI: 10.4236/jdaip.2020.84020.

Intentionally blank page.

Apêndice A

Métodos de Deteção de Anomalias

Neste apêndice é dada uma explicação mais aprofundada a alguns dos inúmeros algoritmos que podem ser usados para a deteção de anomalias.

A.1 *K-Means Clustering*

O algoritmo de *K-means clustering* é um dos mais simples e utilizados algoritmos para deteção de anomalias não supervisionados. O método consiste no agrupamento de pontos com características semelhantes em K *clusters*, com o objetivo de identificar padrões e tendências de modo a detetar pontos anómalos. Estes *clusters* vão variando ao longo de várias iterações (quer as suas dimensões, quer a sua localização), até que ocorra convergência no que toca a estes parâmetros. Na Figura A.1 é apresentada uma simples representação gráfica de como funciona este método: a azul, vermelho e amarelo têm-se os *clusters* definidos onde se encontram um conjunto de pontos próximos. Depois, realçados a verde têm-se os pontos que se consideram anomalias, dado a sua grande distância a qualquer um dos *clusters*.

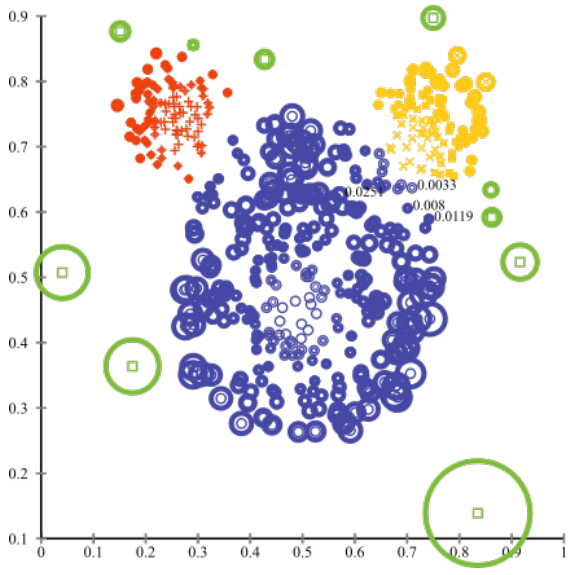


Figura A.1: Representação gráfica do método *K-Means Clustering* (retirado de [59]).

De uma forma muito resumida, de seguida são apresentados os passos fundamentais para a utilização deste algoritmo:

1. **Escolher o número de *clusters*** - o primeiro passo consiste em escolher um número K de *clusters* para promover o agrupamento dos dados. Este valor pode ser difícil de escolher, uma vez que numa série temporal há sempre valores diferentes a ser gerados. Assim, dependendo do período temporal, diferentes valores de K podem ser os ideais;

2. **Inicializar os centroides** - os centroides correspondem ao centro dos *clusters*, havendo portanto K centroides. Há diversas formas de definir quais os centroides, desde uma definição aleatória até à utilização de pequenos algoritmos para essa escolha que permitem por vezes reduzir o número de iterações do método;

3. **Atribuir os dados ao seu *cluster* respetivo** - usando a simples fórmula da distância Euclidiana (Equação A.1 para um caso bidimensional), procede-se ao cálculo da distância entre cada ponto X e cada centroide C associado a cada *cluster*. O ponto pertencerá ao *cluster* onde a distância ao centroide respetivo é a menor;

$$d(X, C) = \sqrt{(x_C - x_X)^2 + (y_C - y_X)^2} \tag{A.1}$$

4. **Reinicializar os centroides** - para cada *cluster*, a localização do novo centroide corresponderá à média de todos os pontos atribuídos a esse mesmo *cluster*

5. **Repetir passos 3 e 4** - depois, repetem-se os passos 3 e 4 até que ocorra convergência nas iterações ou até que se atinja um dado critério de paragem.

No final, as anomalias correspondem a pontos cujas distâncias aos centróides dos respetivos *clusters* sejam significativamente superiores às dos restantes pontos.

A.2 KNN - *K-Nearest Neighbors*

Neste caso, o KNN corresponde a um algoritmo supervisionado para deteção de anomalias, pelo que necessita de um conjunto de dados de treino para posteriormente detetar pontos anómalos. Este método está baseado num pressuposto muito simples, o qual dita que pontos similares podem ser encontrados próximos uns dos outros.

De forma muito resumida, após se escolher um valor para K são avaliadas as distâncias entre o ponto a estudar e os seus K vizinhos mais próximos (esta distância pode ser novamente dada pela distância Euclidiana, cuja fórmula está representada na Equação A.1). Como se pode ver na Figura A.2, onde está um exemplo com K= 7 vizinhos, através da análise das distâncias do ponto a classificar aos seus K vizinhos é possível atribuir a esse ponto uma determinada categoria, que corresponderá àquela cuja distância aos vizinhos se revela menor. Por fim, uma anomalia corresponderá a um ponto cuja distância é muito superior a um determinado valor base, de tal forma que o ponto não partilha características com os restantes pontos de qualquer uma das categorias.

Figura A.2: Representação esquemática do funcionamento do algoritmo de KNN (retirada de [60]).

A.3 LOF - *Local Outlier Factor*

O LOF corresponde a outro exemplo de um método não supervisionado de deteção de anomalias. Este possui algumas noções partilhadas com o método KNN, sendo que usa um conceito muito importante de densidade local: pegando nos k vizinhos mais próximos, a distância entre o ponto e os seus vizinhos é posteriormente usada para determinar a densidade. Quanto menor a densidade relativa a determinado ponto, maior é a probabilidade deste ser classificado como sendo uma anomalia.

Para a compreensão deste método, existem algumas noções que são importantes salientar:

- ***k-distance*** - corresponde à distância entre o ponto a estudar e o seu k^n vizinho;
- ***Reachability Distance* (RD)** - considerando um ponto A e um dos seus vizinhos B, corresponde ao máximo do valor entre a distância de A ao seu vizinho B (k-*distance*(B)) ou a distância direta entre dois pontos A e B (d(A,B)):

$$\mathrm{RD(A,B)} = \max(k\text{-}distance(\mathrm{B}), d(\mathrm{A,B})) \tag{A.2}$$

- ***Local Reachability Density* (LRD)** - corresponde ao inverso do valor médio de todas as RD aos k vizinhos do ponto A (onde é incluído o ponto B), e permite o cálculo da densidade dos k vizinhos em torno de A. Considerando o número de vizinhos de A como sendo dado por N e cada vizinho como sendo dado por k_n tem-se:

$$\mathrm{LRD} = \left(\frac{\sum_{n=1}^{\mathrm{N}} RD(\mathrm{A}, k_n)}{\mathrm{N}} \right)^{-1} \tag{A.3}$$

- ***Local Outlier Factor* (LOF)** - fazendo o cálculo do LRD não só para o ponto A como também para os seus vizinhos pode-se finalmente determinar o valor de LOF como sendo a média, aplicada ao número de pontos da vizinhança, do quociente entre o LRD de todos os seus vizinhos a dividir pelo LRD do próprio ponto A:

$$\text{LOF} = \left(\frac{\sum_{n=1}^{\text{N}} \frac{\text{LRD}(k_n)}{\text{LRD(A)}}}{\text{N}} \right)^{-1} \tag{A.4}$$

Posto isto, quando se obtêm valores de LOF significativamente superiores a 1, pode-se concluir que o ponto considerado se trata de uma anomalia.

A.4 Redes Neuronais

Para promover a deteção de anomalias, na dissertação conduzida por Oliveira [7], a qual está muito focada nos métodos de deteção de anomalias, há duas grandes variações de redes neuronais recorrentes (RNNs), referidas como sendo muito usadas para situações que envolvem deteção de anomalias - LSTM (*Long-Short Term Memory*) e GRU (*Graded Recurrent Units*). Cada uma destas variações é explicada nas secções seguintes (A.4.1 e A.4.2), pelo que antes é dada uma revisão geral às RNNs.

Uma RNN corresponde a um tipo de redes neuronais o qual está adaptado para trabalhar em séries temporais. Enquanto as redes neuronais convencionais são mais adequadas à utilização com pontos e dados que são independentes, este tipo de redes recorrentes está adaptado a pontos que partilhem características e que sejam dependentes entre si (por exemplo, com dependências temporais). Assim, através de alguma capacidade de memória, as RNNs conseguem guardar informações relativas a estas dependências. A arquitetura base de uma rede neuronal é apresentada na Figura A.3.

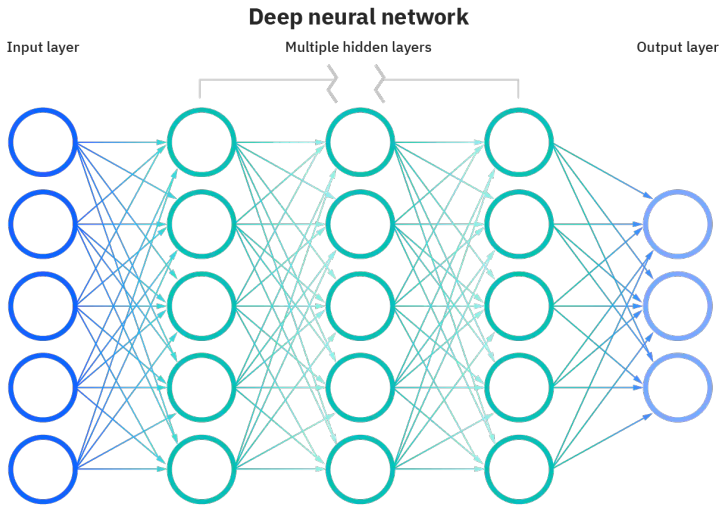


Figura A.3: Representação esquemática de uma rede neuronal (retirada de [61]).

Numa rede neuronal convencional tem-se um conjunto de camadas, distinguindo-se as camadas de *input* e *output* (entrada e saída) e depois um conjunto de camadas intermédias que não são visíveis (*hidden*), tal como se pode ver na Figura A.3. Todas estas camadas encontram-se ligadas entre si, e tentam até certo ponto simular o processo de associação

e raciocínio presente no cérebro humano. Dentro das camadas há um conjunto de nodos, cada um com um determinado peso e um determinado valor base (*threshold*). Quando o *output* de determinado nodo é superior ao seu valor base, então esse nodo é ativado e envia informação para a camada seguinte.

Enquanto numa rede convencional o fluxo de informação acaba por ser unidirecional ao longo das várias camadas, numa RNN existe a possibilidade de haver certos ciclos de *feedback* onde se consegue retirar informação e alguns dos *outputs* prévios para ajudar a determinar o novo *output*. É através desta definição que este tipo de redes neuronais está especialmente adaptada para processar séries temporais.

A.4.1 LSTM - Long-Short Term Memory

Apesar de, como já ter sido dito, as RNNs terem a possibilidade de utilizar alguns *outputs* prévios para influenciar o novo *output*, em séries temporais onde há dependências que abrangem períodos temporais superiores as RNNs não têm a possibilidade de usar outros *outputs* mais "antigos".

Daqui surgiu então esta variação das RNNs, as redes LSTM, as quais estão novamente adaptadas à utilização em séries temporais com a capacidade acrescida de identificar dependências a longo prazo. Para além das várias camadas e elementos tradicionais das redes neuronais (que podem ser vistos na Figura A.3), nas redes LSTM há um novo elemento - célula de memória. Com esta célula, o algoritmo tem capacidade de guardar certas informações durante largos períodos de tempo, e assim ter a capacidade de distinguir certas dependências a longo prazo.

Como se pode ver na Figura A.4, cada célula de memória é composta por três componentes (os *gates*): tem-se um *input gate*, que controla a informação que de facto entra na célula de memória, um *forget gate* que determina toda a informação a remover da célula de memória e o *output gate* que controla a informação que sai da célula de memória. À medida que se vai processando os dados na rede neuronal, estes três *gates* estarão constantemente a filtrar a informação a eliminar e a reter.

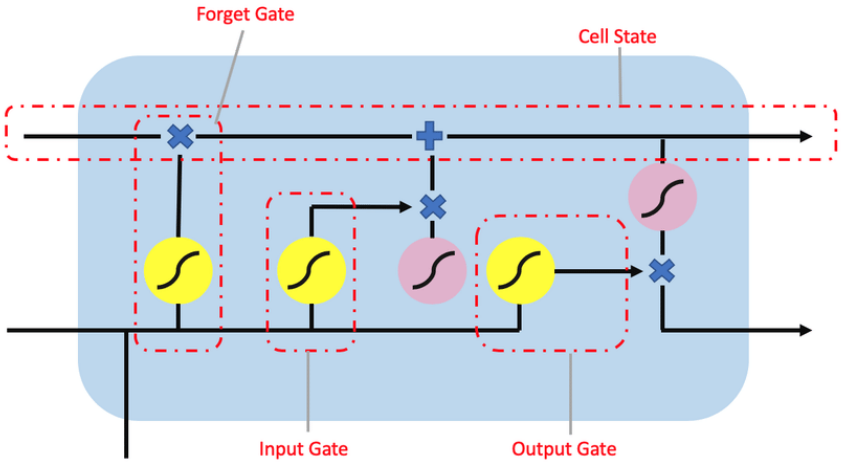


Figura A.4: Representação gráfica da constituição de uma célula de memória LSTM (retirado de [62]).

A.4.2 GRU - Graded Recurrent Units

Corresponde a outra variação das redes neuronais recorrentes, que partilha muitas semelhanças com o LSTM. Tenta à semelhança do método anterior resolver o problema das dependências temporais a longo prazo, patente nas RNNs tradicionais.

Possui também uma estrutura de memória baseada em *gates*, explicados de seguida:

- *Update gate* - é responsável por determinar quanta informação/conhecimento adquirido previamente deve permanecer na memória, funcionando de forma semelhante ao *output gate* do LSTM;
- *Reset gate* - determina a restante informação que deve ser esquecida e eliminada pelo algoritmo, funcionando como uma mistura dos *input* e *forget gates* do LSTM;
- *Current memory gate* - está incorporado no próprio *reset gate*, com o objetivo de tentar reduzir o efeito que a informação passada possui na informação presente que está a ser passada para o futuro.

A.5 SVM - *Support Vector Machine*

Corresponde, à semelhança do KNN, a um algoritmo supervisionado para promover a deteção de anomalias. O objetivo deste algoritmo consiste em encontrar um hiperplano (num problema bidimensional corresponde a uma linha, num tridimensional a um plano) que permite fazer uma distinção e classificação dos pontos num dado conjunto de dados. Havendo a possibilidade de vários hiperplanos a fazer a separação dos dados (como se pode ver na Figura A.5), o objetivo consiste em encontrar aquele que consiga ter a maior distância possível para as várias categorias dos dados.

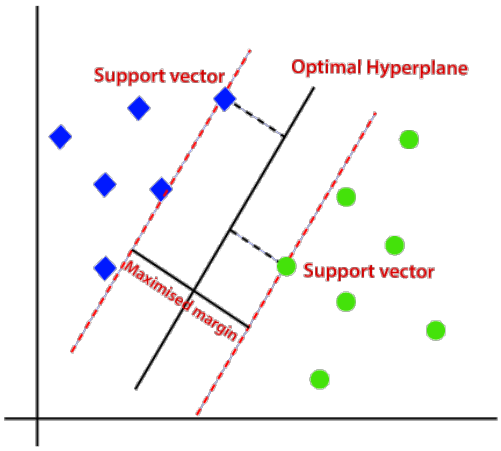


Figura A.5: Representação gráfica, num caso bidimensional, do método SVM (retirado de [63]).

No que toca aos pontos que dão o nome ao algoritmo - *support vectors* - estes correspondem, como se pode ver na Figura A.5, àqueles que se encontram mais próximos do hiperplano. Consequentemente, estes são os pontos que verdadeiramente influenciam a posição do hiperplano, e por isso determinam a divisão dos dados em várias categorias. Uma anomalia corresponderá a um ponto que pertença a determinada categoria, mas

cuja divisão efetuada pelo SVM o atribua a outra das categorias presentes no grupo de dados.

A.6 RF - *Random Forest*

O RF - *Random Forest* - corresponde novamente a um algoritmo de ML supervisionado, que pode ser usado para problemas de classificação e regressão. Este enquadra-se também na categoria do *ensemble learning*, que corresponde à combinação de vários classificadores para a resolução de um problema, com vista a melhorar o desempenho do modelo. Neste caso, o RF resulta da combinação com um conjunto de árvores de decisão.

As árvores de decisão correspondem também a um algoritmo supervisionado de ML aplicável a problemas de regressão e classificação. O seu objetivo é criar um modelo que consiga proceder à previsão da classe da variável objetivo através da aprendizagem de um conjunto de regras simples de previsão, inferidas a partir do conjunto de dados de treino. Este método, como se pode ver na Figura A.6, possui uma estrutura hierarquizada, composta por um nodo de raiz (*root node*), nodos internos, as folhas (*leaf node*) e depois as várias ramificações (*branches*).

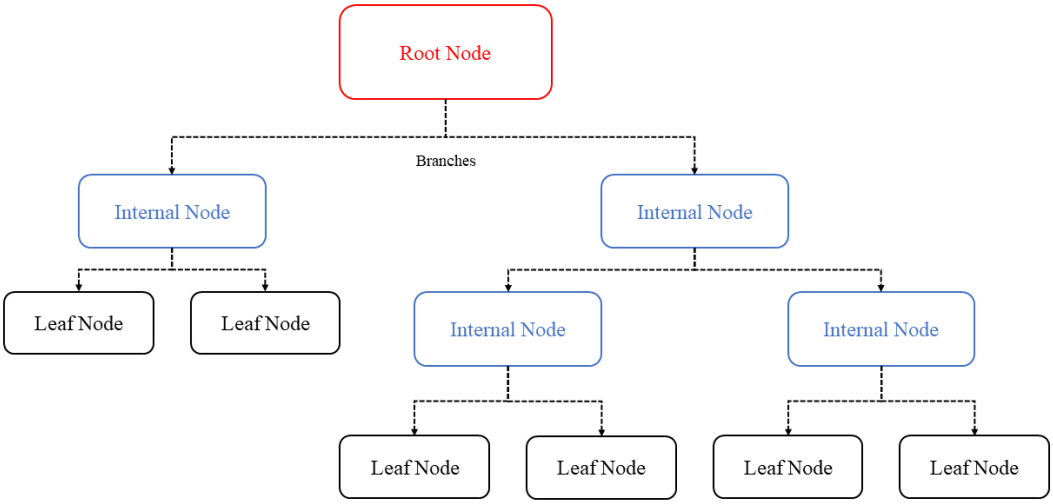


Figura A.6: Representação esquemática do funcionamento de uma árvore de decisão.

O que o algoritmo RF faz é a implementação de um conjunto variado de árvores de decisão. Como se pode ver na Figura A.7, os resultados de cada árvore de decisão são reunidos para fornecer resultados com uma maior precisão.

Assim, cada árvore de decisão vai inicialmente atuar de forma individual, com cada árvore a obter ou uma determinada classificação (no caso de se estar perante um problema de classificação) ou um determinado valor como *output* (num problema de regressão). Posteriormente, o RF irá reunir os resultados de um conjunto de árvores de decisão, e irá ter como resultado ou a classificação mais vezes obtida, ou a média dos *outputs* das várias árvores, dependendo novamente da natureza do problema em questão. Em suma,

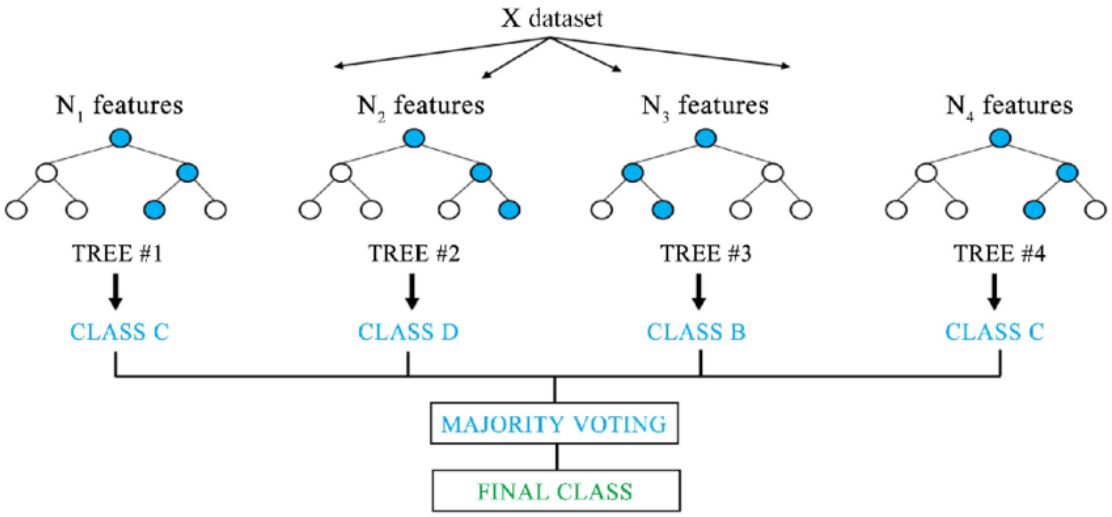


Figura A.7: Representação esquemática do funcionamento do algoritmo RF, neste caso para um problema de classificação (retirado de [64]).

este algoritmo atua sob a premissa de que o agrupamento dos resultados de um conjunto de diferentes árvores de decisão leva a menores erros.

A.7 *Autoencoders*

Os *autoencoders* correspondem a um método não supervisionado de ML e derivado das redes neuronais, o qual é aplicado para uma grande variedade de tarefas, desde *feature-extraction* (extração de características), filtragem de ruído de imagens ou amostras sonoras, até à deteção de anomalias. A arquitetura de funcionamento de um *autoencoder* está representada na Figura A.8. Este é composto por três elementos fundamentais: o *encoder*, o *decoder* e o *bottle-neck*, o ponto central da arquitetura, que na figura é dado como sendo o espaço latente.

De uma forma muito resumida, com recurso ao *encoder* é promovida uma "compressão" dos dados (através da extração das suas principais características) para um espaço latente de menor dimensão e mais compacto, o qual se denomina *bottle-neck*. Por sua vez, com recurso a retropropagação (*backpropagation*), tem-se o *decoder* que é encarregue de fazer uma "descompressão" dos dados, com o objetivo de se ter como *output* um resultado o tão próximo quanto possível ao *input* fornecido inicialmente. Aplicado à deteção de anomalias, o *autoencoder* deve ser treinado apenas com dados associados a comportamentos normais. Depois do treino, na sua fase de aplicação, o modelo tentará usar o *encoder* e *decoder* construídos na fase de treino para tentar ter um *output* o tão próximo possível do *input*, pelo que quando isso não se verifica está-se perante uma anomalia.

A utilização de um *autoencoder* implica a definição de 4 parâmetros fundamentais:

- **Tamanho do espaço latente** - trabalhando de forma semelhante às redes neuronais, este tamanho corresponde ao número de nodos no *bottle-neck*. Quanto menor esse número maior a compressão;
- **Número de camadas** - corresponde basicamente à complexidade do *encoder* e *decoder*. Define o número de camadas necessárias para conseguir primeiro a com-

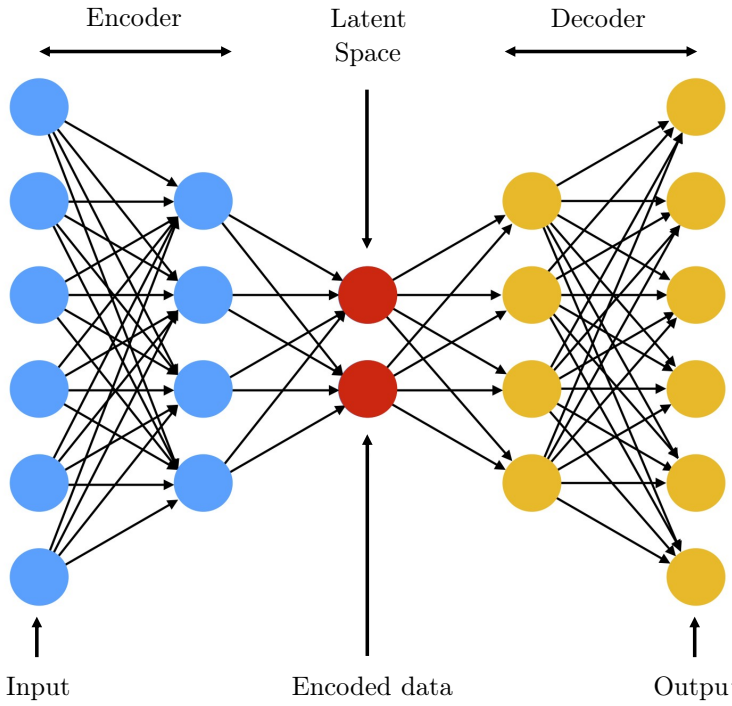


Figura A.8: Arquitetura base de um *autoencoder* (adaptado de [54]).

pressão dos dados no espaço latente e posteriormente a descompressão de volta ao estado o mais próximo possível do inicial;

- **Número de nodos por camada** - à medida que se avança no *encoder*, ter-se-á menos nodos, ou seja, ter-se-á cada vez maior compressão dos dados;
- **Função de perda (*loss function*)** - normalmente usa-se o MSE sendo também definida como sendo o erro de reconstrução. Esta função serve para medir o erro entre os dados originais e os reconstruidos, pelo que com o treino do algoritmo pretende-se minimizar esta função.

Dependendo da aplicação e/ou do seu objetivo, há vários tipos de *autoencoders*: tem-se os *Undercomplete Autoencoders* cujo objetivo final é mesmo a extração das características mais relevantes dos dados; os *Sparse Autoencoders* que possuem flexibilidade para alterar o número de nodos das suas camadas intermédias; os *Convolutional Autoencoders*, muito usados para eliminação do ruído em imagens; ou os *Variational Autoencoders*, cujo *bottle-neck* (ou seja, o espaço latente), é caracterizado segundo uma distribuição probabilística contínua, e tenta resolver problemas dos *autoencoders* tradicionais relacionados com a possível descontinuidade do seu espaço latente.

Intentionally blank page.

Apêndice B

Registo de dispositivos na plataforma Cloud2Edge

Este apêndice serve para expor os diferentes passos que permitem fazer o registo de um determinado equipamento ou dispositivo na plataforma C2E. É de salientar que os passos descritos de seguida referem-se a um registo completamente de raiz de um dispositivo, envolvendo também a criação de uma nova ligação entre o Hono e o Ditto.

1. **Inicializar o ambiente Cloud2Edge** - através de uma nova linha de comandos, deve-se antes de tudo inicializar um conjunto de variáveis correspondentes ao Cloud2Edge, sendo para isso necessário correr as seguintes linhas de comandos:

```
RELEASE=c2e
NS=cloud2edge
./setCloud2EdgeEnv.sh $RELEASE $NS
```

Código B.1: Comandos para inicializar as variáveis do ambiente Cloud2Edge.

2. **Criação de um novo *tenant*** - um *tenant*, dentro do Cloud2Edge, corresponde a uma entidade na qual podem residir um conjunto de dispositivos, com a sua informação a ser guardada e isolada dentro do *tenant* respetivo. O código seguinte representa a criação de um novo *tenant* com o nome "my-tenant" (é de salientar que o "REGISTRY_IP" e o "REGISTRY_PORT_HTTP" correspondem exatamente a variáveis que são inicializadas correndo o código B.1):

```
curl -i -X POST http://${REGISTRY_IP}:${REGISTRY_PORT_HTTP}/v1/
    tenants/my-tenant
```

Código B.2: Criação de um novo *tenant*.

3. **Criação do dispositivo no Hono** - após a criação do *tenant*, deve-se então fazer a associação do dispositivo pretendido ao mesmo, fazendo o seu registo na plataforma do Eclipse Hono. Neste caso, está-se a registar um novo dispositivo no *tenant* "my_tenant", com o nome "my-device-1", e com o *namespace* "org.acme" (vários dispositivos podem partilhar o mesmo *namespace*, pelo que este representa um conjunto de dispositivos com características semelhantes):

```
curl -i -X POST http://${REGISTRY_IP}:${REGISTRY_PORT_HTTP}/v1/
    devices/my-tenant/org.acme:my-device-1
```

Código B.3: Registo de um novo dispositivo.

4. **Definição das credenciais do dispositivo** - as credenciais do dispositivo (ID + palavra-passe) são essenciais para que se possa fazer o envio de informação via MQTT relativa ao dispositivo em questão. Neste caso, para o dispositivo criado anteriormente, o ID registado é "my-auth-id-1" com a palavra-passe a ser "my-password":

```
curl -i -X PUT -H "Content-Type: application/json" --data '[
{
  "type": "hashed-password",
  "auth-id": "my-auth-id-1",
  "secrets": [{
    "pwd-plain": "my-password"
  }]
}]' http://${REGISTRY_IP}:${REGISTRY_PORT_HTTP}/v1/credentials/my-
    tenant/org.acme:my-device-1
```

Código B.4: Definição das credenciais.

5. **Criar uma conexão entre o Hono e o Ditto** - Como referido na Secção 3.2.1, há várias formas distintas de transmitir a informação entre o Hono e o Ditto, como o protocolo AMQP ou via Kafka. O seguinte código a executar na linha de comandos faz exatamente a definição do tipo de conexão que se pretende estabelecer (neste caso AMQP) para o *tenant* já criado acima neste exemplo:

```
HONO_TENANT=my-tenant
DITTO_DEVOPS_PWD=$(kubectl --namespace ${NS} get secret ${RELEASE}-
    ditto-gateway-secret -o jsonpath="{.data.devops-password}" |
    base64 --decode)

curl -i -X POST -u devops:${DITTO_DEVOPS_PWD} -H 'Content-Type:
    application/json' --data '{
  "targetActorSelection": "/system/sharding/connection",
  "headers": {
    "aggregate": false
  },
  "piggybackCommand": {
    "type": "connectivity.commands:createConnection",
    "connection": {
      "id": "hono-connection-for-'"${HONO_TENANT}"'",
      "connectionType": "amqp-10",
      "connectionStatus": "open",
      "uri": "amqp://consumer%40HONO:verysecret@'"${RELEASE}"'-
    dispatch-router-ext:15672",
      "failoverEnabled": true,
      "sources": [
        {
          "addresses": [
            "telemetry/'"${HONO_TENANT}"'",
            "event/'"${HONO_TENANT}"'"
          ],
          "authorizationContext": [
            "pre-authenticated:hono-connection"
          ],
          "enforcement": {
            "input": "{{ header:device_id }}",
            "filters": [
              "{{ entity:id }}"
```

```
          ]
        },
        "headerMapping": {
          "hono-device-id": "{{ header:device_id }}",
          "content-type": "{{ header:content-type }}"
        },
        "replyTarget": {
          "enabled": true,
          "address": "{{ header:reply-to }}",
          "headerMapping": {
            "to": "command/'"${HONO_TENANT}"'/{{ header:hono-
device-id }}",
            "subject": "{{ header:subject | fn:default(topic:
action-subject) | fn:default(topic:criterion) }}-response",
            "correlation-id": "{{ header:correlation-id }}",
            "content-type": "{{ header:content-type | fn:default('
"'"'application/vnd.eclipse.ditto+json'"'") }}"
          },
          "expectedResponseTypes": [
            "response",
            "error"
          ]
        },
        "acknowledgementRequests": {
          "includes": [],
          "filter": "fn:filter(header:qos,'"'"'ne'"'"','"'"'0'"'"'
)"
        }
      },
      {
        "addresses": [
          "command_response/'"${HONO_TENANT}"'/replies"
        ],
        "authorizationContext": [
          "pre-authenticated:hono-connection"
        ],
        "headerMapping": {
          "content-type": "{{ header:content-type }}",
          "correlation-id": "{{ header:correlation-id }}",
          "status": "{{ header:status }}"
        },
        "replyTarget": {
          "enabled": false,
          "expectedResponseTypes": [
            "response",
            "error"
          ]
        }
      }
    ],
    "targets": [
      {
        "address": "command/'"${HONO_TENANT}"'",
        "authorizationContext": [
          "pre-authenticated:hono-connection"
        ],
        "topics": [
          "_/_/things/live/commands",
```

```
            "_/_/things/live/messages"
          ],
          "headerMapping": {
            "to": "command/'"${HONO_TENANT}"'/{{ thing:id }}",
            "subject": "{{ header:subject | fn:default(topic:action-
    subject) }}",
            "content-type": "{{ header:content-type | fn:default('"'
    "'application/vnd.eclipse.ditto+json'"'") }}",
            "correlation-id": "{{ header:correlation-id }}",
            "reply-to": "{{ fn:default('"'"'command_response/'"${
    HONO_TENANT}"'/replies'"'") | fn:filter(header:response-required
    ,'"'"'ne'"'",'"'"'false'"'") }}"
          }
        },
        {
          "address": "command/'"${HONO_TENANT}"'",
          "authorizationContext": [
            "pre-authenticated:hono-connection"
          ],
          "topics": [
            "_/_/things/twin/events",
            "_/_/things/live/events"
          ],
          "headerMapping": {
            "to": "command/'"${HONO_TENANT}"'/{{ thing:id }}",
            "subject": "{{ header:subject | fn:default(topic:action-
    subject) }}",
            "content-type": "{{ header:content-type | fn:default('"'
    "'application/vnd.eclipse.ditto+json'"'") }}",
            "correlation-id": "{{ header:correlation-id }}"
          }
        }
      ]
    }
  }
}' http://${DITTO_API_IP}:${DITTO_API_PORT_HTTP}/devops/piggyback/
    connectivity
```

Código B.5: Definição do tipo de conexão entre Hono e Ditto, para o *tenant* criado.

6. **Criação de uma *policy*** - já relativa à criação do *digital twin* no Ditto, esta *policy* irá determinar quais as ações que se podem executar sobre o *digital twin*, ou seja, se é possível não só ler informação relativa ao *twin* como também escrever e alterar o estado do *twin*, entre outras. No caso do código exemplificado de seguida, está-se a criar uma *policy* denominada "my-policy":

```
curl -i -X PUT -u ditto:ditto -H 'Content-Type: application/json' --
    data '{
  "entries": {
    "DEFAULT": {
      "subjects": {
        "{{ request:subjectId }}": {
          "type": "Ditto user authenticated via nginx"
        }
      },
      "resources": {
        "thing:/": {
```

```
          "grant": ["READ", "WRITE"],
          "revoke": []
        },
        "policy:/": {
          "grant": ["READ", "WRITE"],
          "revoke": []
        },
        "message:/": {
          "grant": ["READ", "WRITE"],
          "revoke": []
        }
      }
    },
    "HONO": {
      "subjects": {
        "pre-authenticated:hono-connection": {
          "type": "Connection to Eclipse Hono"
        }
      },
      "resources": {
        "thing:/": {
          "grant": ["READ", "WRITE"],
          "revoke": []
        },
        "message:/": {
          "grant": ["READ", "WRITE"],
          "revoke": []
        }
      }
    }
  }
}' http://${DITTO_API_IP}:${DITTO_API_PORT_HTTP}/api/2/policies/org.
    acme:my-policy
```

Código B.6: Geração de uma *policy*.

7. **Criação do *digital twin*** - Por fim, tem-se o registo do dispositivo no Ditto e consequente criação do *digital twin*. Neste caso, está-se a criar um dispositivo associado à *policy* criada, com duas propriedades (*features*) com valor nulo - temperatura e humidade.

```
curl -i -X PUT -u ditto:ditto -H 'Content-Type: application/json' --
    data '{
  "policyId": "org.acme:my-policy",
  "attributes": {
    "location": "Germany"
  },
  "features": {
    "temperature": {
      "properties": {
        "value": null
      }
    },
    "humidity": {
      "properties": {
        "value": null
      }
    }
```

```
  }
}' http://${DITTO_API_IP}:${DITTO_API_PORT_HTTP}/api/2/things/org.
    acme:my-device-1
```

Código B.7: Criação do *digital twin*.

Posto isto, é de salientar que quando já se tem um *tenant* criado, assim como uma conexão entre Ditto e Hono para esse *tenant* já devidamente estabelecida previamente e uma *policy* criada, os passos 2, 5 e 6 podem ser suprimidos, pelo que o processo de registo de um novo dispositivo se torna bastante simples. Inclusivamente, caso se tenha um *gateway* a enviar informações ao Hono, é possível automatizar o processo de registo de novos dispositivos que comecem a enviar informação para esse *gateway*. Mais informação relativa a todo este processo pode ser encontrada na documentação oficial do C2E [51].

Apêndice C

Utilização da Aplicação Telegram

Neste apêndice são expostas as diferentes etapas necessárias à criação de um *bot* na aplicação Telegram, assim como de um grupo no Telegram, que permite que haja um envio automatizado de notificações para um utilizador. Adicionalmente, também é exposto algum do código essencial em Python que permitiu o desenvolvimento destas interações entre o utilizador e o *bot*.

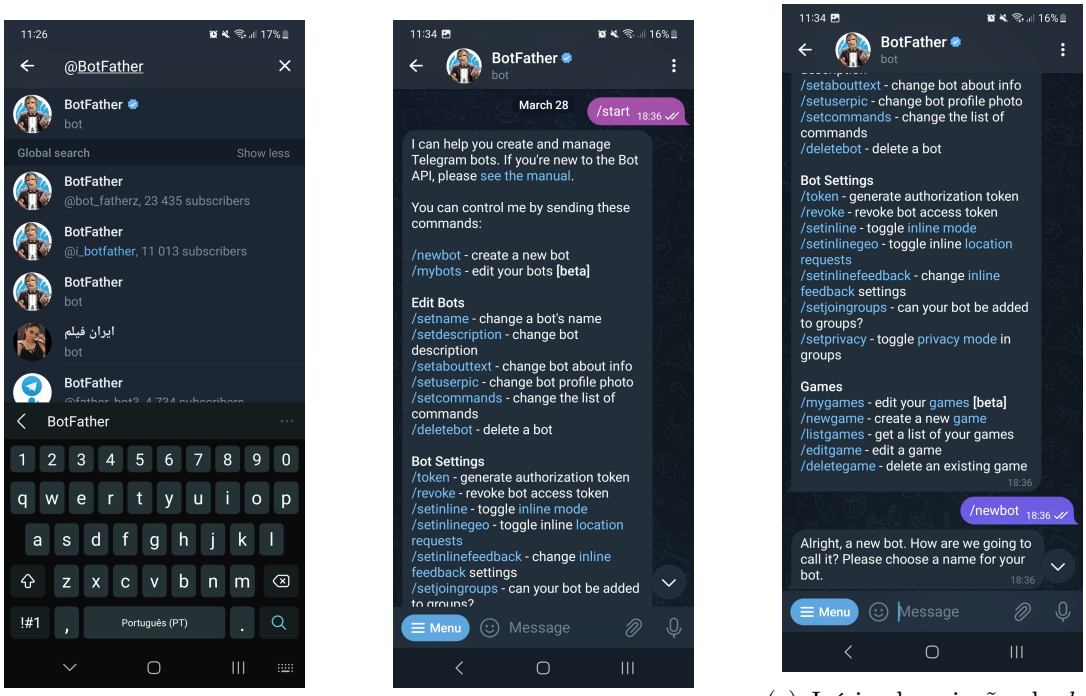
C.1 Criação do *Bot* e do Grupo Telegram

Para criar um *bot* Telegram, são necessários os seguintes passos:

1. Abrir a aplicação Telegram, e na barra de pesquisa procurar "@BotFather", escolhendo o primeiro (Figura C.1a);
2. Escrever "/start" para inicializar/ativar o *bot* "BotFather" (Figura C.1b);
3. Escrever o comando "/newbot", que irá inicializar o processo de geração de um novo *bot* (Figura C.1c);
4. De seguida, é necessário fazer a configuração do nome do *bot* e do nome de utilizador, sendo que no final é enviado um *token* que será necessário para depois se conseguir fazer a conexão ao mesmo (Figura C.2). Feitos estes 4 simples passos, é possível de seguida proceder à parte da programação do *bot*, em Python, que permitirá que o utilizador envie comandos e receba respostas.

No que toca à criação de um grupo:

1. Deve-se começar por aceder ao menu do Telegram e selecionar a opção "New Group" (Figura C.3a);
2. De seguida, devem-se adicionar os membros que se pretendam ao grupo. No caso da Figura C.3b, apenas se adicionou o *bot* criado;
3. Deve-se escolher um nome para o grupo (Figura C.3c);
4. Após isso, deve-se aceder ao URL `https://api.telegram.org/bot<bot_token>/getUpdates`, sendo que o <bot_token> corresponde ao *token* mencionado na Figura C.2. Este URL retorna um documento em formato JSON, com todas as atualizações e eventos associados ao *bot*. É importante procurar, nesse JSON, o ponto onde se tem o objeto *chat*, e retirar o ID associado ao grupo onde o *bot*



(a) Pesquisa do *bot* (b) Inicialização do *bot* (c) Início da criação do *bot* pressoal

Figura C.1: Passos 1-3 da criação de um novo *bot*.

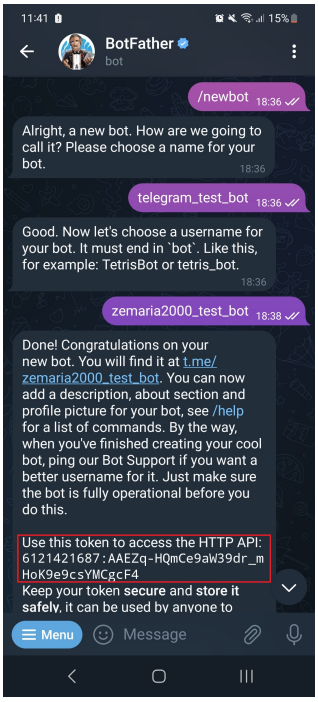
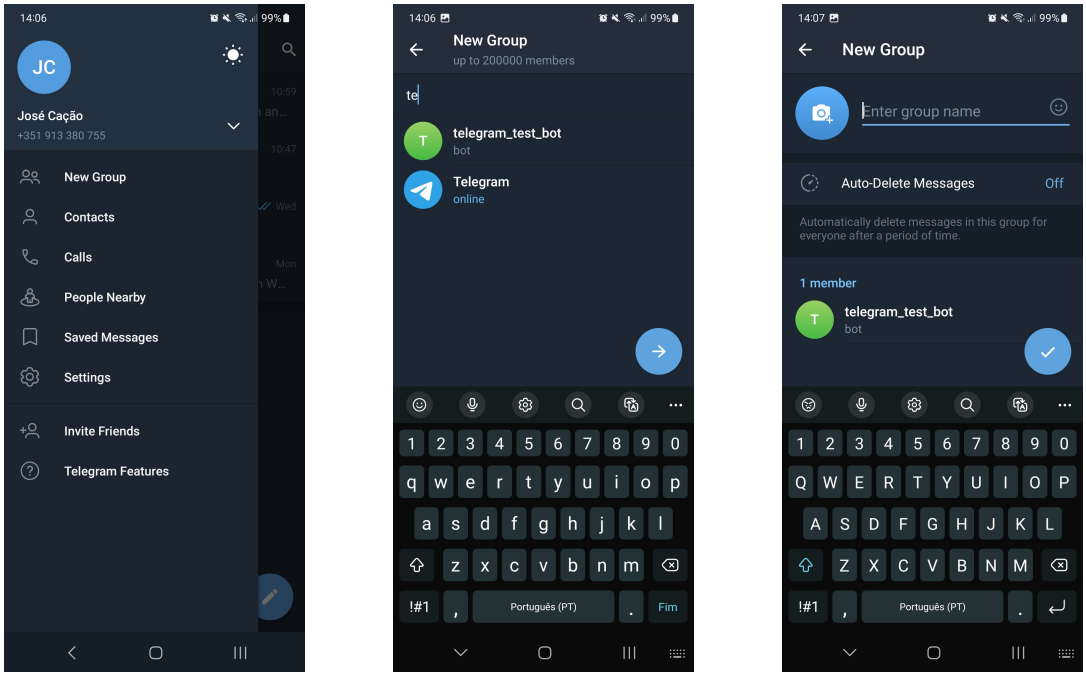


Figura C.2: Criação do *bot* e identificação do *token* associado ao mesmo.



(a) Criação de um novo grupo

(b) Seleção do *bot* para o adicionar ao grupo

(c) Determinação do nome do grupo

Figura C.3: Passos 1-3 da criação de um novo grupo.

está inserido (ver Código C.1). Este ID será depois importante para o *script* em Python, que permitirá o envio automatizado de notificações para o grupo.

```
"chat": {
    "id": -818615129,
    "title": "Notifications",
    "type": "group",
    "all_members_are_administrators": true}
```

Código C.1: Parte do JSON que permite obter o ID do grupo onde o *bot* foi inserido.

C.2 Envio de Pedidos por Parte do Utilizador

Para realizar esta primeira forma de interação, utilizou-se a biblioteca Python telebot. Um exemplo associado à configuração do *bot* criado para responder a um pedido é apresentado no Código C.2.

```
1 import telebot
2 import influxdb_client
3 from influxdb_client.client.write_api import SYNCHRONOUS
4 import pandas as pd
5 from settings import INFLUXDB
6
7
8 # 1 - Defining some important variables, as well as initiating InfluxDB
      client (whose credentials are in the file settings, INFLUX dictionary)
9 data_bucket = INFLUXDB['Bucket']
```

```
# cliente InfluxDB
client = influxdb_client.InfluxDBClient(
    url = INFLUXDB['URL'],
    token = INFLUXDB['Token'],
    org = INFLUXDB['Org']
)
query_api = client.query_api()


# 2 - Token that was generated with the bot
API_token = '6121421687:AAEZq-HQmCe9aW39dr_mHoK9e9csYMCgcF4'


# 3 - Initiating the bot
bot = telebot.TeleBot(API_token)


# 4 - Defining a function for the bot to execute - we need 3 main
    functions

# 4.1 - Function 1 - WHAT IS EXECUTED BY THE BOT WHEN WE MAKE THE REQUEST
# In this case, we'll just query data from de database regarding the last
     N values of a certain variable
def last_n_vals(n_vals, var):

    # The request to the DB
    query = f'from(bucket:"{data_bucket}")\
        |> range(start: -1h)\
        |> sort(columns: ["_time"], desc: true)\
        |> limit(n: {n_vals})\
        |> filter(fn:(r) => r.DataType == "Real Data")\
        |> filter(fn:(r) => r._measurement == "{var}")'

    # Sending it to Influx
    result = query_api.query(org = INFLUXDB['Org'], query = query)

    # Gathering the results
    results = []
    for table in result:
        for record in table.records:
            results.append((record.get_value(), record.get_time()))

    # Separating the results in values and dates/timestamps
    var_vals, date_vals = list(), list()
    for i in range(len(results)):
        var_vals.append(results[i][0])
        date_vals.append(results[i][1].strftime("%m/%d/%Y %H:%M:%S"))

    # Generating a DataFrame to organize the data in a table, and send
    them in a compact fashion
    df1, df2 = pd.DataFrame(date_vals), pd.DataFrame(var_vals)
    df = pd.concat([df1, df2], axis = 1)
    df.columns = ['Date', f'{var} Values']

    return df

# 4.2 - Function 2 - DEFINING THE REQUEST
```

```
64 # This function will look at the request we made through Telegram, and
       decide if it can execute it
65 # In this request we want something like "Last N VAR Values"
66 def last_n_vals_request(message):
67    request = message.text.split()
68    if len(request) < 3 or request[0].lower() not in "last":
69       return False
70    else:
71       return True
72
73 # 4.3 - Function 3 - EXECUTES THE REQUEST AND SENDS THE RESULT TO THE
       TELEGRAM CHAT
74 # This function will look at the request from 4.2, and if valid, execute
       it. It splits the request from the user,
75 # and will gather N measurements from variable VAR, sending the message
76 @bot.message_handler(func = last_n_vals_request)
77 def send_last_n_data(message):
78     number_readings = message.text.split()[1]
79     var = message.text.split()[2]
80     # Executing function 4.1
81     df = last_n_vals(n_vals = number_readings, var = var)
82     df.set_index('Date', inplace = True)
83     # Sending a response
84     bot.send_message(message.chat.id, df[f'{var} Values'].to_string(
       header = False))
85
86
87 # 5 - Bot will wait for requests
88 bot.polling()
```

Código C.2: Código exemplo para executar um dado pedido ao *bot* Telegram, neste caso para obter os últimos valores associados a uma variável.

Como se pode ver no código apresentado, no que diz respeito à biblioteca usada, são necessárias muito poucas configurações para se poder utilizar o *bot*. Basta apenas então identificar qual o *token* associado ao mesmo, inicializar o objeto do tipo *bot* no *script* e depois colocá-lo à escuta de pedidos que batam certo com as funções e a sintaxe definida. O que se torna mais complexo é exatamente a definição dessas funções. Para que um *bot* consiga executar dado pedido e enviar dada resposta, é necessária a definição de três funções distintas:

- Uma primeira função que define o que o *bot* deve executar em função do pedido. No exemplo dado no Código C.2, após validar o pedido o *bot* irá enviar uma *query* à base de dados, recolher os valores da base de dados, e guardá-los num objeto do tipo DataFrame, retornando-o;
- Uma segunda função que é responsável exatamente por validar o pedido. Nesta define-se a sintaxe que o utilizador deve seguir para o pedido. Caso a sintaxe esteja errada, a primeira função não será executada. Aqui poderá eventualmente incluir-se o envio por parte do *bot* de uma mensagem de erro para o utilizador;
- Por fim, a terceira função que trata de unir as duas primeiras: recebe o pedido, processa a sua síntaxe e executa a função 1. Depois, dependendo do pedido efetuado, retornará ao utilizador o pretendido.

Realizando este processo de definição tripartida de funções, é possível então desen-

volver um *bot* com um conjunto de capacidades distintas, podendo responder a diferentes pedidos por parte do utilizador.

C.3 Envio Automatizado de Notificações

Como já referido acima, para promover o envio automatizado de notificações é necessária a criação de um grupo. Obtendo o ID desse grupo, é depois possível desenvolver, em Python, a capacidade do *bot* do envio automatizado de notificações. Para isso, basta importar a biblioteca Python requests, uma biblioteca que permite o envio de pedidos HTTP de forma muito simplificada.

Um código exemplo que envia uma simples notificação, periodicamente, encontra-se exposto no Código C.3.

```
import requests
import time

# Necessary credentials
API_Token = '6121421687:AAEZq-HQmCe9aW39dr_mHoK9e9csYMCgcF4'
GroupID = '-890547248'

# Function that auto-sends notifications to telegram
def SendMessageToTelegram(Message):

    try:
        URL = 'https://api.telegram.org/bot' + API_Token + '/sendMessage?chat_id=' + GroupID

        textdata = {"text": Message}
        response = requests.request("POST", url = URL, params = textdata)

        print(response)

    except Exception as e:

        Message = str(e) + ": Exception occurred in SendMessageToTelegram"
        print(Message)


Message = "Hello World!"

while True:

    SendMessageToTelegram(Message)

    print("Message sent! \n\n")
    time.sleep(5)
```

Código C.3: Código exemplo que envia, periodicamente, uma mensagem "Hello World!" para o grupo criado

À semelhança do código apresentado na secção anterior, deve-se definir qual o *token* associado ao *bot* criado, e depois também o ID do grupo para o qual se pretende fazer o envio de notificações. Após isso, deve-se fazer a definição da função que tratará de fazer o envio das notificações de forma automatizada.

Esta notificação será enviada realizando um pedido HTTP POST no URL `https://api.telegram.org/bot<API_Token>/sendMessage?chat_id=<GroupID`, com recurso à biblioteca requests. No fim deste URL, como se pode ver nas linhas 14 e 15 do Código C.3, é adicionado o conteúdo da mensagem, neste caso um simples "Hello World!". Posto isto, no caso deste exemplo, é enviada a cada 5 segundos uma notificação para o grupo com exatamente essa mensagem.

Definindo um conjunto de diferentes funções, cada uma associada a um dado pedido HTTP, é então possível ter um *bot* capaz de enviar um conjunto de diferentes notificações, de forma automatizada, para o utilizador.

Intentionally blank page.

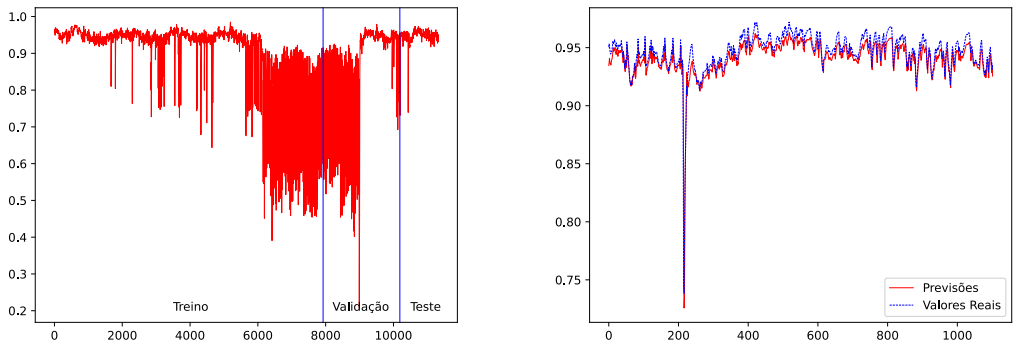
Apêndice D

Resultados da Avaliação de Desempenho dos Modelos

Neste apêndice são expostos um conjunto de gráficos e tabelas com resultados que complementam a análise conduzida ao longo da Secção 5.1.

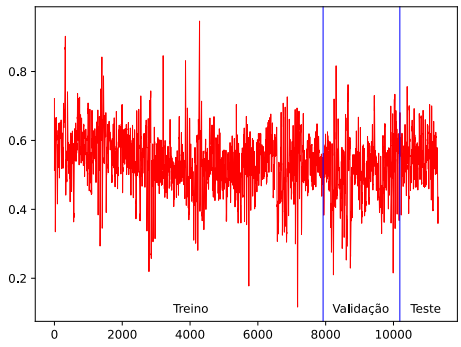
D.1 Desempenho dos Modelos na Realização de Previsões

De seguida são apresentados um conjunto de gráficos que retratam a excelente capacidade dos modelos construidos para fazerem previsões face aos valores reais. Para cada variável tem-se à esquerda a divisão dos *datasets* em treino, validação e teste, e à direita têm-se as previsões face aos valores reais para os dados de teste de cada modelo. É de salientar que nos gráficos da esquerda os dados encontram-se normalizados (entre 0 e 1), e nos da direita já estão no seu estado normal.

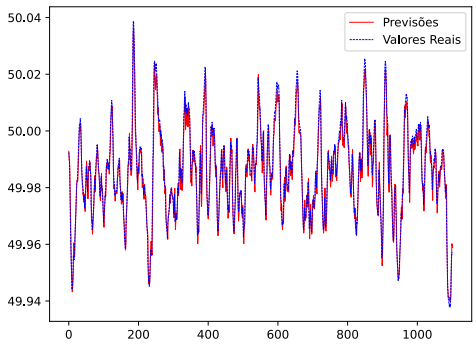


(a) Divisão do *dataset* (b) Previsões para os dados de teste

Figura D.1: Gráficos da variável C_phi_L3.

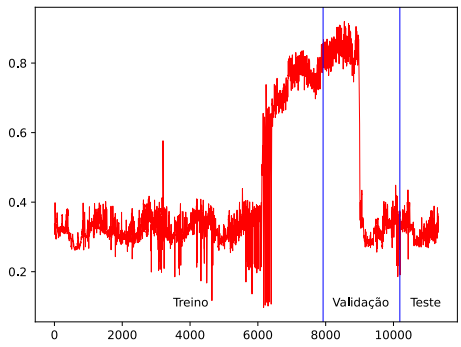


(a) Divisão do *dataset*

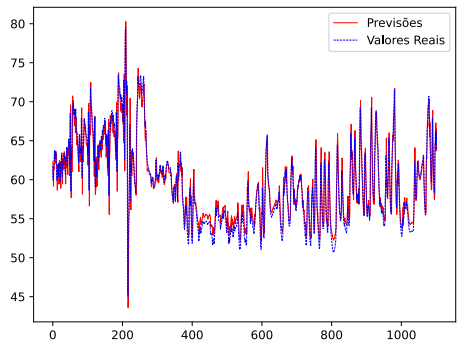


(b) Previsões para os dados de teste

Figura D.2: Gráficos da variável F.

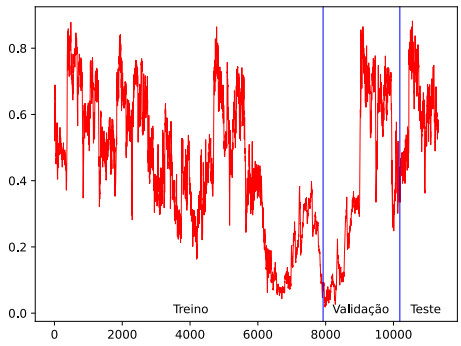


(a) Divisão do *dataset*

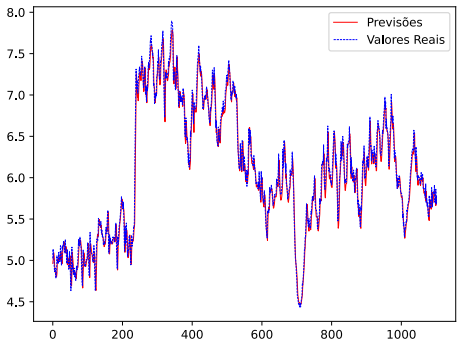


(b) Previsões para os dados de teste

Figura D.3: Gráficos da variável H_TDH_I_L3_N.

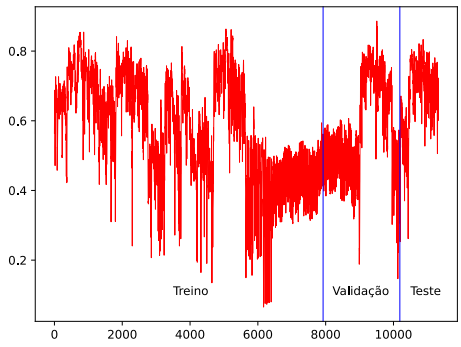


(a) Divisão do *dataset*

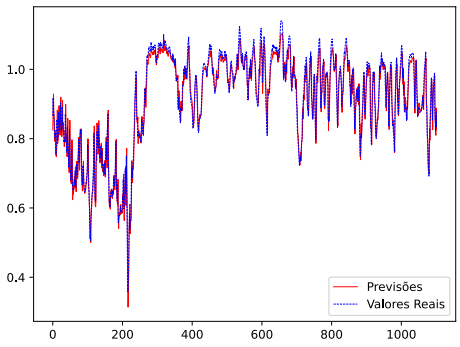


(b) Previsões para os dados de teste

Figura D.4: Gráficos da variável H_TDH_U_L2_N.

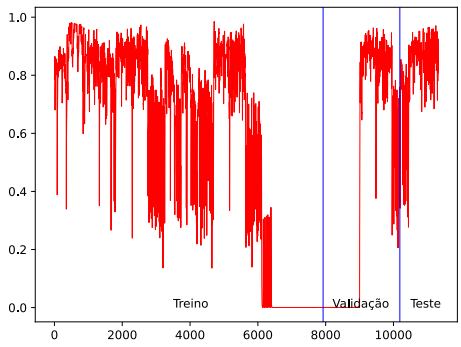


(a) Divisão do *dataset*

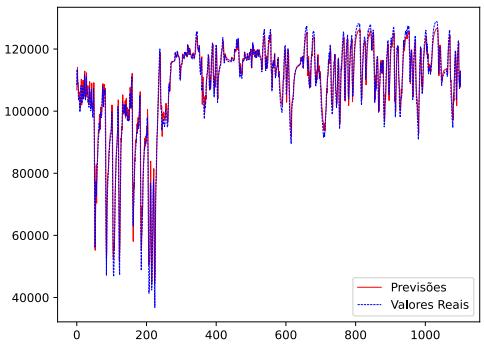


(b) Previsões para os dados de teste

Figura D.5: Gráficos da variável I_SUM.

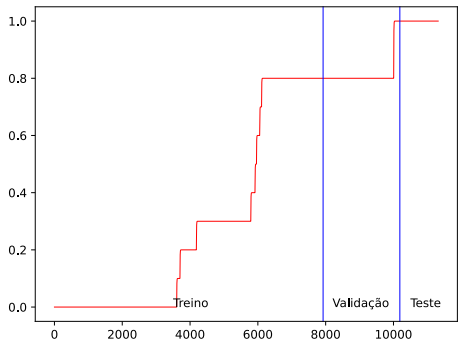


(a) Divisão do *dataset*

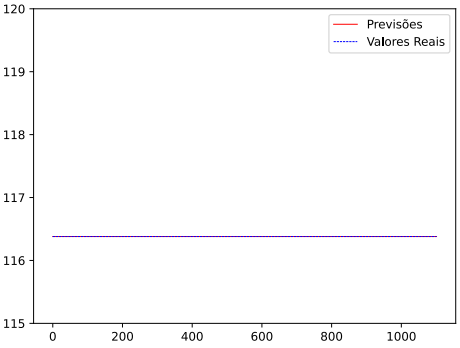


(b) Previsões para os dados de teste

Figura D.6: Gráficos da variável P_SUM.

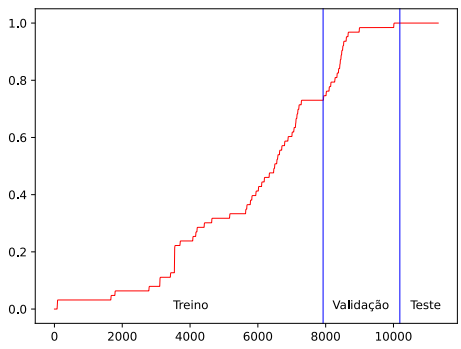


(a) Divisão do *dataset*

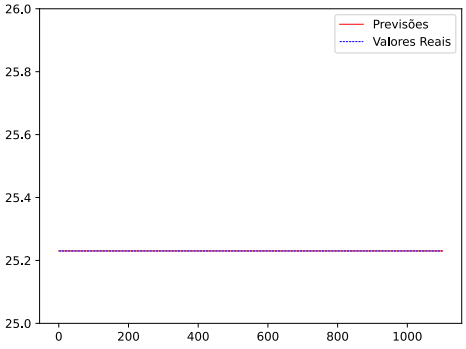


(b) Previsões para os dados de teste

Figura D.7: Gráficos da variável ReacEc_L1.

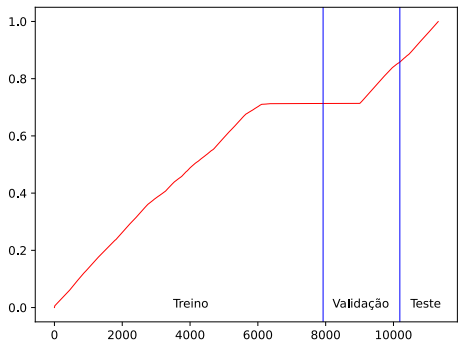


(a) Divisão do *dataset*

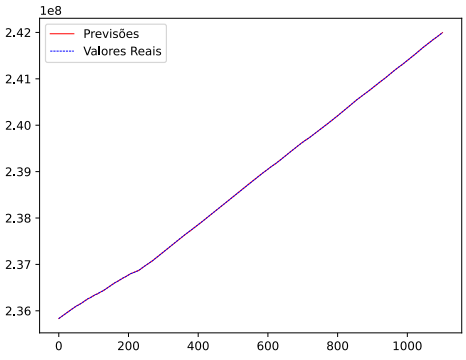


(b) Previsões para os dados de teste

Figura D.8: Gráficos da variável ReacEc_L3.

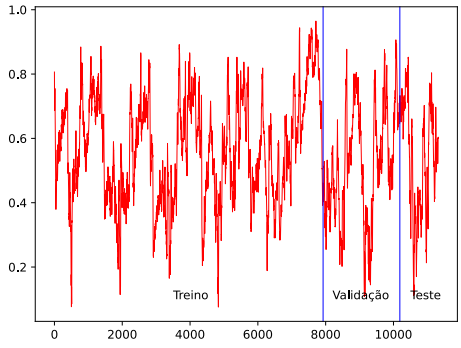


(a) Divisão do *dataset*

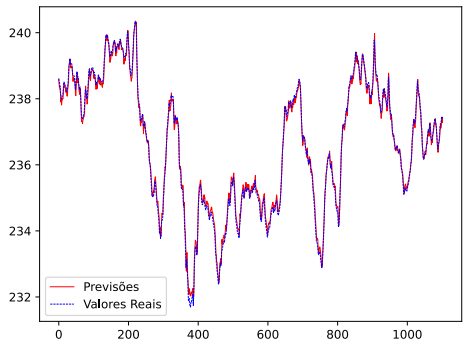


(b) Previsões para os dados de teste

Figura D.9: Gráficos da variável RealE_SUM.



(a) Divisão do *dataset*



(b) Previsões para os dados de teste

Figura D.10: Gráficos da variável U_L1_N.

D.2 Desempenho dos Modelos na Deteção de Anomalias

Como exposto especialmente na Secção 5.1.3, houve uma comparação efetuada entre os modelos otimizados com recurso ao AutoML e dois modelos base, uma regressão linear e um *autoencoder* com uma arquitetura fixa. As tabelas seguintes demonstram um conjunto de dados mais pormenorizado para cada um desses modelos, com a análise dos indicadores da matriz de confusão (FP, FN, TP e TN) assim como o MCC obtido para cada um dos modelos. As Tabelas D.1, D.2 e D.3 refletem os resultados para as anomalias pontuais. As Tabelas D.4, D.5 e D.6 os resultados para as anomalias sequenciais. É de salientar que foram, em ambos os grupos de dados com anomalias, introduzidos 100 pontos anómalos.

Tabela D.1: Resultados completos da deteção de anomalias pontuais para os modelos de regressão linear (para um α de 1.5).

Variável	**TP**	**TN**	**FP**	**FN**	**MCC**
C_phi_L3	97	913	89	3	0.6758
F	97	921	81	3	0.6940
H_TDH_I_L3_N	96	926	76	4	0.6998
H_TDH_U_L2_N	97	981	21	3	0.8816
I_SUM	92	955	47	8	0.7554
P_SUM	94	925	77	6	0.6848
ReacEc_L1	98	909	93	2	0.6732
ReacEc_L3	88	912	90	12	0.6168
RealE_SUM	17	855	147	83	0.0188
U_L1_N	76	933	69	24	0.5873
Média	**85.2**	**923**	**79**	**14.8**	**0.6288**

Tabela D.2: Resultados completos da deteção de anomalias pontuais para os modelos de *autoencoders* fixos (para um α de 1.5).

Variável	**TP**	**TN**	**FP**	**FN**	**MCC**
C_phi_L3	84	935	67	16	0.6458
F	98	938	64	2	0.7431
H_TDH_I_L3_N	75	979	23	25	0.7337
H_TDH_U_L2_N	88	962	40	12	0.7531
I_SUM	69	963	39	31	0.6290
P_SUM	94	950	52	6	0.7525
ReacEc_L1	39	936	66	61	0.3171
ReacEc_L3	34	976	26	66	0.3976
RealE_SUM	34	944	58	66	0.2930
U_L1_N	88	979	23	12	0.8180
Média	**70.3**	**956.2**	**45.8**	**29.7**	**0.6083**

Tabela D.3: Resultados completos da deteção de anomalias pontuais para os modelos de *autoencoders* otimizados.

Variável	α	**TP**	**TN**	**FP**	**FN**	**MCC**
C_phi_L3	0.624	96	946	56	4	0.7531
F	1.771	98	1000	2	2	0.9780
H_TDH_I_Ł3_N	2.153	96	1000	2	4	0.9668
H_TDH_U_L2_N	1.764	100	993	9	0	0.9535
I_SUM	1.989	91	992	10	9	0.8960
P_SUM	1.014	98	939	63	2	0.7459
ReacEc_L1	2.761	98	912	90	2	0.6798
ReacEc_L3	8.577	97	910	92	3	0.6692
RealE_SUM	5.796	98	833	169	2	0.5439
U_L1_N	0.864	92	937	65	8	0.7027
Média	-	**87.2**	**946.2**	**55.8**	**3.6**	**0.7889**

Tabela D.4: Resultados completos da deteção de anomalias sequenciais para os modelos de regressão linear (para um α de 1.5).

Variável	**TP**	**TN**	**FP**	**FN**	**MCC**
C_phi_L3	60	928	74	40	0.4624
F	24	962	40	76	0.2457
H_TDH_I_L3_N	19	925	77	81	0.1153
H_TDH_U_L2_N	25	938	64	75	0.1962
I_SUM	40	920	82	60	0.2913
P_SUM	23	904	98	77	0.1215
ReacEc_L1	20	994	8	80	0.3505
ReacEc_L3	17	994	8	83	0.3125
RealE_SUM	23	991	11	77	0.3638
U_L1_N	32	987	15	68	0.4336
Média	**28.3**	**953.3**	**47.7**	**71.7**	**0.2893**

Tabela D.5: Resultados completos da deteção de anomalias sequenciais para os modelos de *autoencoders* fixos (para um α de 1.5).

Variável	**TP**	**TN**	**FP**	**FN**	**MCC**
C_phi_L3	92	968	34	8	0.7998
F	91	989	13	9	0.8814
H_TDH_I_L3_N	76	991	11	24	0.7979
H_TDH_U_L2_N	56	985	17	44	0.6272
I_SUM	80	992	10	20	0.8286
P_SUM	27	994	8	73	0.4292
ReacEc_L1	23	991	11	77	0.3638
ReacEc_L3	0	1001	1	100	0.0000
RealE_SUM	1	1001	1	99	0.0608
U_L1_N	89	996	6	11	0.9047
Média	**53.5**	**990.8**	**11.2**	**46.5**	**0.5693**

Tabela D.6: Resultados completos da deteção de anomalias sequenciais para os modelos de *autoencoders* otimizados.

Variável	**alpha**	**TP**	**TN**	**FP**	**FN**	**MCC**
C_phi_L3	2.7157	88	997	5	12	0.9042
F	1.5090	90	996	6	10	0.9106
H_TDH_I_L3_N	1.4340	87	984	18	13	0.8336
H_TDH_U_L2_N	2.0037	78	991	11	22	0.8107
I_SUM	1.4490	84	996	6	16	0.8748
P_SUM	5.8036	22	1002	0	78	0.4518
ReacEc_L1	0.0475	97	909	93	3	0.6670
ReacEc_L3	0.0625	93	888	114	7	0.6003
RealE_SUM	0.1299	99	916	86	1	0.6949
U_L1_N	1.1642	86	995	7	14	0.8815
Média	-	**82.4**	**967.4**	**34.6**	**17.6**	**0.7629**

Os resultados para as anomalias pontuais, demonstrados nas Tabelas D.1, D.2 e D.3, validam os resultados expostos no Capítulo 5. Os *autoencoders* com otimização dos valores de α revelam os melhores indicadores, com um valor de MCC elevado quando comparado aos demais. Possuem em média os valores de TP mais elevados, indicando uma boa capacidade de detetar corretamente as anomalias introduzidas nos dados de teste. Curiosamente, possuem um maior número em média de FP face aos *autoencoders* com uma arquitetura fixa. Contudo, possuem um número de falsos negativos consideravelmente mais reduzido face aos restantes modelos. Como já referido aquando a discussão dos resultados, é preferível o modelo detetar mais anomalias do que as realmente existem do que o contrário, sendo incapaz de detetar as anomalias que realmente existem.

Quanto às anomalias sequenciais, cujos resultados são apresentados nas Tabelas D.4, D.5 e D.6, verifica-se novamente a vantagem da utilização dos *autoencoders* otimizados com AutoML, assim como do processo de otimização do valor de α. Aliás, na deteção de anomalias sequenciais os resultados da otimização são ainda mais notórios, com um valor médio de TPs de 82.4 para os *autoencoders* otimizados, face aos 53.5 dos *autoencoders* com arquitetura fixa e aos apenas 28.3 das regressões lineares. Também o número de FNs cai consideravelmente, sendo de 17.6 em média para os melhores modelos face aos 46.5 dos *autoencoders* fixos e aos 71.7 dos modelos de regressões lineares. Tudo isto se reflete no MCC, muito mais elevado no caso dos modelos otimizados.