ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Matemática

Railway Signal Monitoring

An Algorithmic Approach for Improved Maintenance Strategies

Inês Figueiredo Leão Henriques
(Licenciada em Matemática Aplicada à Tecnologia e à Empresa)

Relatório de Estágio para obtenção de grau de mestre em
Matemática Aplicada para a Indústria

Orientadores:
Doutor Filipe Santiago Cal
Doutor Nuno David de Jesus Lopes

Júri:
Presidente: Doutor Luís Manuel Ferreira da Silva
Vogais:
Doutor Tiago Gorjão Clara Charters de Azevedo
Doutor Filipe Santiago Cal

Outubro de 2024

Railway Signal Monitoring
An Algorithmic Approach for Improved Maintenance Strategies

INÊS FIGUEIREDO LEÃO HENRIQUES
(Licenciada em Matemática Aplicada à Tecnologia e à Empresa)

Relatório de Estágio para obtenção do grau de mestre em Matemática Aplicada à Indústria, na Área de Especialização de Tratamento de Dados

Orientadores:

Doutor Filipe Santiago Cal, ISEL

Doutor Nuno David de Jesus Lopes, ISEL

Júri:

Presidente: Doutor Luís Manuel Ferreira da Silva, ISEL

Vogais:

Doutor Tiago Gorjão Clara Charters de Azevedo, ISEL

Doutor Filipe Santiago Cal, ISEL

Outubro de 2024

Acknowledgements

I would like to express my deepest gratitude to my mentors, Dr. Nuno Cota and Eng[a]. Ana Rita Beire, for providing me with the opportunity to participate in and contribute to this project. Their guidance, support, and encouragement throughout this journey have been invaluable, and I am sincerely thankful for the knowledge and experience I have gained under their mentorship.

I am deeply grateful to my teachers, Professor Filipe Cal and Professor Nuno Lopes, for agreeing to guide me throughout this internship. Their unwavering support, willingness to share ideas, and positive involvement have played a significant role in making this experience both enriching and rewarding. I truly appreciate their dedication, the knowledge they have imparted during this journey and the space they provided to share my ideas, which I felt were welcome and valued.

I want to extend my heartfelt thanks to the Solvit team for welcoming me with such warmth and appreciation. The camaraderie and enjoyable moments we shared over the past months have made this experience all the more memorable. I am grateful for the sense of belonging and the support I received throughout this journey.

To my family, whose unconditional love and unwavering support have been my greatest source of strength. Their pride in my achievements means the world to me, and I am forever thankful for their constant encouragement and belief in my potential.

To my friends, for the joy they bring to my days and for constantly reminding me that life is more beautiful when we share these important moments and to my amazing class, whose laughter, support, and shared dreams have created a bond that I know will last for many years to come.

To Juliana and Bia, with whom I spent countless hours studying and completing assignments—those moments not only challenged us but also brought us closer together in ways I'll always cherish.

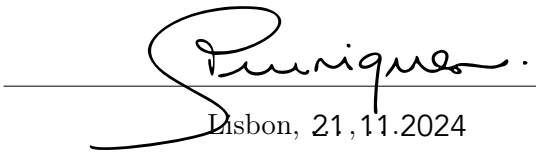
To Xana, my best friend, who never failed to check in on me, always asking about my thesis, cheering me on, and giving me so much strength. Your care and presence mean the world to me, and you hold a special place in my heart and also Takataa, my girlies—thank you for simply being you. I'm endlessly grateful to all of you for being part of this journey and making it truly unforgettable.

To Massi, for his inspiring words, showing how much he believes in me, even though he is far away. I want to take a moment to express how deeply I appreciate the communication we share. It's not just words we exchange but an understanding that brings me so much calmness and joy. Our relationship holds immense importance in my life; it's a source of strength and serenity that I treasure dearly. Thank you for being such an integral part of my journey, for the meaningful conversations, and for the connection that adds so much beauty to my days. You mean more to me than words can ever convey.

Statement of integrity

I declare that this internship report is the result of my personal and independent research. Its content is original, and all sources listed in the bibliographic references were consulted and are duly mentioned in the text. I further declare that all scientific and technical references relevant to the development of the work are duly cited and included in the bibliographic references.

The author



Lisbon, 21,11.2024

Resumo

Este estudo faz parte do projeto SIGRail Monitoring da Solvit e tem como objetivo desenvolver e otimizar algoritmos que permitam melhorar os processos de monitorização e manutenção ferroviária. O trabalho está dividido em quatro secções. A fase inicial do projeto consiste na formulação de um método *ensemble* para a identificação da linha ferroviária onde se encontra o comboio, utilizando coordenadas geográficas e técnicas de aprendizagem automática. Em seguida, o algoritmo de obtenção do ponto quilométrico (PK) é significativamente melhorado, combinando uma rede neural artificial e o método *Golden Section* para identificar a distância ótima entre dois pontos no espaço. A terceira fase do estudo consiste em definir o sentido do comboio, depois separar e contar os trajetos dentro de um determinado ficheiro. Trata-se de uma etapa crucial, pois constitui a base para a fase seguinte, a quarta etapa, em que um algoritmo dinâmico é utilizado para gerar indicadores-chave de desempenho, avaliando a qualidade de cada viagem com base em vários fatores, incluindo o nível do sinal, a qualidade do sinal, o ponto de *handover* e a célula à qual se encontra ligada. A primeira, segunda e terceira partes foram incorporadas no projeto, pelo que algumas das conclusões aqui apresentadas são o resultado desta implementação.

Palavras-chave: Monitorização Ferroviária; Aprendizagem Automática; Avaliação de Sinal; Deteção de Anomalias; Otimização.

Abstract

This study is part of Solvit's project SIGRail Monitoring and it aims to develop and optimise algorithms that will enhance railway monitoring and maintenance processes. The work is divided into four sections. The initial stage of the project entails the formulation of an ensemble method for the identification of the railway line on which the train is situated, utilising geographical coordinates and machine learning techniques. Subsequently, the algorithm for obtaining the kilometric point (PK) is significantly enhanced combining an artificial neural network (ANN) and the golden section method for identifying the optimal distance between two points in space. The third phase of the study requires defining the direction of the train, then separating and counting the journeys within a given file. This is a crucial step, as it forms the basis for the subsequent phase, the fourth stage, in which a dynamic algorithm is employed to generate key performance indicators (KPIs), evaluating the quality of each journey based on a number of factors including the level of signal, the quality of the signal, the handover point, and the serving cell connected. The first, second and third parts have been incorporated into the project, so some of the evidence presented here is the result of this implementation.

Keywords: Railway Monitoring; Machine Learning; Signal Evaluation; Anomaly Detection; Optimization.

Table of Contents

List of Figures

List of Tables

Symbols and Abbreviations

Symbols

Latin

a	Lower end of the search interval
b	Upper end of the search interval
c	Number of cells on a given line
C	Total number of classes
dBm	Decibel-milliwatts
h	Number of handovers that occur during a given journey
i	Number of iterations in the GS method
k	Number of iterations
L_0	Initial interval length
L_n	Interval length after n iterations
mW	Milliwatts
n	Size of the interval
N_0	Natural numbers including zero
$O(n)$	Order of approximation
p_i	Probability of an instance belonging to class i
P	Power in milliwatts
PK_i	Current PK
$T_{average}(n)$	Average time complexity
$T_{best}(n)$	Best time complexity
$T_{worse}(n)$	Worse time complexity
x	Train's latitude
x_{pred}	Predicted value
x_{target}	Target value
y	Train's longitude

Greek

Φ	Golden number
μ_{PK}	Mean of signal within a specified PK
σ_{PK}	Standard deviation within a specified PK

Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BS	Binary Search
CBTC	Communication-Based Train Control
CSV	Comma-Separated Values
CRS	Coordinate Reference System
DT	Decision Tree
EIRENE	European Integrated Railway Radio Enhanced Network
EPSG	European Petroleum Survey Group
ETRS	European Terrestrial Reference System
GIS	Geographic Information System
GPS	Global Positioning System
GRS	Geodetic Reference System
GS	Golden Section
GSM-R	Global System for Mobile Communications – Railway
IP	Infraestruturas de Portugal
ITRS	International Terrestrial Reference System
KPI	Key Performance Indicator
KNN	K-Nearest Neighbors
NN	Neural Network
PK	Kilometric Point
QGIS	Quantum Geographic Information System
QoS	Quality of Service
ReLU	Rectified Linear Unit
RF	Random Forest
TLC	Telecommunications
VRSA	Vila Real de Santo António
WGS	World Geodetic System

Chapter 1

Introduction

1.1 Motivation, Chapter Overview and Goals

In the contemporary business environment, organisations are continually seeking novel approaches to enhance their overall performance and products, given the existence of a highly competitive market. It is of the utmost importance to keep up with state-of-the-art technological development, come up with innovative strategies and possess a comprehensive understanding of the organisation's functioning in order to ensure they succeed. *Solvit - Innovation & Development on Telecommunications, Lda.* is a name of reference in the field of telecommunications (TLC). Headquartered in Angra do Heroísmo, in Ilha Terceira, it has offices in Lisbon and is currently looking to expand to Évora City. The company is responsible for the Portuguese railway system in terms of planning and monitoring, as represented by SIGRAIL Planning and SIGRAIL Monitoring, respectively.

The SIGRAIL Monitoring system, in which this study is integrated, enables the real-time observation of the radio network, allowing the generation of independent and robust KPIs that facilitate the expedient identification and diagnosis of quality of service (QoS) issues, as well as the validation of compliance with EIRENE requirements. It is comprised of on-board units installed on operational or test trains, which are used to run automated active and/or passive GSM-R tests. The data is then transmitted and stored by the central system.

The objective of this work is to analyse the signal data collected during regular train operation or data recorded during the regular train tests to generate KPIs, particulary to identify signal anomalies. We start by predicting the line with model ensemble, where the key idea is that insight drawn from a large group of models is likely to be more accurate than the prediction from any one model alone. Then it is necessary to determine the PKs. This search will be performed using a blend of neural networks and the binary search and golden section algorithms. The combination of regression analysis and optimization provides a powerful tool, since predictive accuracy is enhanced by solutions that prioritize efficiency and performance within constrained environments.

The second part consists of defining the direction and separating the journeys within a given file. The execution of this part now makes it possible to visualise, analyse, and therefore

interpret the data in a clearer and more responsible way. With the data well prepared, it is also possible to load it into Power Query and visualise it in Power BI trip by trip, identifying problems and inconsistencies. Finally, the identification of KPIs is addressed. Given the considerable number of observations that must be handled, it is imperative to implement an automated data analysis process. Our analysis is focused on the identification of signal anomalies. The rationale for this study is the necessity of anticipating potential concerns in order to ensure the smooth operation of the railway line, thereby avoiding issues such as discovering problems with certain antennas. Its structure enables efficient management of data and its adaptability, combined with its simplicity, presents a valuable opportunity for refinement and optimization in future iterations.

Portugal's railway system is undergoing a process of modernisation, with a particular focus on enhancing connectivity, reducing travel times and improving service quality, particularly on routes that experience high demand. The diversity of train operations and the structure of the network reflect Portugal's strategic efforts to maintain an efficient, flexible, and regionally integrated railway system. This study demonstrates the potential of machine learning and shows that, by employing appropriate optimisation techniques, it is possible to enhance the performance of existing processes, thereby creating opportunities for further improvements in other areas.

1.2 Structure

The remainder of this report is organized as follows. Chapter 2 proposes the use of ensemble modeling - neural networks and random forests (RF) - for classifying the line with data received from the probe. Chapter 3 describes the search for PKs using machine learning and optimization models. The identification of the journey's direction and segmentation is presented in Chapter 4. Chapter 5 introduces the proposed algorithm for KPI generation. The final chapter offers concluding remarks, summarizing the key findings of the project and providing the final considerations. Suggestions for future work are also given here.

1.3 Software

The software used to carry this study was the following:

Excel

In this project, Excel was utilized in two key instances to manage and organize data. First, an Excel file was employed to establish the linkage between the CSV files and their corresponding segments of railway. This file served as a reference guide, mapping each CSV dataset to its specific segment, thereby facilitating data integration and organization. Secondly, a separate file has been used to document each of the variables contained in the files coming from the probe. - the *Testlog* files. It contained definitions, units of measurement, and any relevant notes about the variables.

	Linha/Troço	IET 50					Ficheiro da Linha			
		Início		Fim		Extensão (km)	Nome	PK		Extensão (km)
		cod seg	PK	cod seg	PK			Início	Fim	
37	Linha do Sul									
	Agulha Junto à Ponte Santana – Pinhal Novo	371	0.000	371	36.806	36.8	2014_PteSantana_PinhalNovo_VD	0.000	36.806	36.8
	Pinhal Novo - Setúbal	372	15.439	372	28.222	12.8	2014_PinhalNovo_Setubal_VA	15.445	28.157	12.7
	Setúbal - Bifurcação de Águas de Moura-Sul	372	28.222	372	47.506	19.3	2014_Setubal_BAMoraSul_VU	28.157	45.045	16.9
	Bifurcação de Águas de Moura-Sul – Pinheiro	373	8.162	373	20.600	12.4	2013_BAMoraSul_Pinheiro_VU	8.166	20.600	12.4
	Pinheiro – Ermidas-Sado	374	57.777	374	129.631	71.9	2014_Pinheiro_Ermidas_VU	57.777	129.633	71.9
	Ermidas-Sado - Bifurcação de Torre Vã	374	129.631	374	149.760	35.1	2013_Ermidas_TorreVa_VD	129.675	149.815	20.1
	Bifurcação de Torre Vã – Funcheira	374	149.760	374	164.681	152.1	2012_TorreVa_Funcheira_VA	149.845	164.637	14.8
	Funcheira - Tunes (Algarve)	375	217.600	375	301.889	84.3	2018_Funcheira_Tunes_VU	217.602	301.841	84.2
38	Linha de Sines									
	Ermidas-Sado (Sul) - PK extremo da Linha de Sines	381	129.631	381	180.319	50.7	sines	129.630	179.950	50.3
39	Linha de Évora									
	Casa Branca (Alentejo) - Évora	391	90.406	391	116.570	-	Linha391_VU_CB-EV	90.710	117.682	-
	Évora - PK extremo da Linha de Évora	391	116.570	391	126.800	10.2	Não há Linha!			
45	Linha do Algarve									
	Lagos - Tunes	451	347.210	451	301.889	-	2012_Tunes_Lagos_VU	347.095	301.939	-
	Tunes - PK extremo da Linha do Algarve	452	301.889	452	396.468	-	2013_Tunes_VRSA_VU	301.898	395.937	-
46	Concordância do Poceirão									
	Bifurcação do Poceirão (Descendente) - Bifurcação de Agualva	461	0.000	461	3.030	3.0	2011_Bpoceirao_BAgualva_VD	0.008	5.580	5.6
	Bifurcação de Agualva - Bifurcação de Águas de Moura (Sul)	461	3.030	461	8.162	5.1	2011_Bagualva_BAMouraSul_VU	5.575	8.161	2.6
55	Concordância do Bombel									
	PK Inicio Concordância Bombel - Agulha 2 de Vidigal	551	0.000	551	3.112	3.1	BombVidi	0.090	3.100	3.0
58	Ramal do Louriçal									
	PK 0,000 Ramal do Louriçal - Ramal Celbi/Ramal Soporcel (Inserção)	581	0.000	581	5.510	5.5	2018_RamalLouriçal_VU	0.018	5.495	5.5
68	Variante de Alcácer									
	Início Variante de Alcácer - Extremo Variante de Alcácer	681	58.741	681	87.517	28.8	alcacer_exported	59.213	87.517	28.3
79	Ramal Neves Corvo									
	Ourique (Alentejo) - Extremo Ramal Neves Corvo	791	0.000	791	31.217	31.2	our_mnc_exported	0.014	30.862	30.8
80	Ramal Soporcel									
	Ramal Celbi/Ramal Soporcel (Inserção) - Ramal Soporcel (Extremo)	801	5.510	801	8.010	2.5	soporcel_exported (Linha errada?)	5.510	6.871	1.4
90	Ramal de Porto de Aveiro									
	PK 0,000 do Ramal Porto de Aveiro - Porto de Aveiro (Início)	901	0.000	901	8.750	8.8	2011_RamalPortoAveiro_VU	0.000	8.526	8.5
-	Linha Évora - Elvas									
	Évora - Elvas					0.0	Lote A	126.000	146.500	20.5
							Lote_B	146.500	167.000	20.5
							Lote_C	167.000	204.251	37.3

Figure 1.1 Extract from the Excel file connecting the CSVs with the segments.

Power Query

Data transformation tool that is used for importing data from various sources such as Excel, CSV files, databases and entire directories. Useful for cleaning, filtering, merging, reshaping, and manipulating data to prepare it for analysis. This includes handling missing values, splitting or combining columns, and changing data formats. Once transformed, the data can be loaded into applications like Excel, Power BI, or databases for further analysis or reporting.

Power BI

Power BI is used for creating interactive dashboards, reports, and visualizations to represent data in a clear and insightful way. The visual components help explaining trends, patterns, and in this particular case anomalies. Power BI allows to perform advanced data analysis by writing formulas and using built-in analytics features, updating in real-time thus allowing for dynamic conclusions. This software can connect to a wide range of data sources.

QGIS

Open-source software used for geospatial data analysis, mapping, and geographic information system (GIS) applications. It allows users to visualise and manage spatial data. QGIS enables the creation of detailed and customizable maps by layering different types of spatial data over base maps like satellite images or topographic maps. It is possible to visualize geographic features and overlay data to show patterns, trends, and relationships. It also allows for operations like merging, intersecting, and reprojecting datasets to prepare them for analysis or visualization, and to convert coordinate systems.

Python

Versatile, high-level programming language dominant in data science due to libraries such as pandas, NumPy, and Matplotlib, which simplify data manipulation, analysis, and visualization. Python is extensively used for artificial intelligence (AI), deep learning, and ANN development. Libraries such as TensorFlow, Keras, scikit-learn, and PyTorch provide powerful tools for implementing deep learning models for tasks like image recognition, classification, regression, natural language processing, and reinforcement learning, and for developing predictive models, neural networks, and other AI-driven applications. The version used in this study was *Python 3.12.2.*

Chapter 2

Railway Line Identification

Portugal's railway system, overseen by Infraestruturas de Portugal (IP), plays a vital role in both passenger and freight transportation. The network consists of approximately 2,600 kilometers of track, with major lines such as Linha do Norte serving as the backbone of the system, connecting key urban centers including Lisbon, Porto, and Coimbra. Other significant lines include Linha do Sul, Linha da Beira Alta, and Linha do Douro, each serving different regions of the country.

Linha do Algarve is a crucial railway line in the southern region of Portugal, connecting the coastal cities of Lagos and Vila Real de Santo António. Spanning approximately 140 kilometers, this single-track, non-electrified line plays a vital role in regional transportation, serving both local commuters and tourists who frequent the Algarve. Despite its importance, this line faces challenges related to infrastructure and service efficiency, prompting ongoing discussions about potential modernization projects, including electrification and improved rolling stock.

Focusing on this specific segment of the Portuguese railway network, current projects aim to enhance the line's capacity, reduce travel times, and promote greater integration with national and international networks. The modernization of Linha do Algarve would not only improve the quality of service for daily commuters but also bolster the region's tourism economy by providing more reliable and environmentally sustainable transport options. Given the Algarve's status as a major tourist destination, these improvements are seen as a strategic priority in Portugal's broader transportation development plans. Figure 2.1 shows the map of the national railway network where there is active rail traffic (for a more detailed view of the same image with a description, see Appendix A).

The Portuguese rail infrastructure encompasses a variety of train operations, characterized by distinct patterns of movement and routing. Some trains traverse multiple lines, moving seamlessly from one segment or line to another, covering extensive sections of the railway system. In contrast, others, such as those operating on Cascais and Algarve, remain confined to a single line, moving forward and backward along the same track without altering the train's position or orientation. This operational mode is typical of commuter services, where trains follow a more localized, repetitive route.

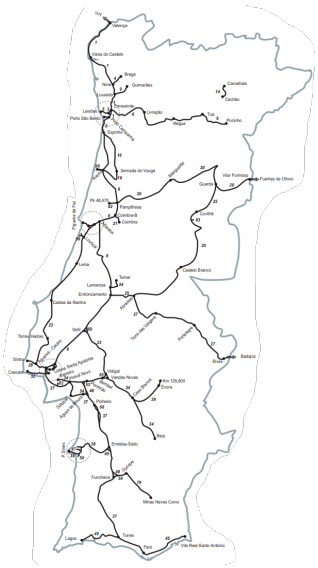


Figure 2.1 Map of Portugal's railway network with rail traffic.
Figure from [9], 2005.

This study will focus exclusively on a single train line (Linha do Algarve, or line 45) within the Portuguese rail system to ensure a detailed and comprehensive analysis. By concentrating on one line, we can thoroughly examine its specific operational characteristics, challenges, and performance metrics, providing a clearer and more accurate evaluation. Limiting the scope to one line allows us to gather more precise data and account for localized factors such as geographic conditions, which may vary significantly across the network. Finally, focusing on a single line enhances the feasibility of the study, making it more manageable within time and resource constraints, while still delivering meaningful insights.

The line is divided into two key segments for operational and infrastructure purposes. Segment 451 extends from Lagos to Tunes, while segment 452 covers the route from Tunes to Vila Real de Santo António, as can be seen in Figure 2.2. These segments form the entirety of the Algarve line, connecting important urban centers along the southern coast. The division into two segments allows for a more detailed and targeted approach, ensuring that the unique characteristics and demands of each section are addressed effectively.

Figure 2.2 Segment 451 (on the left) and segment 452 (on the right) of Linha do Algarve.
Figure by author, using QGIS, 2024.

It is true that the railway system is described in great detail in official documentation [7, 9]. However, the only way to determine the position of the train from the data transmitted by the probe is to use geographical coordinates. A combination of machine learning models and the aforementioned coordinates will be employed in an attempt to identify the line. Global

Positioning System (GPS) coordinates latitude and longitude come expressed in degrees. In contrast, the files pertaining to the mapping of the railroad lines are written in a different coordinate system. In order to apply the machine learning models to the data, it is first necessary to modify the coordinate system from the local one, used in Portugal, to the global one.

2.1 Coordinate System Conversion

Files containing the railway mapping are in the EPSG:3763 - ETRS89/Portugal TM06 system, which is a coordinate reference system (CRS) used mainly in Portugal. The EPSG code is a unique identifier for a specific CRS and stands for *European Petroleum Survey Group*, the organisation that created the CRS database. The EPSG code for ETRS89/PortugalTM06 is 3763. The first part, ETRS89, stands for *European Terrestrial Reference System 1989*. It is a Geodetic Reference System (GRS) widely used in Europe. ETRS89 is based on the International Terrestrial Reference System (ITRS), but is fixed to the stable part of the Eurasian Plate, which makes it very useful for mapping and surveying in Europe. The remaining part of the code, Portugal TM06, is a specific projection of the ETRS89 geodetic datum adapted for use in Portugal (TM06 refers to a Transverse Mercator projection centred on the meridian 6 degrees west of the Greenwich meridian). This projection is designed to minimise distortions for mapping purposes within Portugal's national territory. This CRS is commonly used for topographic mapping, geospatial data analysis and GIS applications in the country.

In order to train the machine learning models with data in the same units as that received from the probe - latitude and longitude - it is necessary to convert them to the same coordinate system, the EPSG:4326 - WGS 84, one of the most widely used in the world. The code 4326 refers to the World Geodetic System 1984 (WGS 84), the global GRS that is used to define coordinates anywhere on Earth. WGS 84 is the standard reference system for GPS. Latitude ranges from -90° to 90°, from the South Pole to the North Pole respectively, and longitude ranges from -180° to 180°, with the Greenwich meridian at 0° as the reference point. WGS 84 defines the shape of the Earth as an ellipsoid and the Earth's location in space and is continually adjusted to better reflect the physical reality of the Earth. Used globally, WGS 84 has many applications, including navigation, mapping, and GIS. This CRS is fundamental to many technologies and applications that rely on precise location, including GPS navigation, global mapping and location-based services. This was accomplished through the utilisation of the QGIS software.

Now that all the data has been converted to the same units of measurement, we can start to build the models. To improve the accuracy of the tests, while reducing the cost of storing, training and inferring from multiple models, we will construct aggregations of models - model ensembling. It is widely accepted that the performance of a set of many weak classifiers is usually better than that of a single classifier, given the same amount of training information.

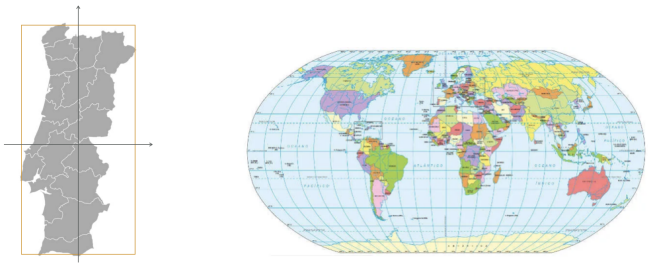


Figure 2.3 Coordinate system conversion from EPSG:3763 - ETRS89/Portugal TM06 (on the left) to EPSG:4326 - WGS 84 (on the right).

2.2 Model Ensembling

Ensemble methods generate many classifiers and combine their results. This is a powerful technique in machine learning, whereby multiple models are associated to enhance overall prediction accuracy and model stability, with the objective of improving predictive performance. The fundamental premise is that a group of relatively weak learners can be combined to form a single, more powerful learner.

An ensemble model typically comprises two stages. Initially, multiple machine learning models are trained independently. Then, their predictions are aggregated in some way, through voting, averaging, or weighting. The ensemble is then employed to make the prediction. This ensemble will combine four RF models and three ANNs to produce an odd number of predictions. This way, when the time comes to vote, there will be no tie.

2.3 Random Forest Classifier

The random forest classifier is an ensembling method applied to decision trees. A decision tree (DT) is a supervised learning model used to predict the value of a target variable Y, using information from several input variables that appear condensed in a matrix X. Each internal node contains a True/False question to be answered about a particular X variable. Based on this, the tree splits into branches. At the end of each branch is another question, and this recursive process continues while the tree gets deeper and wider with each decision made until it ends in a leaf node that makes a prediction. [5]

To create the first decision node, one must pick a variable to split on. DTs use a criterion called Gini index to pick the right variable to look at. After comparing each possible variable to split on, the DT picks the one that leads to the purest branches. The Gini index, default criterion on the *RandomForestClassifier* function in Python to measure the quality of a split, is a metric used to measure the inequality of a dataset's distribution of classes (for mathematical formulation, see [8]). It quantifies the probability that a randomly chosen element from the dataset would be incorrectly classified if it was labeled according to the distribution of labels in the dataset. The Gini index is calculated as:

$$Gini = 1 - \sum_{i=1}^{C} p_i^2 \tag{2.1}$$

Where:

- p_i is the probability of an instance belonging to class i;
- C is the total number of classes.

A Gini index of 0 implies perfect purity, meaning all elements belong to a single class. A Gini index close to 0.5 indicates maximum impurity, meaning the elements are equally distributed among different classes. When building a decision tree, the Gini index is used to decide the best feature to split the data at each node. The feature that results in the lowest Gini index is chosen as the split. The goal is to achieve the highest possible purity in each resulting subset, which helps in making accurate classifications.

In this method of ensembling, we train decision trees, hence the name forest, to take a vote. The class with the most votes becomes the output. The reason behind this choice is that a RF is not only a computationally efficient technique, but can operate quickly over large datasets and has several real-world applications, see for instance [6, 16]. One key aspect to keep in mind for the success of the predictions using RFs is uncorrelatedness. The trees must be different from each other and disagree in splits and predictions. Large groups of uncorrelated trees working together in an ensemble will outperform any of the constituent trees, for the forest is shielded from the errors of individual trees. Methods to decorrelate trees are bootstrap aggregating and feature randomness.

2.3.1 Bagging

Bagging, or bootstrap aggregating, is an ensemble method that involves training multiple models independently on random subsets of the data, and aggregating their predictions through voting (in the case of classification) or averaging (in the case of regression). In detail, each model is trained on a random subset of the data, sampled with replacement, whereby individual data points may be selected more than once. This random subset is referred to as a bootstrap sample. [1]

The application of bagging in conjunction with the training of models on disparate bootstraps serves to diminish the variance exhibited by the individual models. Furthermore, this approach circumvents the issue of overfitting by subjecting the constituent models to disparate segments of the dataset. The predictions from all sampled models are combined through a straightforward voting process, thereby generating the final overall prediction. This approach allows the aggregated model to benefit from the positive aspects of the individual models while compensating for their shortcomings.

Bagging is particularly effective in reducing variance and overfitting, thereby enhancing the robustness and accuracy of the model, particularly in instances where the individual models

are susceptible to high variability. For the mathematical formulation and detailed explanation, see [4].

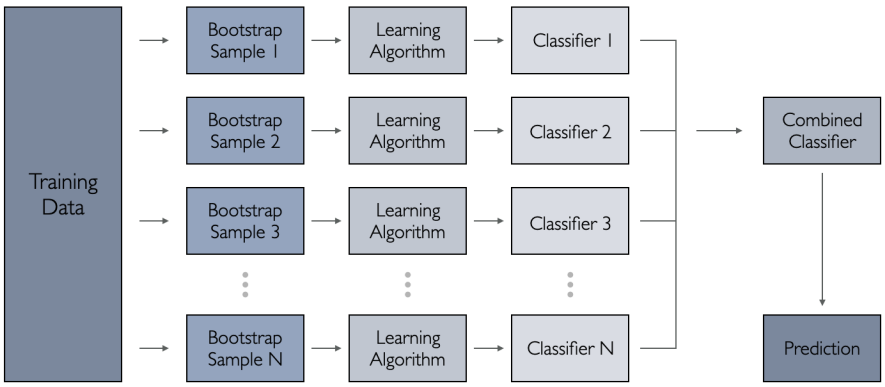


Figure 2.4 Bagging (Bootstrap Aggregating) Flow. Adapted from the original by [20].

2.3.2 Feature Randomness

The second way to introduce variation in decision trees is feature randomness. This technique consists of shuffling which features each tree can split on. By doing so, it encourages diverse trees. In the random forest algorithm, feature randomness is introduced through a process called feature bagging, or random subspace method. This is one of the key mechanisms that help improve the model's performance by reducing overfitting and making it more robust.

When constructing individual decision trees in the forest, each time a split is considered at a node, the algorithm doesn't evaluate all the features. Instead, it selects a random subset of features from the total set of features, and only those selected features are considered for the split. The best split is chosen based on the subset, not the full set of features.

This randomness is controlled by the hyperparameter *max_features* in scikit-learn's *RandomForestClassifier*. By default, for classification problems, this random subset of features is set to the square root of the total number of features. This randomness ensures that the trees in the forest are decorrelated. If all trees were built considering the same set of features, they might be very similar, reducing the diversity and hence the power of the ensemble method. By forcing each tree to look at a different subset of features, the RF reduces the risk of overfitting to the training data and increases the overall generalization power of the model.

Note: Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. [15]

2.3.3 Models

Data Preparation and Exploration

Before starting to construct the models, it was necessary to organise the files according to the document shown in Figure 1.1 ([7]) and to name these files with the number that identifies the line or segment. Then, the first step is to import the data (latitude and longitude) from all the files, adding a column with said line by using the name of the CSV file. There are a total of 1,984,186 entries with no null/missing values from 36 different classes.

The data is then divided into two distinct sets, one for training and one for testing ($test\_size = 1/3$). It should first be noted that the lines in question exhibit differing lengths, which consequently affects the PK count. The stratification ($stratify = y$) was employed due to the considerable variation in file size, as illustrated in Table 2.1 and the designation of a random state ($random\_state = 42$) was introduced for the purpose of ensuring replicability.

Line	Size of the file	Line	Size of the file	Line	Size of the file
81	337185	372	29539	901	8545
252	211645	251	28579	461	8270
201	200402	281	27308	681	7443
231	196434	391	26875	91	7057
61	162687	321	25152	791	6733
12	130536	375	20300	41	3798
341	108590	51	16572	11	2640
374	105607	242	14707	211	1682
452	94231	221	14526	581	1198
331	66561	373	13395	31	942
451	45175	381	11683	551	730
371	36699	292	10462	801	298

Table 2.1 Size of the files organised in descending order.

Hyperparameter Tuning

As the number of trees grows, it does not always mean the performance of the forest is significantly better. So the next step is to try to find an optimal number of trees, that is to find a threshold from which increasing the number of trees would bring no significant performance gain, and would only increase computational cost.

In order to select the most appropriate hyperparameters for the model, it was necessary to evaluate a number of potential options for max_depth and $n\_estimators$. The maximum depth was set to the default value (None), and the number of estimators was tested for every value from 10 to 200, in increments of 10. Subsequently, the standard value for $n\_estimators$

(100) was set and the max_depth was tested for every value from 10 to 50, with increments of 5. The three parameters that showed the best accuracy on the test set under these conditions were 170, 190, and 200 for $n\_estimators$ and 40, 45, and 50 for max_depth. Now, we test all the nine possible combinations using the *sk.learn* function *GridSearchCV* and performing a manual search. Cross-validation was used for robust performance estimation.

GridSearchCV

GridSearchCV is a tool used to fine-tune the hyperparameters of our model, leveraging the three most promising parameter combinations identified during an initial manual search. A 3-fold cross-validation approach was employed to ensure robust model evaluation. The optimal hyperparameters determined through this process were a maximum tree depth (max_depth) of 50 and 190 decision trees ($n\_estimators$). This will be model number 1.

Manual Search

The manual search method tests every conceivable combination from the three best parameters of max_depth and $n\_estimators$. The pair that showed the best results in terms of accuracy was $(170, 40)$, so this will be model number 2.

Random Forest #1

Parameters:

- $n\_estimators = 190$
- $max\_depth = 50$

The accuracy of this model was 99,92% on the test set, which comprised 1,322,790 instances, corresponding to 1110 mistaken observations. The latitude played a bigger part in the prediction, being responsible for 63,26% and the longitude 36,74%.

Random Forest #2

Parameters:

- $n\_estimators = 170$
- $max\_depth = 40$

The accuracy of this model was 99,91%, corresponding to 1136 mistaken observations. Also here, the latitude played a bigger part in the prediction, being 63,18% in comparison to 36,82% from the longitude.

The next two models follow the same guidelines, treatment, pre-processing and parameters, the only difference being the addition of the height attribute.

Random Forest #3

Parameters:

- $n\_estimators = 110$
- $max\_depth = 45$

The accuracy of this model was 99,99% on the testing set, which is about 116 errors. The latitude played a bigger part in the prediction, being responsible for 49,36%, the longitude 31,89%, and the altitude 18,75%.

Random Forest #4

Parameters:

- $n\_estimators = 180$
- $max\_depth = 45$

The accuracy of this model was 99,99%, corresponding to 106 errors. The latitude gave the biggest contribution, being responsible for 49,33%, the longitude for 31,89%, and the altitude the remaining 18,78%.

2.4 Artificial Neural Networks

Artificial neural networks are computational models designed to process complex patterns and perform tasks such as classification, prediction, and decision-making. In Python, ANNs are commonly implemented using machine learning libraries such as TensorFlow, which provide structure for building and training neural networks across a range of applications, including image recognition, natural language processing, and time series analysis.

An ANN typically consists of multiple layers of interconnected neurons, organized into an input layer, one or more hidden layers, and an output layer. The neurons within each layer are connected by weighted edges, which are adjusted during the training process to minimize errors in the network's predictions. The hidden layers usually employ a non-linear activation function such as the Rectified Linear Unit (ReLU), which introduces non-linearity into the model, enabling it to learn complex patterns and relationships in the data. The output layer, particularly in classification tasks, often uses the softmax activation function, which converts the network's output into a probability distribution across multiple classes. [11]

The training of an ANN is driven by an optimization algorithm, with one commonly used option being the *Adam* optimizer. *Adam* combines the advantages of both momentum and adaptive learning rates to accelerate convergence and improve performance. The loss function, which quantifies the error between the predicted and actual values, is crucial for guiding the optimization process. For multi-class classification tasks, the *categorical_crossentropy*

loss function is typically employed, as it is well-suited to measure the difference between the predicted probability distribution and the true class labels.

Python's machine learning libraries streamline the implementation of these components, enabling to efficiently design, train, and evaluate neural networks for a wide array of tasks in both academic and industrial settings.

2.4.1 Models

There is no formula for the choice of the number of layers and number of neurons. With regard to the number of layers, the guidelines set out in Table 2.2 have been followed. With regard to the number of neurons, there are several hypotheses. One such hypothesis is that the number of neurons should be calculated using the following formula:

$$\#\ of\ neurons = \sqrt{input\ size \times output\ size} \tag{2.2}$$

which would be approximately $\sqrt{2 * 36} \approx 8,48$, so about 8 neurons. Some argue that the size of the output should be within a certain range, typically between that of the input and the output. They contend that the number of neurons should never exceed twice that of the input [12]. These principles serve as preliminary guidelines for experimentation rather than definitive formulas. In the majority of real-world problems, fine-tuning through experimentation and techniques such as cross-validation will yield superior results than relying solely on a formula. The models were selected based on the above mentioned empirical evidence.

Number of hidden layers	Result
None	Only capable of representing linear separable functions or decisions
1	Can approximate any function that contains a continuous mapping from one finite space to another
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy

Table 2.2 Determining the number of hidden layers. Table from [12].

ANN #1

The input size is 2 - latitude and longitude. A total of 420 ((2×8+8)+(8×8+8)+(8×36+36)) parameters were trained. The accuracy of this model is 96,46%.

ANN #2

The input size is 2 - latitude and longitude. A total of 312 ((2×6+6)+(6×6+6)+(6×36+36)) parameters were trained. The accuracy of this model is 96,51%.

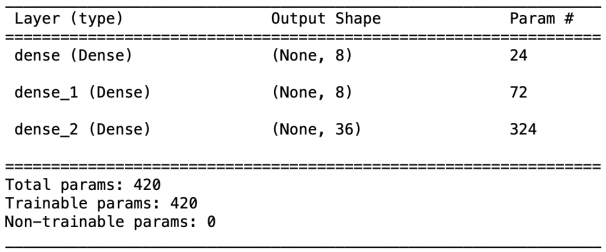


Figure 2.5 Architecture of ANN #1 composed by 2 hidden layers with 8 neurons each.

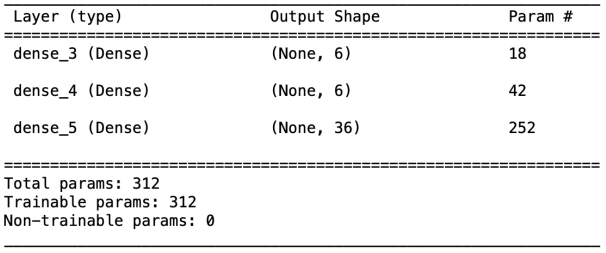


Figure 2.6 Architecture of ANN #2 composed by 2 hidden layers with 6 neurons each.

ANN #3

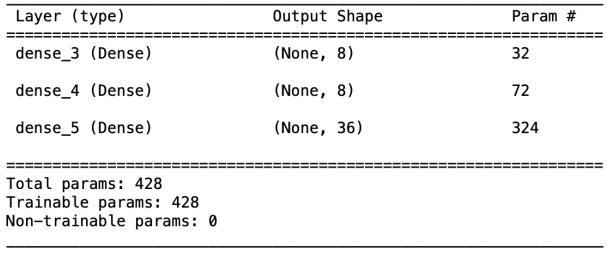


Figure 2.7 Architecture of ANN #3 composed by 2 hidden layers with 8 neurons each.

Note: In a dense neural network, also known as a fully connected neural network, each neuron in a given layer is connected to every neuron in the subsequent layer. [14]

The distinguishing feature of this model is the incorporation of the attribute height. To illustrate, trains coming from the south, crossing the 25 de Abril bridge near Alcântara, pass over Linha de Cascais. This model will be useful in situations where the height distinguishes between lines. The accuracy of the model is 97,19%, which is slightly higher than the previous ones. However, on some many cases, the latter gave a better prediction. It is important to retain all three models, as it is preferable to have a robust estimation than a perfect one.

2.5 Errors

The errors identified in the forecast are primarily related to the crossing of lines. As illustrated in Figure 2.8, there is a section between the end of one line and the beginning of the other where there is no line assignment. The points in the vicinity of this area are prone to generating errors.

Figure 2.8 The station of Tunes, where the line is divided into segments 451 and 452. The image depicts the railway line with an overlay of dots, while the roads are represented in white.

Apart from this station, the majority of errors occur at points of intersection or crossing of lines, as illustrated in Figure 2.9.

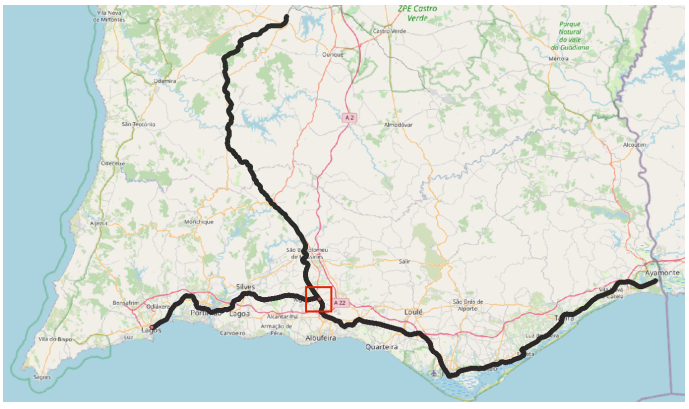


Figure 2.9 Point of intersection between two principal railway lines, namely Linha do Sul and Linha do Algarve.

2.6 Advantages of the method

Random forests generally have high accuracy because they combine the predictions of multiple decision trees, reducing the risk of overfitting compared to individual DTs. They are robust to outliers and noise in the data because the ensemble of trees averages out the impact of noisy instances. RFs can handle high-dimensional data effectively without requiring dimensionality reduction techniques. They provide estimates of feature importance, helping in understanding which features are most influential in making predictions. They can be used for both classification and regression tasks and generally require less parameter tuning compared to other machine learning algorithms like neural networks. [2]

As for artificial neural networks, they are highly effective at identifying complex patterns and relationships within large datasets due to their ability to model non-linear functions. ANNs excel in adaptability, learning directly from data without the need for explicit programming, making them ideal for tasks like classification. Their capacity for parallel processing further enhances their efficiency in solving complex problems. [11]

2.7 Disadvantages of the method

Random forests can be computationally intensive and require significant memory resources, especially with large datasets and many trees. The model can be difficult to interpret compared to a single decision tree and the ensemble of trees in itself makes it hard to visualize and understand the decision process. If the data is imbalanced, RFs might be biased towards the majority class, although this can be mitigated with techniques like balanced random forests or by adjusting class weights and while more trees generally improve accuracy, beyond a certain point they may lead to diminishing returns and potential overfitting.

As for ANNs, they too have several limitations. They require large amounts of data and significant computational resources for training, which can be resource-intensive. ANNs also function as "black boxes", making it difficult to interpret or understand the decision-making process, thus lacking transparency. Additionally, they are prone to overfitting and often require careful tuning of hyperparameters. Training can be time-consuming and may converge to local minima, leading to suboptimal solutions.

2.8 Conclusions

The random forest classifier is a powerful and flexible algorithm that excels in many scenarios, especially with complex datasets. Its robustness, ability to handle high-dimensional data, and feature importance insights are significant advantages. Before applying the RF classifier, the data was subjected to pre-processing using the *StandardScaler* and *MinMaxScaler*. However, this resulted in a deterioration of the results.

Though ANNs have drawbacks, such as their opacity and resource-intensive nature, they are still highly effective for classification tasks. Their ability to learn complex patterns and

make accurate predictions from large datasets outweighs their limitations. [11] Despite requiring careful tuning and large amounts of data, ANNs consistently deliver strong performance.

This method proved particularly efficacious in Linha do Algarve and was also trialled in Linha do Sul, yielding favourable outcomes. Given the existence of zones within the country where the network is more densely populated, it would be beneficial to test the ensemble method and analyse the results meticulously, as they may yield novel or as yet unidentified insights.

Chapter 3

Kilometric Point Identification

Once the line in question has been identified, it is necessary to ascertain the precise location of the train on that line. This can be expressed in terms of the kilometric point (PK) to which the train is assigned. In railway terminology, PKs represent a reference system that is used to indicate the precise location of a train along a given railway line. It is typically expressed in kilometres and metres from a predefined starting point, which is often the beginning of the railway line or a major station.

It should be noted that the lines in question do not necessarily originate at PK 0. This phenomenon can be attributed to the immutable nature of PKs, which function as labels and remain fixed after assignment, regardless of any subsequent modifications to the line. For instance, both segments (451 and 452) from Linha do Algarve begin in Tunes, at PK 301.889 - all stations and their respective PKs are documented in [9]. The segment encompassing the Western Algarve reaches its conclusion at PK 347.210, situated within the city of Lagos, while the segment pertaining to the Eastern Algarve culminates at PK 396.468, located in Vila Real de Santo António, see Figure 2.2 in the previous chapter. The station of Faro, capital of the Algarve subregion, in PK 340.008 is situated 38 km and 119 m to the east of Tunes.

Kilometric markers are utilised for a multitude of purposes, including the delineation of precise locations for maintenance or repair work, the provision of an accurate description of the location of incidents or accidents on the line, and the determination of the location of stations, bridges, tunnels, and other infrastructure. Furthermore, they assist train operators and control centres in the administration of train movements, signalling, and safety protocols. For all the aforementioned purposes, this system is indispensable for maintaining safety in railway operations.

There is an algorithm implemented which identifies the section of the line where the point may be located using geolocation. It then defines a small interval of search and employs a brute-force algorithm or exhaustive search. A brute-force approach typically involves evaluating all possible points within a given search space to identify the optimal solution. This approach may yield a more precise solution, but it requires a greater number of evaluations. The objective is to enhance the performance of this step of the procedure, thereby facilitating a rapid and efficacious execution time.

Idea Behind the Algorithm

Given the volume of data to be processed, it is required that the algorithm be optimized to the greatest extent possible. In order to achieve this, a number of techniques have been employed with the objective of reducing the algorithm's complexity. The fundamental concept underlying the algorithmic approach is as follows:

1. **Separate Data**
   At this stage of the procedure, the *TestLog* file has already been populated with a column that serves to identify the line. In order for the neural network to be applied in the subsequent step, the data from this file must be separated by line.

2. **Predict Approximate Location**
   Each line is associated with a distinct neural network. The application of the ANN will yield an approximate estimate of the PK, which will then dictate the interval of points of search where the closest point is to be found.

3. **Find Closest Point**
   Ultimately, the golden section method is applied with the objective of identifying the point on the rail that is in closest proximity to the train's position. The precise location on the railway will establish the true kilometric point.

Figure 3.1 illustrates the sequence of steps comprising the algorithm.

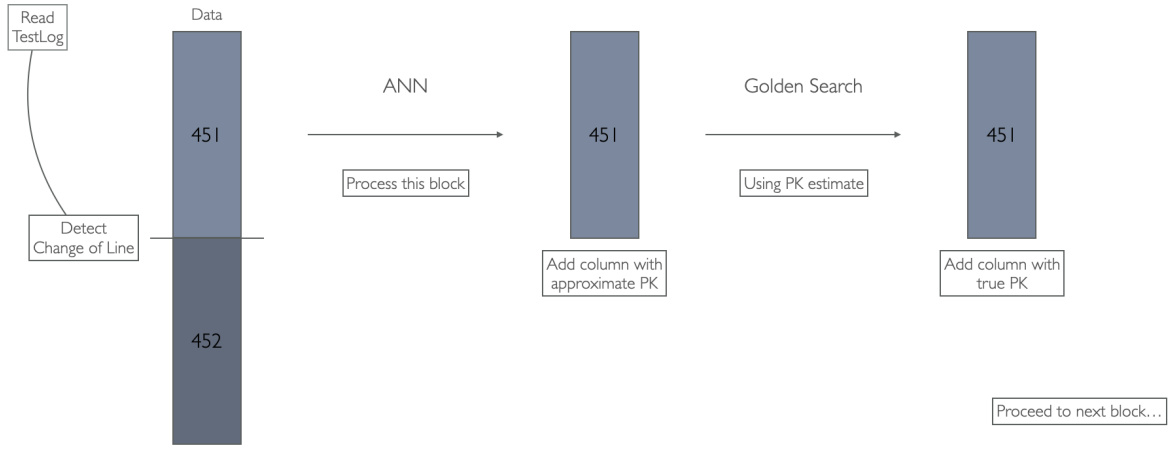


Figure 3.1 Diagram of the algorithm to identify PKs.
Figure by author, 2024.

The following sections provide a detailed account of each methodology, concluding with the manner in which the process is understood to operate.

3.1 Use of Neural Networks for Prediction

As previously stated, the method for identifying the line using the data received from the probe is through the coordinates. The same methodology may be employed to identify the PK. In this context, the utilisation of a neural network serves to facilitate the prediction of the approximate location of the train, thereby reducing the search interval.

In consideration of the disparate segments and the occurrence of multiple repeated PKs within the line mapping (with both segments commencing at PK 301, but concluding at PK 347 and PK 396 in segments 451 and 452, respectively), it is essential to train a neural network for each section. Files **451.csv** and **452.csv** containing the line mapping were incorporated into the training process. In this section, only the latitude, longitude, and PK columns will be utilised, as the other three (PTTM06_X, PTTM06_Y, and ALT) will not be included. The height attribute will not be considered, as the same networks were previously tested using this attribute, resulting in suboptimal outcomes. Tables 3.1 and 3.2 demonstrate a sample of the data employed for the purpose of network training.

Line 451	Data		
	LAT	LON	PK
0	37.165368	-8.256667	301.939
1	37.165376	-8.256674	301.940
2	37.165383	-8.256680	301.941
⋮	⋮	⋮	⋮
45173	37.108031	-8.670817	347.094
45174	37.108036	-8.670826	347.095

Table 3.1 First and last rows of the data set used to train the ANN for segment 451.

Line 452	Data		
	LAT	LON	PK
0	37.164895	-8.256336	301.898
1	37.164887	-8.256329	301.899
2	37.164880	-8.256323	301.900
⋮	⋮	⋮	⋮
94229	37.199457	-7.421850	395.936
94230	37.199463	-7.421841	395.937

Table 3.2 First and last rows of the data set used to train the ANN for segment 452.

Preprocessing

In both files, the unit of measurement of the kilometric points was converted from metres to kilometres, thus ensuring consistency in the data, given that the final output is intended to be in kilometres. Furthermore, the PKs were arranged in ascending order, with the intention of applying the binary search algorithm. A series of tests were conducted on the *StandardScaler* and the *MinMaxScaler*, with the former demonstrating superior performance. With regard to the partitioning of the data for training and testing the neural network, 80% of the observations were allocated to the training set, while the remaining 20% constituted the testing set.

Architecture

The network was trained with varying numbers of layers, including one, two, and three hidden layers. It was determined that a single layer was insufficient for accurate prediction. The application of a third layer yielded results comparable to those obtained with a second layer, albeit with an increased computational burden. In regard to the number of neurons in each layer, the powers of 2 were tested. Table 3.3 presents a breakdown of the number of neurons tested in each layer, the number of trainable parameters in each model, and the maximum error obtained in the prediction with the test set.

	Number of neurons		Total Parameters/	Biggest error	
	1st layer	2nd layer	Treinable Parameters	S 451	S 452
1	4	4	37	0.572	1.030
2	8	4	65	0.486	0.886
3	8	8	105	0.175	0.497
4	16	8	193	0.235	0.241
5	16	16	337	0.115	0.204
6	32	16	641	0.133	0.257
7	32	32	1185	0.107	0.184
8	**64**	**32**	**2305**	**0.062**	**0.135**
9	64	64	4417	0.122	0.182
10	128	64	8705	0.127	0.417
11	128	128	17025	0.124	0.354

Table 3.3 Models tested to ascertain the optimal architectural configuration for the ANNs employed in the prediction of the approximate PK in segments 451 and 452.

It is important to acknowledge that as the number of neurons rises, the maximum error obtained in the prediction declines. However, at a specific point, this trend reverses, indicating that further increases are not beneficial. As can be seen in Table 3.3, the optimum value of the hyperparameters for both segments are 64 neurons in the first layer and 32 in the second layer. A total of 2305 $((2 \times 64 + 64) + (64 \times 32 + 32) + (32 \times 1 + 1))$ parameters were trained.

Layer (type)	Output Shape	Param #
dense_6 (Dense)	(None, 64)	192
dense_7 (Dense)	(None, 32)	2080
dense_8 (Dense)	(None, 1)	33

Total params: 2,305
Trainable params: 2,305
Non-trainable params: 0

Figure 3.2 Architecture of the ANNs employed in the prediction of the approximate PK in segments 451 and 452.

Results

Although Linha do Algarve appears to run almost horizontally across the southern part of the country, fluctuations in latitude and longitude must be taken into account. The graphs in Figure 3.3 illustrate the non-linearity of the line. This renders the accurate prediction of the train's position challenging, due to the inevitable margin of error.

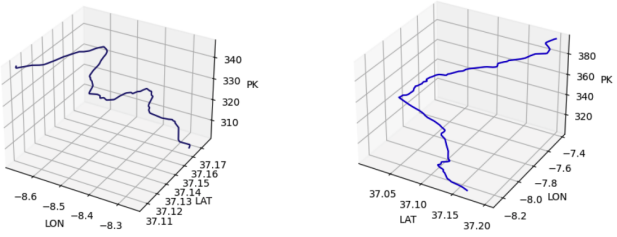


Figure 3.3 3D representations of each section of Linha do Algarve. Section 451 (on the left) and section 452 (on the right).

It is impending that the predictions made by the ANN will be subject to a margin of error. While in some instances the network offers a prediction that is highly accurate, in others it presents errors that are significant and cannot be ignored. In many cases, the observations are not as precise as the points documented, which have a greater number of decimal places. Consequently, the observations do not fall precisely on top of the line, but instead are offset to a slight degree. This phenomenon is illustrated in Figure 3.5. Figure 3.4 shows some observations captured by cells 7A0D and 7B39 that fall on top of the line.

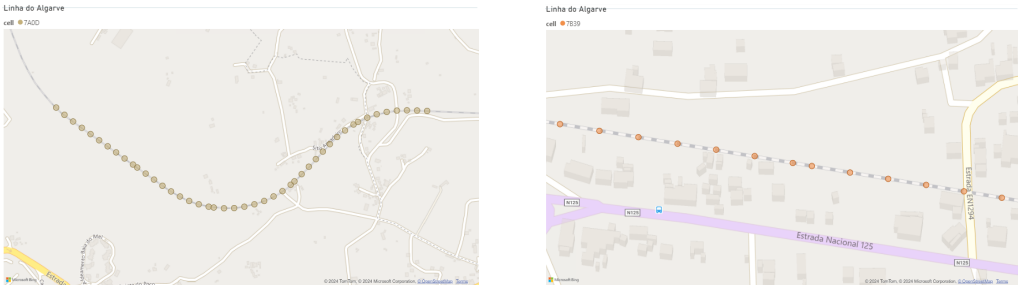


Figure 3.4 Observations captured by cell 7A0D (on the left) and by cell 7B39 (on the right). The image depicts the railway line with an overlay of dots corresponding to one single trip, while the roads are represented in white, yellow or purple.

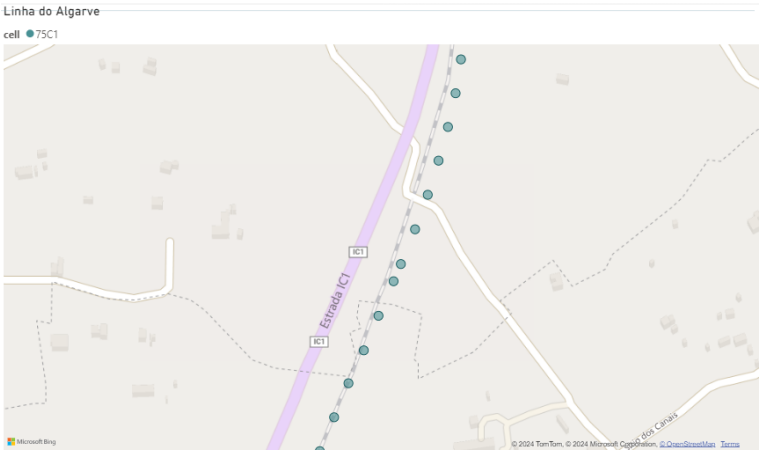


Figure 3.5 Observations captured by cell 75C1 where dots must overlay the railway line but instead exhibit a slight lateral deviation. Some roads are represented in white and the one parallel to the railway is *Estrada IC1*.

It is therefore necessary to define a small range of search, or in other words a specific portion of the entire line where a given point - the PK - is known to exist. In order to calculate this interval, it is essential to consider the most significant inaccuracies in the forecast. The greatest distance observed between the train's position and the forecast provided by the network on segment 451 was 62 metres, while on segment 452 it was 135 metres. To calculate the search range, the value of the forecast is taken and expanded by 200 metres to each side, i.e. the prediction ± 200 metres, thereby allowing for a certain degree of flexibility.

In mathematical terminology, let x_{target} be the value that we are looking for, i.e. the kilometric point indicating the train's position, and x_{pred} the prediction given by the neural network. Then:

$$x_{target} \in [x_{pred} - 0.200, x_{pred} + 0.200] \tag{3.1}$$

$x_{pred}, x_{target} \in \mathbb{R}$, and

$$[x_{pred} - 0.200, x_{pred} + 0.200] \subset [301.889, 347.210] \tag{3.2}$$

for segment 451, and

$$[x_{pred} - 0.200, x_{pred} + 0.200] \subset [301.889, 396.468] \tag{3.3}$$

for segment 452.

Note: For further details, please refer to the chapter introduction on page 19.

3.2 Binary Search

The binary search algorithm, also referred to as logarithmic search, is a search algorithm that is used to identify the position of a target value within a sorted array. Binary search entails a comparison between the target value and the middle element of the array. In the event of an inequality, the half in which the target value cannot be situated is eliminated, and the search is continued on the remaining half. This involves a further comparison between the target value and the middle element, and this process is repeated until the target value is found. Should the search conclude with the remaining half being devoid of elements, it can be inferred that the target value is not present within the array. [17]

In the majority of files, the line is mapped in accordance with a metre-by-metre schedule; however, in some instances, mapping occurs at intervals of two or three metres. In order to align with the requirements of this project, the algorithm has been modified to consistently return the value that is most proximate to the array value in question, for the default in the absence of a value is to return -1. The pseudo-code for the BS is presented below for reference. For an illustration of the modified code in Python, see Appendix B.

Algorithm 1 Binary Search

```
1:  function BinarySearch(array, low, high, target)
2:      if array[high] <= target then
3:          return high
4:      end if
5:      if array[low] > target then
6:          return low
7:      end if
8:      closest index = low
9:      while low ≤ high do
10:         mid ← (low + high) // 2
11:         if array[mid] <= target and (mid == high or array[mid +1] > target) then
12:             return mid
13:         else if array[mid] < target then
14:             closest index = mid
15:             low ← mid + 1
16:         else
17:             high ← mid - 1
18:         end if
19:     end while
20:     return closest value                                    ▷ Target found
21: end function
```

Convergence

In every iteration, the algorithm halves the search space by setting *low* or *high* to the middle index *mid*. The size of the search interval at the beginning is n. After the first iteration, it becomes $\frac{n}{2}$, then $\frac{n}{4}$, and so on. After k iterations, the search space has size $\frac{n}{2^k}$. The algorithm terminates at most when the search space becomes one, or before if said value is found. That is, when

$$\frac{n}{2^k} \geq 1 \tag{3.4}$$

Solving for k, becomes

$$\Leftrightarrow n \geq 2^k \tag{3.5}$$

$$\Leftrightarrow k \leq \log_2 n \tag{3.6}$$

Thus, the algorithm terminates after at most $\log_2 n$ iterations. Therefore, binary search converges, and the number of iterations is bounded by $\log_2 n$, ensuring that the algorithm will terminate after a finite number of steps.

Complexity

The time complexity of an algorithm describes how the running time of the algorithm scales with the size of the input. In each iteration of the BS algorithm, we calculate the middle index, which takes constant time $O(1)$, then compare the middle element with the target, which also takes constant time $O(1)$ and either move to the left or right half of the array (again $O(1)$). As established in the convergence proof, the BS algorithm runs for at most $\log_2 n$ iterations, where n is the size of the input array. Thus, the total time complexity for the worst case [19] is

$$T_{worse}(n) = O \log_2(n) \tag{3.7}$$

In the best case, the target element is found in the first comparison, when *mid* happens to be the target, so the algorithm terminates immediately. The best-case time complexity is

$$T_{best}(n) = O(1) \tag{3.8}$$

On average, assuming the target could be anywhere in the array with equal probability, the time complexity is still dominated by the logarithmic behavior of the binary search. Hence, the average-case time complexity is also

$$T_{average}(n) = O \log_2(n). \tag{3.9}$$

Recursion and Iterativity

The BS algorithm can be implemented in both recursive and iterative ways. The optimal choice depends on the context and requirements of the problem at hand. The recursive approach may be less efficient, as the act of making a recursive call results in the consumption of additional memory on the call stack. Furthermore, there are potential limitations on the maximum depth of recursion that can be achieved, which are imposed by the programming language or system in question. On the contrary, iterative methods are typically more efficient in terms of memory usage and speed, particularly when dealing with large arrays. This approach avoids the overhead associated with function calls and stack space, making it easier to optimise for enhanced performance. In light of the aforementioned considerations, an iterative implementation is to be preferred in instances where large arrays are involved or when optimisation for performance is the objective, given its efficiency. The two approaches were subjected to testing, with the iterative implementation proving to be the fastest of the two.

3.3 Golden Section

The Golden Section algorithm is a technique for finding the extremum, be it the minimum or maximum, of a unimodal function [3].

Definition 1. *(Strictly Unimodal Function). Consider* $f : [0, T] \to \mathbb{R}$. *The function* f *is strictly unimodal on the interval* $[0, T]$ *if it has a unique global minimum* x^* *in* $[0, T]$*, and if the following conditions are verified:*

1. *for each* x_1, x_2 *such that* $x_1 < x_2 < x^*$*, we have* $f(x_1) > f(x_2) > f(x^*)$,
2. *for each* x_1, x_2 *such that* $x^* < x_1 < x_2$*, we have* $f(x^*) < f(x_1) < f(x_2)$,

that is, the function decreases on the left of x^* *and increases on its right.*

It is first necessary to define the function to be minimised and to establish the initial interval $[a, b]$, within which the minimum is known to exist. Subsequently, two internal points, designated as c and d, are calculated within the above interval using the golden ratio. The golden ratio is a constant given by

$$\phi = \frac{1 + \sqrt{5}}{2} \tag{3.10}$$

used to divide the interval in the golden section. The function is evaluated at these points. The interval is narrowed by discarding the portion where the minimum is less likely to be located, depending on the function values at c and d. This process is repeated iteratively, with internal points recalculated and the interval adjusted until it is sufficiently small, indicating that the minimum has been found.

The function f we want to minimize is the distance between the train's position and each point on the railway. The interval $[a, b]$ is the initial search interval where the minimum is known to exist and where a is Tunes and b is Lagos or VRSA, since $a < b$. The tolerance is the precision of the result we mean to achieve, and the search stops when the interval size is smaller than this tolerance. The chosen tolerance was 10^{-3}, since it corresponds to one meter. The code for the algorithm can be found in Appendix B. The pseudo-code is given below and, to better understand it, let:

(x, y) = (train's latitude, train's longitude)

a - Lower end of the search interval

b - Upper end of the search interval

Algorithm 2 Golden Section

function GOLDENSECTION(a, b, x, y, $tol = 10^{-3}$)
  while $|b - a| > tol$ **do**
    $c \leftarrow b - \frac{b-a}{\phi}$
    $d \leftarrow a + \frac{b-a}{\phi}$
    $c_{index} \leftarrow$ BINARYSEARCH(c)
    $d_{index} \leftarrow$ BINARYSEARCH(d)
    $c_{lat} \leftarrow$ latitude(c_{index})
    $c_{lon} \leftarrow$ longitude(c_{index})
    $d_{lat} \leftarrow$ latitude(d_{index})
    $d_{lon} \leftarrow$ longitude(d_{index})
    if NORMSQUARED((x, y), (c_{lat}, c_{lon}) < NORMSQUARED((x, y), (d_{lat}, d_{lon}) **then**
      $b \leftarrow d$
    else
      $a \leftarrow c$
    end if
  end while
  return $\frac{b+a}{2}$
end function

Note: In the code, it is calculated the norm squared, that is (point c was chosen for illustration purposes):

$$(x - c_x)^2 + (y - c_y)^2 \tag{3.11}$$

instead of the norm, or distance, given by

$$\sqrt{(x - c_x)^2 + (y - c_y)^2} \tag{3.12}$$

to reduce the computational effort. This approach avoids the extra step of computing the square root, which is required when calculating the norm itself. Since the square root function is computationally expensive, especially in large-scale calculations or high-dimensional data, working with the norm squared simplifies the process while still preserving the relative relationships between distances. In many applications, the norm squared is sufficient for comparison purposes, making it a more efficient alternative.

Convergence

To prove that the GS algorithm converges to the optimal solution, we need to consider several mathematical properties that establish convergence. In each iteration, the length of the search interval $[a, b]$ is reduced by a factor of Φ (where $\Phi \approx 1.618$, the golden ratio). This geometric reduction ensures that the interval becomes arbitrarily small after enough iterations. Specifically, after each iteration, the length of the interval is multiplied by $\frac{1}{\Phi}$, which is approximately 0.618.

The algorithm continues until the difference $b - a$ is smaller than a given tolerance (tol). Starting with an interval length $L_0 = b - a$, after i iterations, the interval length will be approximately

$$L_n = L_0 \times \left(\frac{1}{\Phi}\right)^i \tag{3.13}$$

As $i \to \infty$, the interval length approaches 0, meaning the algorithm will converge to a single point. The algorithm stops when $L_i \leq tol$, so we can solve for the number of iterations i as:

$$\frac{L_0}{\Phi^i} \leq tol \iff \Phi^i \geq \frac{L_0}{tol} \tag{3.14}$$

Applying the logarithm to both sides of the equation, we have:

$$\log(\Phi^i) \geq \log\left(\frac{L_0}{tol}\right) \quad \Leftrightarrow \quad i \times \log(\Phi) \geq \log\left(\frac{L_0}{tol}\right) \tag{3.15}$$

$$\Leftrightarrow \quad i \geq \frac{\log\left(\frac{L_0}{tol}\right)}{\log(\Phi)} \tag{3.16}$$

This guarantees that, after a finite number of iterations, the algorithm will stop and provide an answer within the desired tolerance. The final interval contains the minimum. Because the function evaluations at c and d (the inner points of the interval) progressively discard the suboptimal half of the interval, the minimum point is contained within the final interval of length tol. The method doesn't miss the minimum because it discards only the half of the interval that is guaranteed to be suboptimal based on the function evaluations. Thus, proving that the GS algorithm converges to the optimal solution.

Complexity

The complexity of the algorithm can be analysed based on its iterative nature and how it reduces the search interval. Since each iteration involves a constant amount of evaluating a few function values and comparing norms, and the number of iterations grows logarithmically with the inverse of the tolerance, the overall time complexity is

$$O \log\left(\frac{L_0}{tol}\right). \tag{3.17}$$

3.4 Calculating PKs

This section explains the operation of the algorithm by means of the methods described above. It initially reads the *TestLog* file, which holds the data that came from the probe. This file contains 90 columns, the final of which was added later and serves to identify the line (topic from the preceding chapter). Upon the detection of a line change, for instance from 451 to 452, the processing of all observations up to that point initiates. The *StandardScaler* is applied to the income variables, and the forecast (given by the ANN) is generated and stored. In employing these forecasts, the search range is calculated. To this smaller portion of line, the golden section method is applied, giving us the train's position, that is the final PK, thus ending the search. The time required to search for a specific value within the file is reduced greatly by using the binary search method.

In order to further reduce the processing time, a condition is implemented which verifies whether the preceding observation exhibits identical latitudinal and longitudinal coordinates as the current observation. If affirmative, then the same PK should be given. This approach obviates the necessity for recalculation in the case of a substantial number of repeated rows, such as those occurring when the train is stopped at a station. The implementation of this modification resulted in a reduction in computing time in most instances, with a maximum decrease of 18%.

3.5 Results

With regard to the final results in comparison to the previous algorithm, let us consider Figure 3.6. It demonstrates the distance between 50 observations of a given file and the attributed PKs. This example is presented to illustrate that there are instances when the previous algorithm (brute-force approach) exhibits a slightly closer distance, other instances where the new algorithm is a bit closer (ANN + GS method), and a considerable number of instances where the calculated point is identical.

Note: The distance shown on the y-axis was calculated using the coordinates - latitude and longitude.

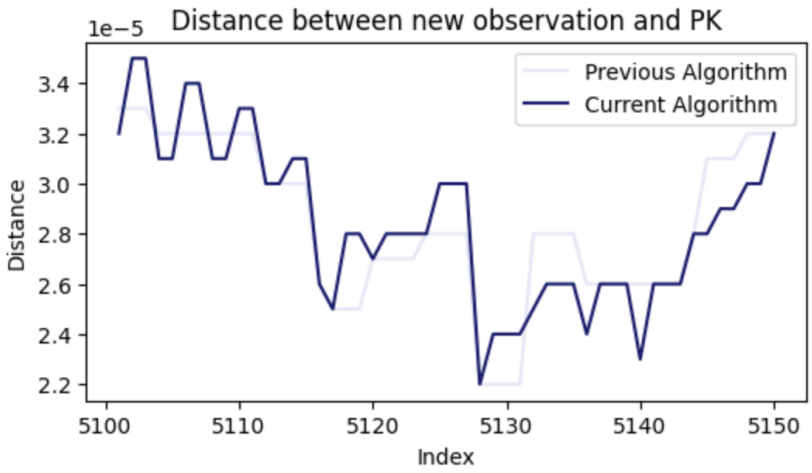


Figure 3.6 Distance between the train's position and the point (PK) on the railway.

3.6 Conclusions

This algorithm was developed from the ground up with the objective of achieving the highest possible levels of both speed and precision. While the training of ANNs may be a time-consuming process, once trained, they offer a rapid means of making predictions. As the initial algorithm is not sufficiently accurate, a second component has been incorporated to identify the precise value of PK.

The algorithm was significantly improved through the incorporation of logarithmic convergence towards the desired precision, derived from algorithms such as the binary search and golden section, which employ iterative reduction. The two algorithms in question converge, and this convergence can be demonstrated through the use of three key principles: the geometric reduction of the interval, the preservation of the optimal point, and the stopping condition. The BS is invoked on multiple occasions when a target value is required for a large file. It is important to note that when attempting to replicate the process with other lines, it is essential to ensure that the files are in ascending order of PKs. Lines 11, 12, 61, 81, 91, 201, 221, 231, 252, 292, 331, 341, 371, 374 and 461 do not comply with this requirement.

The method has been tested and implemented with very satisfactory results. The processing time for a million instances was previously approximately seven hours; the current processing time for the same volume of data is approximately one hour, representing an 85% reduction in processing time.

Chapter 4

Direction and Journey Identification

As previously stated, the probe is unable to provide information regarding the position of the train. In point of fact, the probe is similarly unable to provide information regarding the direction of the train's movement. After processing the *TestLog* files and adding columns with information about the line and PK, it can be surmised that if the PKs are increasing in the direction of travel, the train is moving upwards and we may conclude that the direction is ascendant. Conversely, if the PKs are decreasing, it can be inferred that the train is moving downwards, and thus the direction is descendent. This way, the direction can be readily determined, as can the end of each journey and the beginning of a new one.

Henceforth, the kilometric points will be rounded to the first decimal spot. This means that from now on, in this chapter and the subsequent one, the railway points will no longer be recorded in metre increments; instead, they will be aggregated at 100-metre intervals. To illustrate, all points from PK 302.050 to PK 302.149 are now designated as PK 302.1.

The rationale for this decision is based on the premise that processing the signal on a metered basis is challenging, in terms of both time and computational expense, and unnecessary, given that the objective is to detect significant signal deviations by examining the entire line holistically, rather than focusing on a single segment with a single meter. The paragraph on the EIRENE System Requirements Specification that provides the grounds for this argument is presented in Figure 4.1. For further details, see reference [10].

3.2.5 The specified coverage probability means that with a probability value of at least 95% in each location interval (length: 100m) the measured coverage level shall be greater than or equal to the figures stated above. The coverage levels specified above consider a maximum total loss of 6 dB between antenna and receiver inputs (including a margin of e.g. 3dB for ageing). This value is defined in order to ensure that the level at the input of the receiver will never be lower than the reference sensitivity level of the receiver. The relevant clauses according to the appropriate frequency band defined in [EN 301 515, Index [35]] should apply. It is possible to compensate losses higher than 3 dB between antenna and receiver inputs by using an antenna with the corresponding gain above the assumed value of 0dBi (I).

Figure 4.1 EIRENE System Requirements Specification Version 16.0.0.

In light of the above, all PKs are now rounded, with the second and third decimal

places being disregarded. The data will now be employed to determine the trajectory of the train. Figure 4.2 illustrates an exemplar journey occurring within segment 451. The x-axis represents the temporal dimension, whereas the y-axis represents the PKs. At the outset of the journey, the train is situated in Lagos (PK 347). Then, the train proceeds along the railway until reaching Tunes (PK 301). During the journey, there are instances where the train is stationary, indicated by horizontal lines. These correspond to stations. In the illustration, a prolonged halt is evident at PK 330. This is due to the train's stop at Portimão (PK 330.281), an important station with high traffic volume.

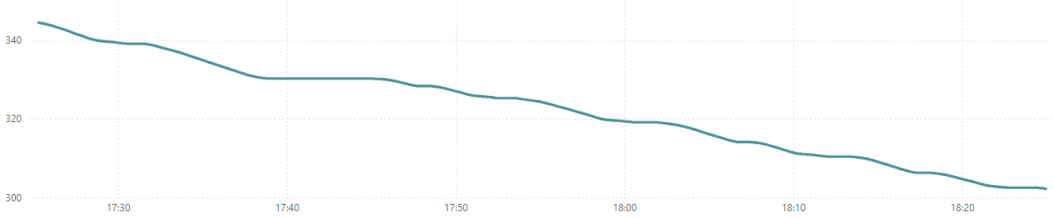


Figure 4.2 Example of a journey on segment 451 in the descending direction.

In relation to the same journey, Figure 4.3 shows the signal levels across the line, separated by cell. It can be observed that the train starts its journey at a higher PK and subsequently moves down the line, descending. This is evidenced by the fact that, when the figure is viewed from right to left, the signal continues to decrease until it reaches its lowest point, after which there is an abrupt rise in signal levels when the probe connects to the next cell. This is also illustrated in Figure 4.4, which depicts the mean signal levels from the same journey.

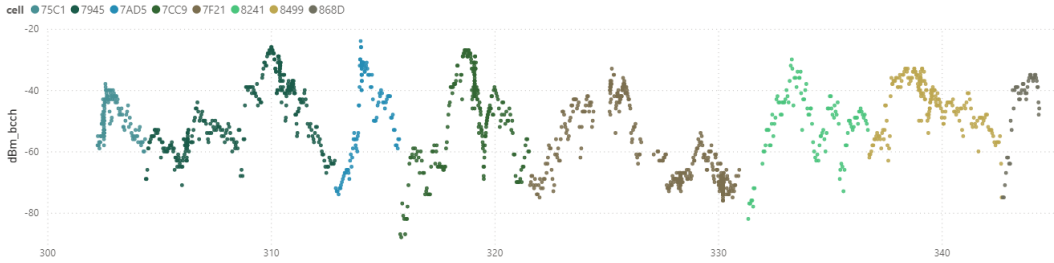


Figure 4.3 Signal levels along the line, divided by cell.

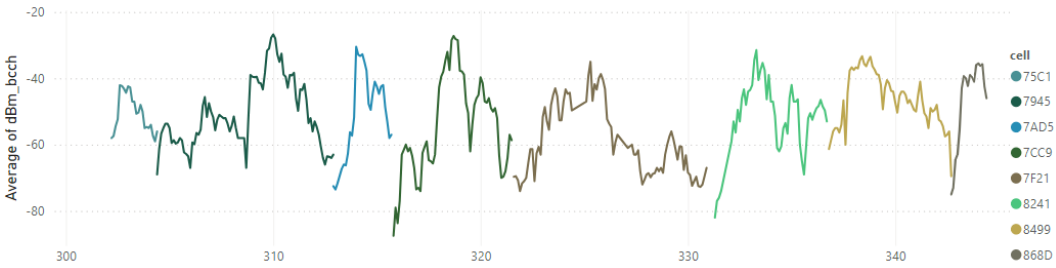


Figure 4.4 Mean of signal levels along the line, divided by cell.

The three images above illustrate a file comprising a single journey, which is a relatively uncommon occurrence. It is evident that the signal levels for this particular journey can

be displayed with clarity in the accompanying graphs. However, it is typical for each file to comprise a number of journeys and in such instances the data from the various journeys overlap, rendering them indistinguishable. Furthermore, it should be noted that PKs are repeated up to a specific value. Consequently, the data from the cells up to approximately PK 347 also overlap, as illustrated in Figure 4.5.

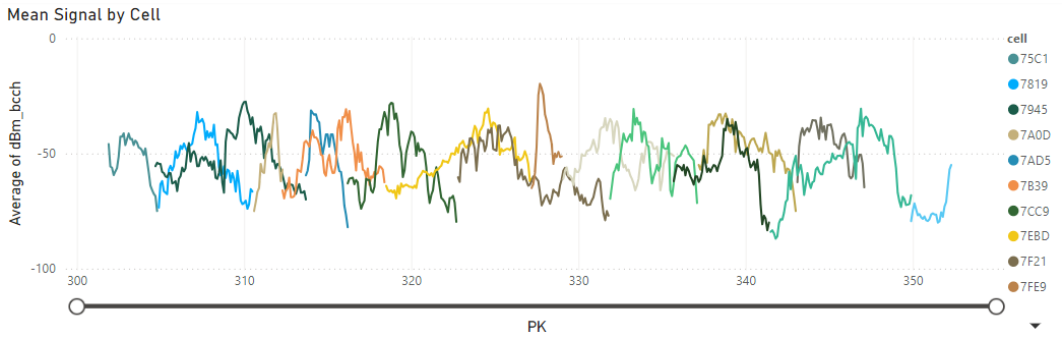


Figure 4.5 Mean of signal levels along the line from various journeys.

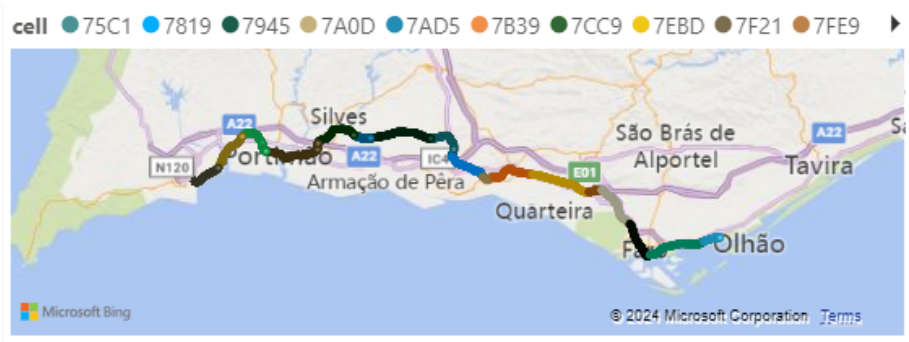


Figure 4.6 Map representation of journeys in the figure above.

As this file contains journeys in both segments, in the part where there are repeated PKs the cells overlap. For example, in Figure 4.5 we observe that cell 7819 in blue overlaps with cell 7945 in green, which in turn overlaps with cells 7A0D and 7B39, in beige and orange respectively.

4.1 Determining Directions

To find the direction in which the train is going, we need to take a look at the first part of the assignment, where PKs were calcuted. By seeing how the train is moving through the line, we can perceive if in terms of the kilometric points the train is ascending or descending down the line.

The *set_ direction* function, as illustrated in Appendix C, simply calculates the difference between two consecutive observations in order to determine whether the values are increasing or decreasing. The difference is calculated between two consecutive observations and the function either returns immediately or, if the difference is 0 (the next PK is equal to the current one), iterates until the values are no longer equal. Below is the pseudo code.

Algorithm 3 Set Direction

```
1: function SetDirection(values, p1, p2)
2:     PK1 ← values[p1]
3:     PK2 ← values[p2]
4:     difference ← PK2 - PK1
5:     if difference > 0 then
6:         return True                                   ▷ Is increasing
7:     else if difference < 0 then
8:         return False                                  ▷ Is decreasing
9:     end if
10:    i ← p2
11:    while i < Size of values - 1 and PKi = PKi−1 do
12:        i ← i + 1
13:    end while
14:    if i < Size of values and PKi > PKi−1 then
15:        return True                                   ▷ Increasing
16:    else
17:        return False                                  ▷ Decreasing
18:    end if
19: end function
```

The direction is then appended at the same time as the journey number, and the function is invoked on multiple occasions throughout the execution of the code to set the direction whenever there is a new journey.

4.2 Isolating Journeys

Separating data in a file into individual train journeys is crucial for both signal analysis and anomaly detection. By isolating each journey, it becomes easier to pinpoint and analyze specific operational conditions, such as speed, acceleration, braking patterns, and external factors (e.g., weather or track conditions). This focused analysis allows for more accurate signal processing, reducing noise and variability from unrelated journeys.

Signal data, such as vibrations, temperatures, and electrical currents, vary throughout different stages of a journey (e.g., acceleration, cruising, braking). Analyzing one journey at a time enables better detection of anomalies or irregularities specific to a particular segment. This can help in identifying patterns related to mechanical wear, electrical failures, or environmental impacts that may not be noticeable across multiple journeys lumped together.

Regarding maintenance strategies, the ability to link detected signal anomalies to specific journeys enables more targeted interventions. By knowing which part of a journey or which conditions triggered specific signals, maintenance teams can focus efforts on certain components or track sections. This not only improves maintenance efficiency but also helps prevent

over-maintenance or neglect of critical issues, leading to better resource allocation and predictive maintenance. Separating journeys also helps track the progression of specific faults or performance issues over time (crucial for the algorithm of generation of KPIs). This way, it is possible to monitor how a signal evolves across different journeys, helping in predicting when a component may fail and scheduling timely interventions before breakdowns occur.

Figure 4.7 shows an example of the graphical representation of journeys within a file before labelling. The dots illustrate the signal levels across the line, but from three different journeys and it is not possible to distinguish which belong to which journey. This example was also chosen to show that not all journeys cover the whole line. In fact, the dataset is very irregular in terms of the number of observations on a given trip, the number of trips in a given file, the periodicity of entries, handover points, etc.

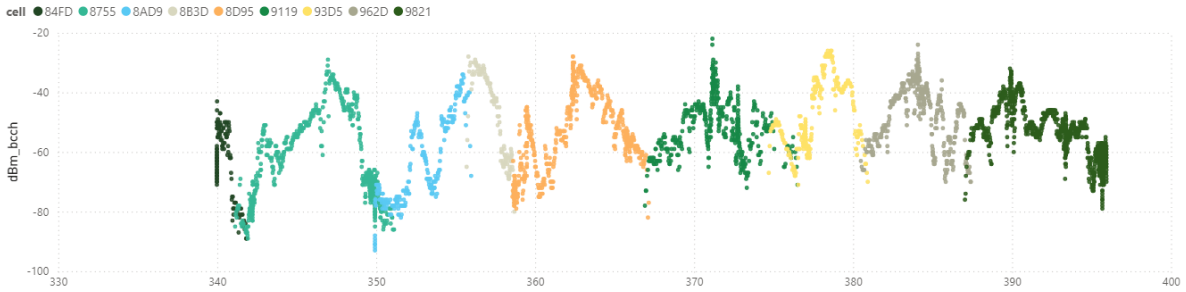


Figure 4.7 Signal levels from several journeys in the same file.

In order to provide comprehensive coverage, three conditions have been identified as defining a change in journey. These are:

1. **Change of direction**
   If the direction shifts, i.e. if there is a change in the monotony of the kilometric points, the current journey is stopped and a new one is started;

2. **No data emitted**
   The probe is disconnected and does not emit data for a pre-defined period of time (in the function's arguments there is a parameter - *step=pd.Timedelta(minutes=15)* - which sets this period of time);

3. **Train is stopped**
   In the event that the probe is connected and the train is emmiting data but does not move its position more than a pre-defined distance, the journey ends (this distance is configurable and is set to 1 km).

4.2.1 Labeling Journeys

Figure 4.8 shows the data from a given file. Same as Figure 4.2, the x-axis represents the temporal dimension and the y-axis the PKs.

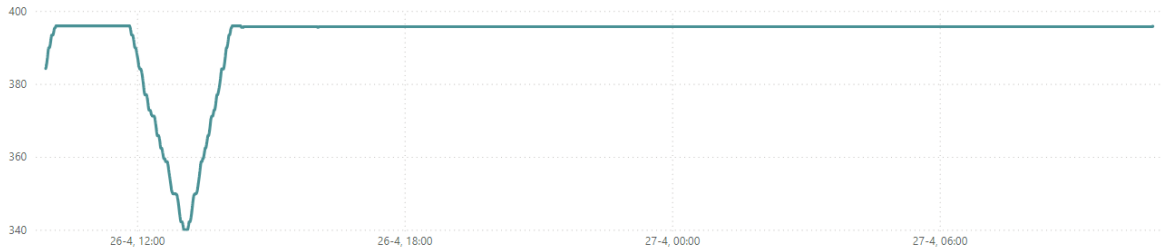


Figure 4.8 Several journeys in the same file.

Looking at the image, it is intuitive to say that the file contains three journeys. The first trip does not cover the whole line. A possible reason for this is that the probe was switched on in the middle of the journey. Nevertheless, it should be considered as one trip. The second and third journeys also do not cover the whole line, but it is perceivable that one is descending and the other ascending, respectively. Figure 4.9 shows how the journeys in this file should be labelled.

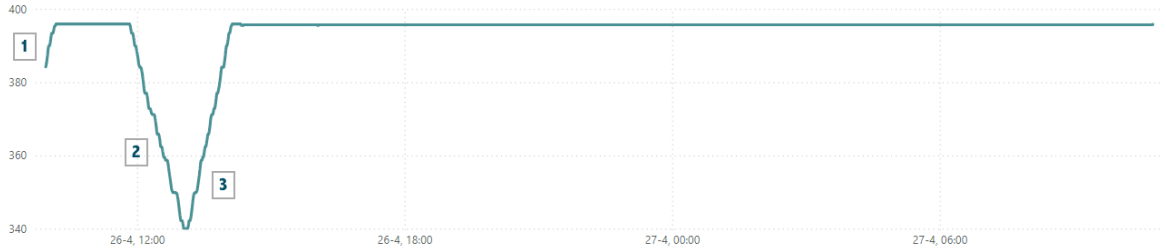


Figure 4.9 Correct label given to the journeys in the same file.

However, due to the small deviations that sometimes occur when assigning PKs, there is a change in the monotony and thus an additional trip. Figure 4.10 shows the same image, but zoomed in. The red circles indicate zones where this happens. To correct this, there is a condition that discards journeys that have been assigned but where the train has not moved more than a certain distance - this distance is configurable in the algorithm, for now it is set at $1km$.

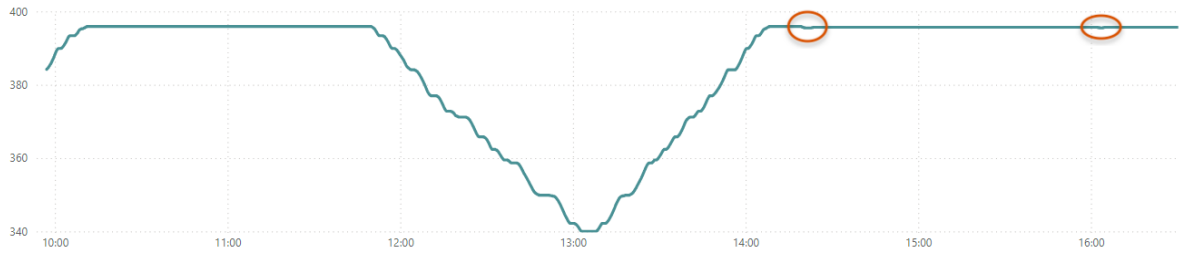


Figure 4.10 Slight variations in PKs that cause errors when labeling journeys.

This chapter is of significant importance for the next one, as the algorithm will be populated with data from each new trip undertaken.

4.3 Conclusions

The algorithm for separating journeys within a file offers several key advantages, particularly in terms of data management and analytical precision. By isolating individual journeys, the algorithm facilitates a more organized and structured dataset, which enhances clarity and reduces the potential for data overlap or misinterpretation. This compartmentalization allows for targeted analysis of each journey, improving the accuracy of performance metrics and trend identification.

Furthermore, the algorithm enhances scalability, as it enables the efficient handling of large datasets by breaking them into smaller, more manageable components. This promotes more effective analysis, reducing computational complexity. Additionally, the ability to separate journeys supports comparative studies, where multiple journeys can be evaluated independently or in relation to each other, fostering deeper insights.

The algorithm's systematic approach contributes to the optimization of both data processing workflows and analytical outcomes, making it a valuable tool for operational purposes. In sum, separating journeys allows for precise, journey-specific signal analysis, improving fault detection, optimizing maintenance schedules, and ultimately enhancing the safety and reliability of train operations.

Chapter 5

Generation of Key Performance Indicators

The popularity of railway transportation has been rising over the past decades, as it provides safe and reliable service. With growing urbanization and increasing environmental concerns, railways have become a crucial component of modern infrastructure, offering an efficient and sustainable alternative to road and air transport. The rise in rail activity, however, presents new challenges in terms of monitoring and managing train operations. Ensuring punctuality, safety, and efficiency across complex rail networks requires advanced systems capable of monitoring and at times decision-making. This chapter explores the development of an algorithm designed to optimize the oversight of train activity, enhancing operational performance and safety.

The objective of this part is to develop a model capable of detecting significant variations in railway signal data through a comprehensive signal analysis. The proposed algorithm will be designed to identify anomalies in signal behavior, assess issues related to signal quality, and accurately detect handover events between different sites or cells. By analyzing fluctuations in signal strength and patterns, the model aims to enhance the reliability of railway communication networks, which are essential for safe and efficient train operations. This approach will not only focus on anomaly detection but also on ensuring the integrity of the data used for critical decision-making in railway traffic management. Additionally, the algorithm will provide insights into the root causes of signal disruptions, facilitating proactive maintenance and minimizing the risk of operational failures.

To achieve this, the data structure selected for implementation is the dictionary. Dictionaries offer an efficient means of managing one-to-one relationships, as they allow for dynamic and flexible handling of data [13]. With constant average time complexity $O(1)$ for lookups, insertions, and deletions, dictionaries are well-suited for scenarios requiring rapid updates to key-value pairs. This efficiency makes them particularly advantageous in these algorithms, where quick access and modification of entries are crucial for maintaining operational integrity. Additionally, dictionaries support the scalability needed for large datasets, enabling the seamless addition, updating, or removal of relationships without compromising perfor-

mance. This choice of data structure ensures that the algorithm can process signals effectively while maintaining optimal computational efficiency.

5.1 Generating Alarms

Figure 5.1 illustrates data extracted from a file that contained several journeys. It is detailing one of the train's journeys, during which the train failed to connect to the appropriate cell in the network. Instead of transitioning to the next cell as the train moved along its route, the train remained connected to the same cell. This misalignment between the train's movement and the expected cell handovers led to a progressively declining signal strength, as the train moved farther away from the original cell's coverage area. The figure highlights how this failure to switch cells in a timely manner can result in deteriorating signal quality, impacting the reliability of the communication system.

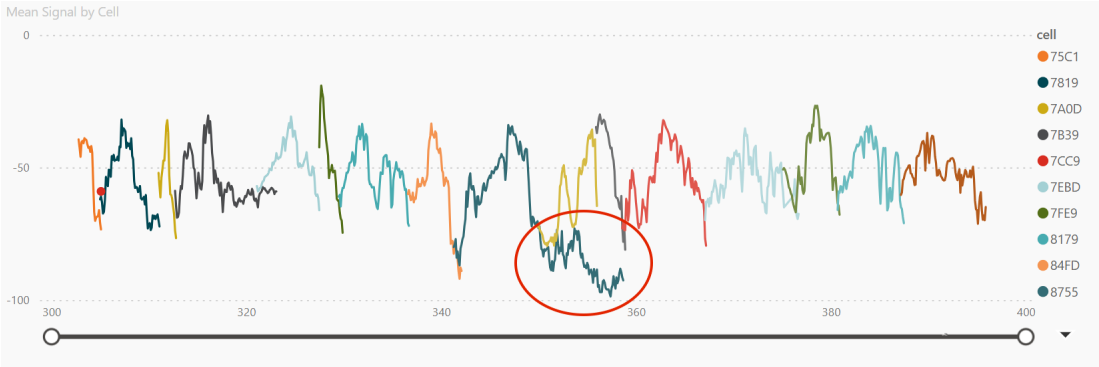


Figure 5.1 The probe was unable to establish a connection with the intended site.

The algorithm is designed to detect anomalies and generate a detailed message specifying the origin file, the particular journey, and the cells involved. Additionally, it identifies and displays the specific instances that triggered the alert. The message should also indicate the underlying reason for the alarm, distinguishing whether the anomaly originated from signal irregularities or from handover issues between cells. This comprehensive output enhances traceability and allows for precise diagnosis of operational disruptions.

In order to enhance the code's intuitiveness and readability, it has been organised into classes. In addition to the main class, the algorithm comprises three further classes. These are:

1. Class **Dictionary**
2. Class **Signal**
3. Class **Handover**

The use of classes offers several key advantages that enhance organization and scalability. First, classes encapsulate data and methods, allowing control over access and modification

while promoting abstraction. They also support code reusability, making it easy to reuse and extend functionalities across different projects. By breaking down complex problems into modular units, classes improve code organization and enhance maintainability by promoting a clear separation of responsibilities, making it easier to debug and scale applications. Finally, classes combine state and behavior, keeping related data and actions together for better organization. The following section provides an explanation of the functionality of each method inside the classes.

5.2 Class Dictionary

The **Dictionary** class encompasses all functions related to the manipulation of dictionaries. This includes operations such as dictionary creation, loading existing data, adding new information, and saving the updated content. These functions collectively manage the lifecycle and integrity of the dictionary data.

5.2.1 Methods

Function *create*

This function is to be invoked with a single argument, namely a letter. The letter 'S' is indicative of signal, while the letter 'H' is indicative of handover. This initiates the creation of the file with a key-value pair for storing data. The key must be unique within a dictionary while the value may not be. The values in a dictionary can be of any data type and can be duplicated, whereas the keys must be of an immutable data type such as strings, numbers, or tuples [13]. Here, the keys are the PKs.

Function *load*

This function is responsible for the loading of dictionaries. In order to engage with the consultation of entries, whether for the purpose of adding, removing or substituting data, it is first necessary to open the dictionary in question.

Function *save*

Once modifications have been made to a specific dictionary, it is imperative to save the dictionary in question, as failure to do so will result in the loss of said modifications. This function is responsible for the saving of files in *pickle* format, where you can easily save different variables and load them back in a different Python session, recovering data exactly the way it was without having to edit code [18].

Function *add*

Once the new information has been received, processed and filtered, the relevant information is to be added to the dictionary. This function adds the new piece of information.

5.3 Class Signal

5.3.1 Unit

The unit dBm (decibel-milliwatts) is used to measure power levels in relation to 1 milliwatt (mW). It expresses power logarithmically, allowing for a simpler comparison of large or small power values. The formula for dBm is

$$dBm = 10 \times \log_{10}\left(\frac{P}{1mW}\right) \tag{5.1}$$

where P is the power in milliwatts. In this scale, 0 dBm equals 1 mW, with positive values representing power greater than 1 mW and negative values representing power below 1 mW. The dBm scale is widely used in TLC due to its convenience in representing low power levels typical of wireless systems.

5.3.2 Application

In railway transportation, dBm is essential for managing wireless communication systems such as train control systems (e.g., GSM-R) and other communication-based train control systems. These systems rely on the transmission of radio signals to ensure safe and efficient train operations. dBm is used to measure and manage the signal strength between trains and control centers, ensuring that the communication links remain robust over long distances and through complex environments like tunnels or dense urban areas. By monitoring signal strength in dBm, railway operators can maintain reliable data transmission for train location, speed control, and signaling, which is critical for safety and operational efficiency.

The minimum allowable signal strength is approximately -95 dBm, although this value may vary depending on the specific zone. To accommodate different areas and operational conditions, this threshold (*min_signal*) is configurable. This flexibility allows for adjustments to the signal strength criteria based on geographic or environmental factors that may influence signal reception quality in different zones.

Additionally, alongside signal strength, signal quality is monitored using a parameter known as *rxQual*, which serves as an indicator of the error rate in signal transmission and reception. The *rxQual* value ranges between 0 and 7, with lower values indicating better signal quality and higher values signifying increasing levels of error or degradation. For the purposes of this analysis, *rxQual* values of 6 and 7 are considered problematic, as they suggest a significant level of interference or error that may compromise the reliability of the signal.

In practice, during the assessment of a journey's signal quality, both *min_signal* and *rxQual* are evaluated. If the signal strength falls below the configured *min_signal* threshold, or if *rxQual* exceeds the acceptable range (i.e., values of 6 or 7), the journey is flagged for potential signal issues. This dual-layered approach ensures that both the strength and quality of the signal are maintained within operational standards.

5.3.3 Methods

Function *populate*

This function is tasked with populating the signal dictionary with information regarding the signal levels. The initial state of this dictionary comprises the data obtained from the files present within the folder designated $Train$. These files were selected from a number of potential candidates on the basis of their perceived quality. From there, we can evaluate the signal with a statistical approach.

Function *raise_ alarm*

The following function is employed to test the new instances derived from the files contained within the folder designated as New. In the event that all is well, there is no requirement to issue an alert. On the contrary, in the event of irregularities in the data, the function will return a Boolean value of $True$ and the array containing the faulty instances.

Function *include*

The categorisation of new journeys as either valid or invalid is determined based on the statistical analysis of the initial set of journeys. Once the new information is processed and validated, the relevant piece is to be included to the dictionary. This function updates the values in the key-value pairs.

5.4 Class Handover

In the context of telecommunications, the term *handover* is used to describe the process of transferring an ongoing call or data session from one cell or site to another as a mobile device moves through the coverage area. This is a vital function in guaranteeing uninterrupted connectivity during periods of movement, such as when travelling by train.

In the signal function, each observation is assessed to determine whether it falls within predefined limits. If an observation meets these criteria, it is integrated into the existing dataset. In contrast, during the handover process, all data points are incorporated regardless of whether the handover occurred before or after the most probable point of transfer. This method allows for the analysis of the handover behavior over time, enabling a more accurate estimation of the process dynamics. As a result, information about the handover can be continually updated and refined at specific moments, improving the understanding of its temporal patterns.

For every c cells on a given line, there are $c - 1$ handover points. However, as each journey is distinct and may encompass data from the entire line or a specific portion thereof, it follows that not all of them retain information about the $c-1$ handovers. If h is the number of handovers that occur during a specified journey, then

$$h = 0, 1, \ldots, c - 1, \quad h \in \mathbb{N}.$$

5.4.1 Methods

timestamp_difference

This function is used to calculate the time elapsed between each observation and the first in that file, thereby maintaining a record of the passage of time.

populate

This function is tasked with processing of all files within the designated *Train* folder, which contains files pertaining to valid journeys, and populating the handover dictionary. In this case, all handover points are added.

raise_alarm

The following function is employed to check whether the handover points derived from the files contained within the folder designated as *New* are occurring in the appropriate location. In the event that no irregularities are present, no alert is required. Conversely, in the event of irregularities in the handover - occurring before or after the expected - the function will return a Boolean value of $True$ and the faulty instances.

include

In contrast to the aforementioned signal, all instances are included when discussing the handover. The objective is to obtain a reliable characterisation of the handover zones by recording all journeys.

plot

In order to arrange the data in a manner that facilitates its display in histogram format, this method contains the entire mechanism that underlies the subsequent, **visualize**.

visualize

It enables the user to observe the location and distribution of handover points in the form of a histogram.

5.5 Algorithm

5.5.1 Modes

The algorithm is comprised of two distinct modes.

Mode 1

The initial mode starts with the construction of the data structure, which serves to retain the information. This entails the initialisation of the dictionaries pertaining to both the signal and the handover. Subsequently, these dictionaries are populated with the journeys contained within the folder *Train*, which were previously selected with regard to typical signal levels. This selection was based on empirical grounds, due to the absence of any pre-established criteria.

```
Dictionary S451_ASCENDENT.pkl created.
Dictionary S451_DESCENDENT.pkl created.
Dictionary S452_ASCENDENT.pkl created.
Dictionary S452_DESCENDENT.pkl created.
```

```
Dictionary H451_ASCENDENT.pkl created.
Dictionary H451_DESCENDENT.pkl created.
Dictionary H452_ASCENDENT.pkl created.
Dictionary H452_DESCENDENT.pkl created.
```

Figure 5.2 The prints exhibited when dictionaries are created demonstrate whether the file is for signal/handover, as well as the line and direction.

Mode 2

Now that we have established what is expected to be the standard behavior of the signal, we can proceed to compare the signal data from new journeys against the predefined parameters. For each new journey, the signal will be analyzed to determine whether it falls within the established range, defined as the mean of the signal $\pm$ 4 standard deviations. For lack of a criterion, this interval was defined on the basis that, in a normal distribution, it would encompass 99.9% of the observations within said PK. So, this interval is meant to represent the expected behavior of the signal, encapsulating the normal fluctuations observed during a valid journey.

The process involves fetching the signal of the new journey and comparing it point-by-point with the predefined range. If it falls within the acceptable range ($\mu_{PK} \pm 4\sigma_{PK}$), it is considered valid at that specific point. Conversely, if the signal deviates beyond this range, it is flagged as an anomaly.

To further refine the validation process, a configurable parameter (*num_ occurrences*) is introduced. This parameter defines the maximum allowable number of deviations outside the $\mu_{PK} \pm 4\sigma_{PK}$ range for a journey to remain valid. If the number of occurrences where the signal falls outside this range exceeds the *num_ occurrences* threshold, the journey is deemed invalid. This approach ensures that the validation process is both robust and flexible, allowing for minor, acceptable deviations in signal behavior while still flagging journeys that exhibit significant irregularities.

5.5.2 Outputs

While the program is operational, a message is displayed on the screen, stating:

- The file being processed and the number of journeys within that file;

- The journey being processed;

- The railway line;

- The direction of the train's movement.

Figure 5.3 shows that the file being processed has 6 journeys, where journey number 1 happened in line 452 in the direction of VRSA.

```
Journeys in file scanData_11_line_journey.csv: 6

Journey 1
Line 452
Direction ASCENDENT
```

Figure 5.3 Information displayed while running the algorithm.

In the event of an invalid journey, it is possible to display a detailed account of the number of entries and the specific entries that led to this outcome, as seen in Figure 5.4.

```
Journeys in file scanData_11_line_journey.csv: 6

Journey 1
Line 452
Direction ASCENDENT
Journey 1 not added.
View? (yes/no): yes
Alarm (signal):
      obuId  testplanId  testlogId  ...   date_time_diff  new_journey  new_direction
198       2          16         11  ...  0 days 00:02:08            1      ASCENDENT
201       2          16         11  ...  0 days 00:02:10            1      ASCENDENT
204       2          16         11  ...  0 days 00:02:11            1      ASCENDENT
237       2          16         11  ...  0 days 00:02:32            1      ASCENDENT
240       2          16         11  ...  0 days 00:02:34            1      ASCENDENT
261       2          16         11  ...  0 days 00:02:47            1      ASCENDENT
672       2          16         11  ...  0 days 00:07:06            1      ASCENDENT
693       2          16         11  ...  0 days 00:07:20            1      ASCENDENT
1908      2          16         11  ...  0 days 00:20:20            1      ASCENDENT
2379      2          16         11  ...  0 days 00:25:20            1      ASCENDENT
2382      2          16         11  ...  0 days 00:25:22            1      ASCENDENT
2385      2          16         11  ...  0 days 00:25:23            1      ASCENDENT
2388      2          16         11  ...  0 days 00:25:25            1      ASCENDENT
2391      2          16         11  ...  0 days 00:25:27            1      ASCENDENT
2394      2          16         11  ...  0 days 00:25:29            1      ASCENDENT
2397      2          16         11  ...  0 days 00:25:31            1      ASCENDENT
2400      2          16         11  ...  0 days 00:25:33            1      ASCENDENT
2403      2          16         11  ...  0 days 00:25:35            1      ASCENDENT
2412      2          16         11  ...  0 days 00:25:40            1      ASCENDENT
2631      2          16         11  ...  0 days 00:27:59            1      ASCENDENT
3136      2          16         11  ...  0 days 00:33:23            1      ASCENDENT

[21 rows x 39 columns]
Alarm (error):
Empty DataFrame
```

Figure 5.4 Example of an invalid journey.

In the event that the user has no interest in viewing the faulty observations, a message is presented stating: *"Not displaying"*.

```
Journey 1 not added.
View? (yes/no): no
Not displaying.
Handover happened out of time 5 time(s).
View? (yes/no): no
Not displaying.
```

Figure 5.5 Output from the same journey when the choice to view is set to no.

Handover

With the handover process, the approach to analysis differs slightly. In this case, all handover points, the locations where transitions or transfers occur, are tracked and recorded

in the dictionary. The keys of the dictionaries themselves remain the PKs and each key holds a dictionary inside. The smallest dictionary contains either zero, one or two keys, which are the cell names. The key-value pair represents the number of times that the last observation of one cell is recorded, or the first observation of the next cell is recorded. Letters S and E are used to identify the start or end of the cell, respectively.

From here, the handover point with the highest frequency is identified. This location is crucial, as it represents the most commonly observed handover point and, therefore, serves as a reference for future comparisons. Once this dominant handover point is established, an allowable tolerance range of ± 100 meters is defined around it. This range forms the acceptable boundary for handovers, ensuring that minor variations in the exact handover location are accounted for.

```
Journeys in file scanData_24_line_journey.csv: [1, 2, 3, 4, 5]
Journey 1 processed.
Journey 2 processed.
Journey 3 processed.
Journey 4 processed.
Journey 5 processed.
```

Figure 5.6 Message exhibited after journeys are processed and handovers added.

When a new journey is analyzed, its handover points are compared to this reference range. If a handover occurs within the boundary of the most frequent handover point (i.e., within ± 100 meters of the identified point), the handover is considered valid and expected. However, if the handover occurs outside this predefined range, an alert or message is generated, indicating a potential anomaly or irregularity in the handover process. The message displayed in Figure 5.7 states that it happened 8 times earlier or later than expected.

```
Handover happened out of time 8 time(s).
View? (yes/no): yes
Alarm (handover):
      obuId  testplanId  testlogId  ...  date_time_diff  new_journey  new_direction
186       2          16         11  ... 0 days 00:02:00            1      ASCENDENT
189       2          16         11  ... 0 days 00:02:02            1      ASCENDENT
234       2          16         11  ... 0 days 00:02:30            1      ASCENDENT
237       2          16         11  ... 0 days 00:02:32            1      ASCENDENT
261       2          16         11  ... 0 days 00:02:47            1      ASCENDENT
264       2          16         11  ... 0 days 00:02:49            1      ASCENDENT
1179      2          16         11  ... 0 days 00:13:19            1      ASCENDENT
1182      2          16         11  ... 0 days 00:13:21            1      ASCENDENT

[8 rows x 39 columns]
```

Figure 5.7 Message exhibited when there are handover events happening out of time and showing the faulty instances.

The occurrence of events at handover points can be effectively analyzed using a histogram. A histogram provides a visual representation of the frequency or distribution of events at various handover stages, allowing for a clearer understanding of patterns or trends. By plotting these events along the timeline of handovers, you can observe how often certain events occur, identify any spikes or irregularities, and assess the overall performance or efficiency of the handover process.

Figure 5.8, which corresponds to the end of cell 7945, depicts a perfect handover zone, in which all events occurred in the same spot. This situation is ideal because it is expected

that handovers always happen in the same place, i.e., in the same PK. However, as we will see, this is not always the case, and variations in the handover zones are common, leading to deviations from this ideal scenario.

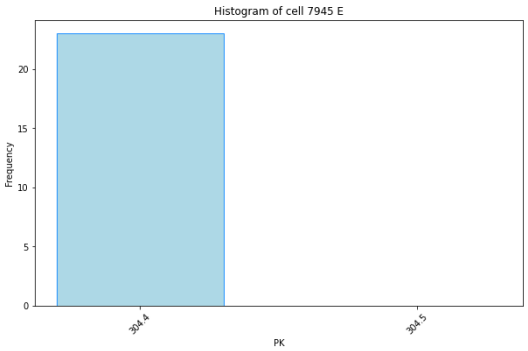


Figure 5.8 End of cell 7945.

Figure 5.9, which corresponds to the start of cell $7AD5$, shows the majority of occurences in PK 315.8. Nevertheless, there are six occurences in PK 315.7. All these are valid, since the interval comprises 100 meters to each side of the PK with higher frequency.

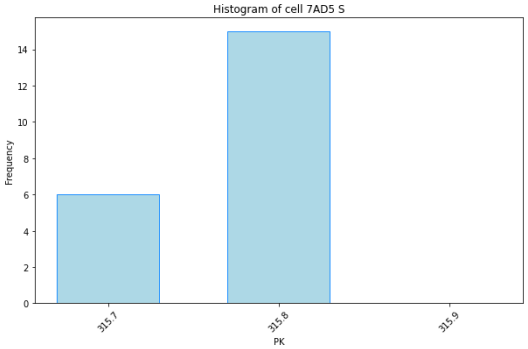


Figure 5.9 Start of cell 7AD5.

Figure 5.10, which corresponds to the start of cell 8241, depicts a handover zone in which all events occur within 300 meters.

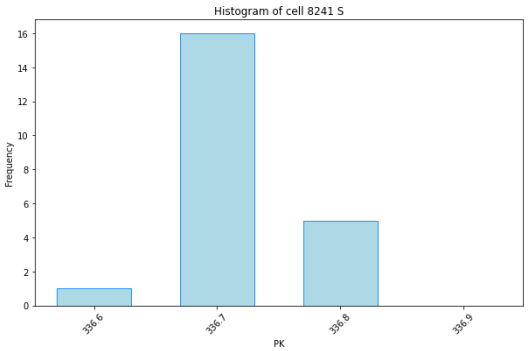


Figure 5.10 Start of cell 8241.

Figure 5.11 corresponds to the end of cell $7F21$. In contrast to the preceding images, this one depicts a handover zone where an anomalous event has occurred. Such occurrences are

retained for the purpose of providing an accurate and comprehensive account of the events in question. Nevertheless, the occurrence of new observations below 321.5 or above 321.7 would result in the generation of a warning message.

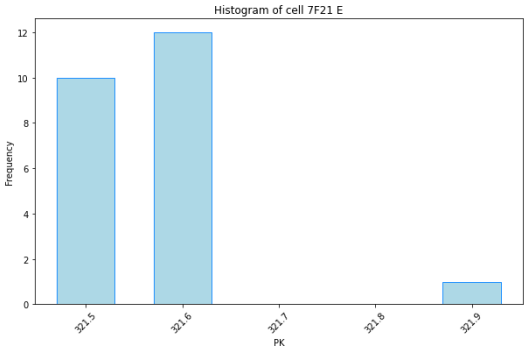


Figure 5.11 End of cell 7F21.

5.6 Conclusions

The decision to use classes for code organization is rooted in the principles of object-oriented programming, which promotes modularity, reusability, and maintainability of code. By encapsulating data and related functionalities within a class, we ensure a clear structure that enhances the readability of the program and simplifies debugging or future expansion. Furthermore, dictionaries were selected as the primary data structure due to their inherent efficiency in key-value pair storage and retrieval. The constant-time average complexity of lookups in dictionaries makes them an optimal choice for managing datasets where fast access to values is required based on unique identifiers. Additionally, dictionaries offer flexibility in data organization, allowing dynamic updates and storage of heterogeneous data types, which aligns well with the evolving nature of the program's requirements. Combining classes with dictionaries thus facilitates a structured, efficient, and scalable approach to data management and code organization.

By employing this methodology, we can systematically assess the validity of each journey and ensure consistency in signal and handover behavior, thereby improving the reliability of the analysis and the detection of anomalies. This approach ensures a methodical and data-driven method for evaluating signal and handovers, allowing for small variations while maintaining strict oversight on deviations that could signal an issue in the signal transfer. The use of frequency-based identification of handover points ensures that the analysis adapts to the most common behaviors, while the alert system quickly identifies any deviations.

While the current approach offers a practical framework, it is not without its limitations, and there remains significant room for improvement. A key shortcoming is the absence of established guidelines for selecting valid journeys and determining other critical parameters, such as the minimum number of occurrences to generate alarm. This lack of standardization introduces ambiguity, which may affect the consistency and reliability of the results. However,

despite these challenges, the approach has notable strengths, particularly in its intuitive nature. Its flexible structure allows for easy adaptation to various contexts, making it responsive to evolving requirements and specific scenarios.

Chapter 6

Final Considerations

The application of mathematics to industry plays a critical role in enhancing operational efficiency, innovation, and decision-making across a wide range of sectors. Mathematical techniques such as optimization, statistical analysis, and predictive modeling are employed to solve complex problems, improve processes, and reduce costs. In manufacturing, for instance, mathematical algorithms optimize production schedules and resource allocation, leading to higher productivity and minimized waste. Similarly, in finance, quantitative models are integral for risk assessment and investment strategies. Moreover, advancements in computational mathematics have enabled industries to harness the power of big data, driving more accurate forecasting and product development. Consequently, the integration of mathematical methods into industrial practices fosters technological advancements and contributes to the overall competitiveness of businesses in the global market.

Particularly in the context of trains and railway systems, mathematics has long been essential for optimizing operations, improving safety, and enhancing efficiency. In recent years, the integration of machine learning models has transformed this landscape, offering new avenues for solving complex, data-driven problems. With the increasing availability of large datasets and computational power, machine learning has emerged as a critical tool in predicting system behavior, analyzing maintenance needs, and ensuring the smooth operation of railway networks. This growing reliance on data science underscores the importance of applied mathematics as a foundation for building predictive and optimization models in industrial settings. In the railway sector, these models are crucial for advancing automation, predictive maintenance, and resource management, where small improvements can lead to substantial operational and economic gains.

The use of ensemble machine learning models, which combine multiple algorithms to improve classification accuracy, has proven highly effective in this work. By leveraging the strengths of various models and minimizing their individual weaknesses, the ensemble approach produced more robust and reliable results in the classification tasks undertaken. The success of ensemble methods in classification tasks in this context highlights their adaptability and effectiveness in solving real-world problems where the complexity of data requires more sophisticated analytical techniques.

In addition to classification tasks, machine learning models also offer considerable potential for regression analysis, particularly when combined with optimization techniques. Regression models are essential for tasks such as predicting PKs. When augmented by optimization methods, these models provided a powerful solution for finding the train's position.

One of the most straightforward yet critical components of this work was the identification of the direction and journey segmentation. Although seemingly simple in comparison to more complex machine learning tasks, correctly identifying the direction and individual journeys is foundational to the success of further analyses. The accurate segmentation of journeys ensures that all subsequent data processing, such as performance tracking or anomaly detection, is based on reliable and well-structured data. Without this initial clarity, even the most advanced models would be rendered ineffective due to erroneous or incomplete input data.

The algorithm developed for the generation of KPIs is another significant contribution of this work. By automating the analysis of KPIs, the algorithm provides a systematic way to assess journey performance. This not only streamlines decision-making processes but also allows for immediate identification of issues such as inefficiencies or maintenance needs. The ability to identify them dynamically across multiple files and journeys enables the IP to take proactive measures, enhancing the overall efficiency and reliability of railway services.

The importance of using the right tools in machine learning, mathematics, and software development cannot be overstated in a world that is becoming increasingly technological. The complexity of modern industrial systems, such as railways, demands the use of advanced analytical tools to manage and optimize operations effectively. Machine learning, combined with rigorous mathematical modeling and robust coding practices, provides the foundation for developing solutions that are not only efficient but also scalable and adaptable to future challenges. As the industry continues to evolve with advances in technology, the integration of these tools will be critical to maintaining competitiveness and driving innovation.

This code was designed for both segments of the Algarve line - the part from Lagos to Tunes and from Tunes to Vila Real de Santo António. With slight modifications, it has the potential to be replicated and adapted to the other lines of the Portuguese railway. This work demonstrates the transformative potential of applied mathematics and machine learning in the railway industry. By combining classification models, regression analysis, optimization techniques, and KPI generation, this approach offers a comprehensive framework for enhancing the performance and reliability of train operations. The success of this method illustrates the importance of embracing technological advancements and applying them strategically to solve complex industrial problems, setting a clear path for future innovation. As far as future work is concerned, I leave here two suggestions:

1. Experimenting with different models and quantity in the ensemble to see how this affects the classification result, or combine with methods such as clustering algorithms or KNN;

2. Identification of improvements and enhancement of the algorithm through the addition of new features.

Appendix A

Supporting Material for Chapter 2

The following material is provided as supplementary information for the second chapter of this report.

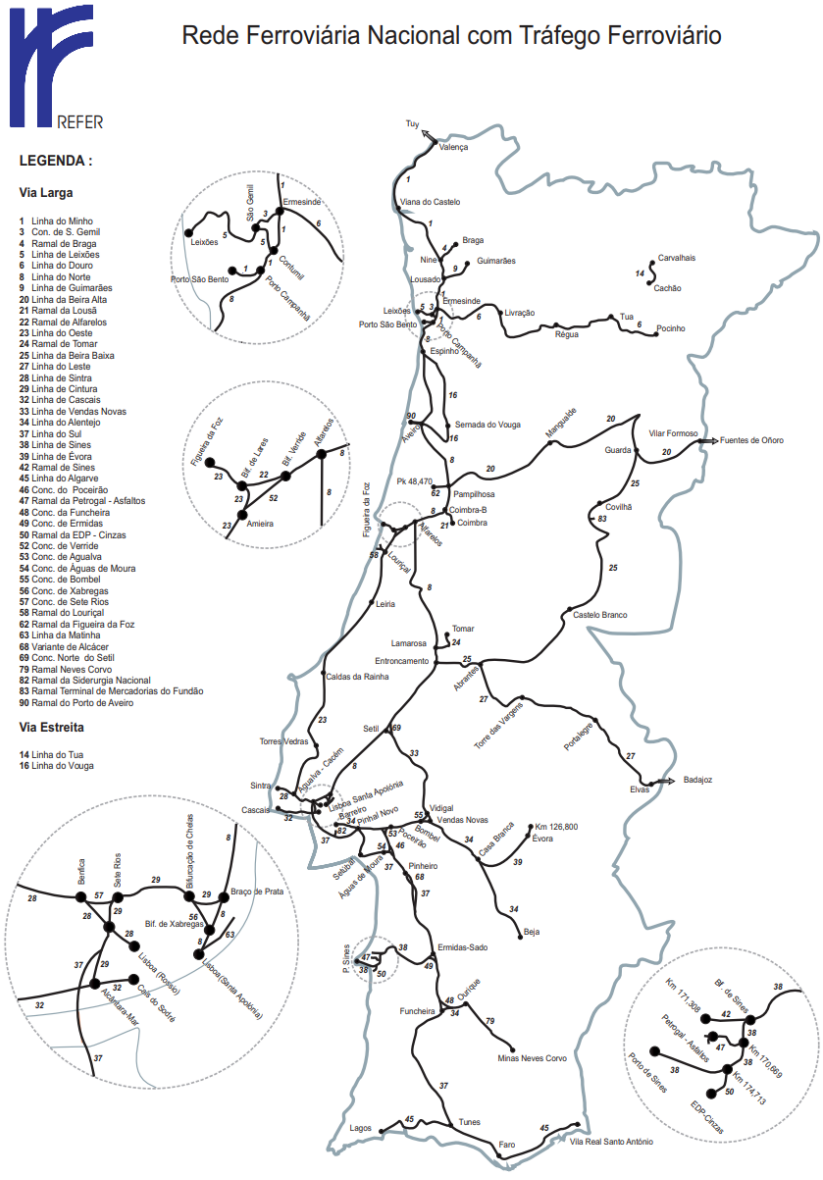


Figure A.1 Portuguese railway with active railway traffic, with details and caption.

Function to generate the RF classifiers

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV,
    StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import pickle

csv_files = [file for file in os.listdir('Lines/') if file.endswith('.
    csv')]
dfs = []

for file in csv_files:
    df = pd.read_csv('Lines/' + file, delimiter=',', decimal='.',
        usecols=('LAT','LON','ALT'))
    df['LINE'] = file.split(".")[0]
    dfs.append(df)

railway_data = pd.concat(dfs, ignore_index=True)

railway_data.head();
railway_data.tail();
railway_data.info();
df.isnull().sum()

num_classes = len(np.unique(railway_data['LINE']))
num_classes
railway_data['LINE'].value_counts()

plt.figure(figsize=(6, 8))
sns.countplot(railway_data['LINE'], color = "lightskyblue")
plt.xlabel("Count")
plt.title("Size of Each Line")
plt.show()

X = railway_data.loc[:, df.columns != 'LINE'].values
y = railway_data['LINE'].values
X[:5]
np.unique(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
    test_size=1/3, random_state=42)
print(X_train.shape)
```

```
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)

# Hyperparameter Tuning
n_estimators = list(range(10, 201, 10)) # 10, 20, 30 ... 200
max_depth = list(range(10, 51, 5)) # 10, 15, 20 ... 50
top_score = 0

for m in max_depth:
    rf = RandomForestClassifier(max_depth=m, max_samples=0.25,
       random_state=42).fit(X_train, y_train)
    current_score = rf.score(X_test, y_test)
    print("max_depth =", m)
    print(current_score)
    print()
    if current_score > top_score:
        top_score = current_score
        best_m = m

best_m

for n in n_estimators:
    rf = RandomForestClassifier(n_estimators=n, max_samples=0.25,
       random_state=42).fit(X_train, y_train)
    current_score = rf.score(X_test, y_test)
    print("n_estimators =", n)
    print(round(current_score, 8))
    print()
    if current_score > top_score:
        top_score = current_score
        best_n = n

best_n

# GridSearch
n_estimators = np.array([110,150,180]) # best 3 parameters
max_depth = np.array([30,35,45,50]) # 30 = 35; 45 = 50
top_score = 0

param = {'n_estimators':n_estimators, 'max_depth':max_depth}
k_fold = StratifiedKFold(n_splits=3, shuffle=True)
rf = RandomForestClassifier()

grid_search = GridSearchCV(rf, param_grid=param, cv=k_fold, n_jobs=5).
   fit(X,y)
```

```
85
86  grid_search.best_params_
87  grid_search.best_score_
88
89  # Manual Search
90  for n in n_estimators:
91      for m in max_depth:
92          rf = RandomForestClassifier(n_estimators=n, max_depth=m,
                max_samples=0.25, random_state=42).fit(X_train, y_train)
93          current_score = rf.score(X_test, y_test)
94          print(f'n_estimators = {n}')
95          print(f'max_depth = {m}')
96          print(round(current_score, 8))
97          print()
98          if current_score > top_score:
99              top_score = current_score
100             best_pair = (n, m)
101
102 best_pair
103
104 # RF Classifier
105 rf_110_45 = RandomForestClassifier(n_estimators=110, max_depth=45,
        random_state=42)
106 rf_110_45.fit(X_train, y_train)
107
108 model_pkl_file = "RF_110_45.pkl"
109
110 with open(model_pkl_file, 'wb') as file:
111     pickle.dump(rf_110_45, file)
112
113 pred = rf_110_45.predict(X_test)
114 accuracy_score(y_test, pred)
115
116 features = railway_data.columns
117 importances = rf_110_45.feature_importances_
118 indices = np.argsort(importances)
119
120 plt.figure(figsize=(6,2))
121 plt.title("Feature Importances")
122 plt.barh(range(len(indices)), importances[indices], color='aliceblue',
        edgecolor='dodgerblue', align='center')
123 plt.yticks(range(len(indices)), [features[i] for i in indices])
124 plt.xlabel("Relative Importance")
125 plt.show()
```

Appendix B

Supporting Material for Chapter 3

The following material is provided as supplementary information for the third chapter of this report.

Function for calculating PKs

```
import os
import time
import joblib
import numpy  as np
import pandas as pd

from itertools import chain
from tensorflow.keras.models import load_model

folder_file = "../Line/Points"
folder_railway = "../../Lines"

def load_file(file):
    with open(os.path.join(folder_file, file), "r") as file:
        content = pd.read_csv(file, delimiter=',', decimal='.')
    return content

def load_railway(file):
    with open(os.path.join(folder_railway, file), "r") as file:
        df = pd.read_csv(file, delimiter=',', decimal='.',
                         usecols=['LAT', 'LON', 'PK'])
    return df

def load_model_and_scalers(line):
    # Load model
    model = load_model(f"file_{line}.keras")
    # Load scalers
    scaler_X = joblib.load(f"scalerX_{line}.save")
```

```
    scaler_y = joblib.load(f"scalery_{line}.save")
    return model, scaler_X, scaler_y

def make_predictions(model, scaler_X, scaler_y, points):
    # Scale features
    coordinates = scaler_X.transform(points)
    # Predictions
    predictions_scaled = model.predict(coordinates)
    # Transform (inverse) the scaled predictions
    predictions = scaler_y.inverse_transform(predictions_scaled)
    flattened_list = list(chain(*predictions))
    return flattened_list

def binary_search(arr, low, high, x):
    if arr[high] <= x:
        return high
    if arr[low] > x:
        return low

    closest_index = low

    while low <= high:
        mid = (low + high) // 2
        if arr[mid] <= x and (mid == high or arr[mid + 1] > x):
            return mid
        elif arr[mid] < x:
            closest_index = mid
            low = mid + 1
        else:
            high = mid - 1
    return closest_index

def norm_squared(x1, x2, y1, y2):
    return (x1-x2)**2 + (y1-y2)**2

def golden_search(pi, pf, x, y, tol=1e-3):

    while abs(pf-pi) > tol:

        c = pf - (pf - pi) / phi
        d = pi + (pf - pi) / phi

        pos_c = binary_search(df['PK'], 0, len(df)-1, c)
        pos_d = binary_search(df['PK'], 0, len(df)-1, d)
        c_lat = df.loc[pos_c, 'LAT']
        c_lon = df.loc[pos_c, 'LON']
```

```
        d_lat = df.loc[pos_d, 'LAT']
        d_lon = df.loc[pos_d, 'LON']

        if norm_squared(x, c_lat, y, c_lon) < norm_squared(x, d_lat, y
            , d_lon):
            pf = d
        else:
            pi = c
    return (pf + pi) / 2


def calculate_pk(df, content, prediction, I):

    for pos in range(len(prediction)):
        x = content.loc[I[pos], 'latitude']
        y = content.loc[I[pos], 'longitude']

        if I[pos] != 0 and pos > 0:
            x_before = content.loc[I[pos]-1, 'latitude']
            y_before = content.loc[I[pos]-1, 'longitude']

            if x_before == x and y_before == y:
                pk = pks[-1]
                pks.append(pk)

        # Get index of prediction
        result = binary_search(df['PK'],
                               0,
                               len(df)-1,
                               prediction[pos])

        # Define range of search
        pi = max(df.loc[0, 'PK'], df.loc[result, 'PK']-0.4)
        pf = min(df.loc[result, 'PK']+0.4, df.loc[len(df)-1, 'PK'])
        pk = golden_search(pi, pf, x, y)
        pks.append(round(pk, 3))

    return pks
```

Code for Training the Neural Network (Jupyter Notebook)

```
import pandas            as pd
import numpy             as np
import matplotlib.pyplot as plt
import joblib

from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing   import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from mpl_toolkits            import mplot3d

df = pd.read_csv('Lines/452.csv', delimiter=',', decimal = '.',
    usecols=('LAT','LON','PK'))
df['PK'] = round(df['PK'],3)
df.tail()
df.info()

fig = plt.figure(figsize=plt.figaspect(0.5))
# =============
# First subplot
# =============
ax = fig.add_subplot(1, 2, 1, projection='3d')

# Data for three-dimensional scattered points
zdata = df['PK'].values
ydata = df['LAT'].values
xdata = df['LON'].values
ax.set_xlabel('LAT')
ax.set_ylabel('LON')
ax.set_zlabel('PK')
ax.plot3D(ydata, xdata, zdata, c='midnightblue')

# ==============
# Second subplot
# ==============
ax = fig.add_subplot(1, 2, 2, projection='3d')

# Data for a three-dimensional line
zline = df['PK'].values
yline = df['LAT'].values
xline = df['LON'].values
ax.plot3D(xline, yline, zline)

plt.show()
```

Appendix C

Supporting Material for Chapter 4

The following material is provided as supplementary information for the fourth chapter of this report.

Function to set direction

```
def set_direction(values, v1, v2):
    diff = values[v2] - values[v1]

    if diff > 0:
        return True     # Values are growing
    elif diff < 0:
        return False    # Values are shrinking

    i = v2
    while i < len(values)-1 and values[i] == values[i-1]:
        i += 1
    if i < len(values) and values[i] > values[i - 1]:
        return True
    else:
        return False
```

Function to convert *date_time* to object

```
def convert_date_time_to_object(df):
    date_time = []

    for i in range(len(df)):
        date_time_raw = df.loc[i]
        date_time_string = date_time_raw[:19]
        date_time_format = '%Y-%m-%d %H:%M:%S'
        date_time_obj = datetime.strptime(date_time_string,
            date_time_format)
        date_time.append(date_time_obj)
```

```
10
11      return date_time
```

Function to calculate time

```
1  def timestamp_difference(df):
2      times = (df - df.min())
3      return times
```

Function that separates journeys, labels them and adds the direction

```
1  def label_sequence(values, time_incrementation, step=pd.Timedelta(
       minutes=15)):
2
3      # Define initial condition
4      growing = set_direction(values, 0, 1)
5
6      labels = [1]
7      current_label = 1
8
9      if growing:
10         direct = ['ASCENDENT']
11         current_dir = 'ASCENDENT'
12
13     else:
14         direct = ['DESCENDENT']
15         current_dir = 'DESCENDENT'
16
17     # Labels and direction
18     for i in range(1, len(values)-1):
19
20         if growing:
21             if time_incrementation[i] - time_incrementation[i-1] >
                   step:
22                 current_label += 1
23                 labels.append(current_label)
24                 growing = set_direction(values, i, i+1)
25                 if growing:
26                     direct.append(current_dir)
27                 else:
28                     current_dir = 'DESCENDENT'
29                     direct.append(current_dir)
30             else:
31                 labels.append(current_label)
32                 direct.append(current_dir)
33                 if values[i+1] - values[i] < -0.1:
```

```
                    growing = False
                    current_dir = 'DESCENDENT'
                    current_label += 1

        elif not growing:
            if time_incrementation[i] - time_incrementation[i-1] >
               step:
                current_label += 1
                labels.append(current_label)
                growing = set_direction(values, i, i+1)
                if growing:
                    direct.append(current_dir)
                else:
                    current_dir = 'ASCENDENT'
                    direct.append(current_dir)
            else:
                labels.append(current_label)
                direct.append(current_dir)
                if values[i+1] - values[i] > 0.1:
                    growing = True
                    current_dir = 'ASCENDENT'
                    current_label += 1

        else:
            labels.append(current_label)
            direct.append(current_dir)

    labels.append(current_label)
    direct.append(current_dir)

    return labels, direct
```

Function to eliminate trips that are not valid

```
def eliminate_invalid_trips(journey, values, first_position=0, step=1)
   :

    for i in range(len(journey)-1):
        if journey[i] != journey[i+1] or i == len(journey)-2:
            dist = values[i] - values[first_position]
            if abs(dist) < step:
                for j in range(first_position, i+1):
                    journey[j] = 0
            first_position = i+1
    journey[-1] = journey[-2]

```

```
    return journey
```

Function that corrects the number of the journeys, in order to be sequential

```
def correct_journey(array):

    unique_values = np.unique(array)

    if unique_values[0] == 0:

        # Mapping where unique_values[i] will be replaced with i
        mapping = np.arange(len(unique_values))

        # Find indices in the unique array
        indices = np.searchsorted(unique_values, array)

        # Substitute values in original array with the corresponding
           indices
        new_array = mapping[indices]

    else:
        mapping = np.arange(1, len(unique_values)+1)
        indices = np.searchsorted(unique_values, array)
        new_array = mapping[indices]

    # print("Original array:", array)
    print("Unique values:", unique_values)
    print("Mapping:", mapping)
    print("New array:", new_array)

    return new_array
```

Appendix D

Supporting Material for Chapter 5

The following material is provided as supplementary information for the fifth chapter of this report.

Dictionary class

```
import os
import pickle
import numpy as np

class Dictionary:

    def __init__(self):

        # Define pk ranges
        self.pk_451 = np.round(np.arange(301.9, 347.2, 0.1), 1)
        self.pk_452 = np.round(np.arange(301.9, 396.5, 0.1), 1)

        # Dictionaries to keep info about signal level
        self.S_451A = dict.fromkeys(self.pk_451, None)
        self.S_451D = dict.fromkeys(self.pk_451, None)
        self.S_452A = dict.fromkeys(self.pk_452, None)
        self.S_452D = dict.fromkeys(self.pk_452, None)

        self.dict_for_signal = [self.S_451A,
                                self.S_451D,
                                self.S_452A,
                                self.S_452D]

        # Files for signal
        self.filename_signal = ['S451_ASCENDENT.pkl',
                                'S451_DESCENDENT.pkl',
                                'S452_ASCENDENT.pkl',
                                'S452_DESCENDENT.pkl']
```

```
29
30          # Dictionaries to keep info about handover points
31          self.H_451A = dict.fromkeys(self.pk_451, None)
32          self.H_451D = dict.fromkeys(self.pk_451, None)
33          self.H_452A = dict.fromkeys(self.pk_452, None)
34          self.H_452D = dict.fromkeys(self.pk_452, None)
35
36          self.dict_for_handover = [self.H_451A,
37                                    self.H_451D,
38                                    self.H_452A,
39                                    self.H_452D]
40
41          # Files for handover
42          self.filename_handover = ['H451_ASCENDENT.pkl',
43                                    'H451_DESCENDENT.pkl',
44                                    'H452_ASCENDENT.pkl',
45                                    'H452_DESCENDENT.pkl']
46
47      def load(self, filename):
48          if os.path.exists(filename):
49              with open(filename, 'rb') as f:
50                  return pickle.load(f)
51          return None
52
53      def save(self, dictionary, filename):
54          with open(filename, 'wb') as f:
55              pickle.dump(dictionary, f)
56
57      def create(self, letter):  # S: Signal; H: Handover
58
59          if letter == 'S':
60              for i, dictionary in enumerate(self.dict_for_signal):
61                  self.save(dictionary, self.filename_signal[i])
62                  print(f'Dictionary {self.filename_signal[i]} created.'
                        )
63
64          if letter == 'H':
65              for i, dictionary in enumerate(self.dict_for_handover):
66                  self.save(dictionary, self.filename_handover[i])
67                  print(f'Dictionary {self.filename_handover[i]} created
                        .')
68
69      def add(self, array, key, cell, place):
70
71          if array[key] is None:
72              array[key] = {}
```

```
            array[key][cell] = place
        elif cell in array[key]:
            array[key][cell] = np.append(array[key][cell], place)
        else:
            array[key][cell] = place
```

Signal class

```
import os
import numpy  as np
import pandas as pd
from statistics import stdev, median, mode
from dictionary import Dictionary

class Signal:

    def __init__(self):

        # Cells Line 451
        self.cells_451 = ['75C1', '7945', '7AD5', '7CC9', '7F21', '
            8241', '8499', '868D']

        # Cells Line 452
        self.cells_452 = ['75C1', '7819', '7A0D', '7B39', '7EBD', '7
            FE9', '8179', '84FD', '8755', '8AD9', '8B3D', '8D95', '9119
            ', '93D5', '962D', '9821']

    def include(self, dictionary, serving_cell, to_process):

        for i in to_process:
            pk = serving_cell.loc[i, 'pk_rounded']
            signal = serving_cell.loc[i, 'gsm_rx_level_bcch']

            if dictionary[pk] is None:
                dictionary[pk] = signal
            else:
                dictionary[pk] = np.append(dictionary[pk], signal)

    def populate(self):

        csv_files = [file for file in os.listdir(
            'Journey/Train') if file.endswith('.csv')]

        for file in csv_files:
            df = pd.read_csv('Journey/Train/' + file,
                             delimiter=',', decimal='.')
```

```
36
37              journeys = np.unique(df['new_journey']).tolist()
38              if 0 in journeys:
39                  journeys.remove(0)
40              print(f'Journeys in file {file}:', journeys)
41
42              for j in journeys:
43
44                  subset = df[df['new_journey'] == j]
45                  line = mode(subset['Line'])
46                  direct = mode(subset['new_direction'])
47                  # print(f'\nJourney {j} \nLine {line} \nDirection {
                        direct}')
48
49                  # Load dictionary
50                  filename = f'S{line}_{direct}.pkl'
51                  dictionary = Dictionary().load(filename)
52
53                  # Keep only 'SERVING_CELL'
54                  condition = subset['cell_type'] == 'SERVING_CELL'
55                  to_process = subset.index[condition]
56                  serving_cell = subset.drop(subset[subset['cell_type']
                        == 'NEIGHBOUR_CELL'].index)
57
58                  Signal().include(dictionary, serving_cell, to_process)
59                  print(f'Journey {j} added.')
60                  Dictionary().save(dictionary, filename)
61
62      def raise_alarm(self, dictionary, serving_cell, to_process,
            min_signal=-95, num_oc=20):
63          """
64          min_signal: minimum (absolute) level of signal
65          num_oc: define (minimum) number of ocorrences to set alarm
66
67          """
68
69          alarm_signal = []
70          alarm_error = []
71
72          for i in to_process:
73
74              signal = serving_cell.loc[i, 'gsm_rx_level_bcch']
75
76              if signal < min_signal:
77                  return False, alarm_signal, alarm_error
78
```

```
            pk = serving_cell.loc[i, 'pk_rounded']
            erro = serving_cell.loc[i, 'rxQual']

            if dictionary[pk] is not None:
                mediana = median(dictionary[pk])
                desvio = stdev(dictionary[pk])
                upper_bound = mediana + 4*desvio  # 99.9%
                lower_bound = mediana - 4*desvio

                if signal >= lower_bound and signal <= upper_bound:
                    # print("In bounds")
                    continue
                else:
                    alarm_signal.append(i)

                if erro > 5:
                    alarm_error.append(i)

        pk_problem = np.unique(alarm_signal)

        if len(pk_problem) <= num_oc:
            return False, alarm_signal, alarm_error
        else:
            return True, alarm_signal, alarm_error
```

Handover class

```
import os
import numpy  as np
import pandas as pd
import matplotlib.pyplot as plt
from statistics import mode
from dictionary import Dictionary

class Handover:

    def __init__(self):
        self.cells_451 = ['75C1', '7945', '7AD5', '7CC9', '7F21', '
            8241', '8499', '868D'] # Cells Line 451
        self.cells_452 = ['75C1', '7819', '7A0D', '7B39', '7EBD', '7
            FE9', '8179', '84FD', '8755', '8AD9', '8B3D', '8D95', '9119
            ', '93D5', '962D', '9821'] # Cells Line 452

    def timestamp_difference(self, df):
        times = (df - df.min())
        return times
```

```
17
18      def include(self, array, serving_cell, to_process,
            time_incrementation):
19          step = pd.Timedelta(seconds=3)
20
21          # Add handover info
22          for j in range(len(to_process) - 1):
23              i = to_process[j]
24              next_i = to_process[j + 1]
25
26              if serving_cell.loc[i, 'gsm_cell_id'] != serving_cell.loc[
                    next_i, 'gsm_cell_id']:
27                  if time_incrementation[next_i] - time_incrementation[i
                        ] < step:
28
29                      # Add info regarding current PK
30                      key = serving_cell.loc[i, 'pk_rounded']
31                      cell = serving_cell.loc[i, 'gsm_cell_id']
32                      place = 'E'  # End
33                      Dictionary().add(array, key, cell, place)
34
35                      # Next PK
36                      next_key = serving_cell.loc[next_i, 'pk_rounded']
37                      next_cell = serving_cell.loc[next_i, 'gsm_cell_id'
                            ]
38                      next_place = 'S'  # Start
39                      Dictionary().add(array, next_key, next_cell,
                            next_place)
40          return array
41
42      def raise_alarm(self, dic, serving_cell, to_process, line):
43          alarm = []
44          time_incrementation = serving_cell['date_time_diff']
45          step = pd.Timedelta(seconds=3)  # handover typically takes 2
                segs +/-
46          handover_df = pd.DataFrame()
47
48          if line == 451:
49              cells = self.cells_451
50          if line == 452:
51              cells = self.cells_452
52
53          # Iterate through each key-value pair in the dictionary
54          for key, value in dic.items():
55              for cell in cells:
56                  if value is None:
```

```
                    continue
                elif cell in value:
                    unique, counts = np.unique(dic[key][cell],
                                               return_counts=True)
                    # Add row
                    new_row = pd.DataFrame({'PK': [key],
                                            'Cell': cell,
                                            'Place': str(unique[0]),
                                            'Counts': counts[0]})

                    handover_df = pd.concat([handover_df, new_row],
                                            ignore_index=True)

                    # This df shows the number of ocorrences and
                    # the PK where they happened.

        # Check if handover was done in the right point
        for j in range(len(to_process) - 1):
            i = to_process[j]
            next_i = to_process[j + 1]

            # Detect change of cell
            if serving_cell.loc[i, 'gsm_cell_id'] != serving_cell.loc[
                next_i, 'gsm_cell_id']:

                # Check whether the observations are sequential
                if time_incrementation[next_i] - time_incrementation[i
                    ] < step:
                    key = serving_cell.loc[i, 'pk_rounded']
                    cell = serving_cell.loc[i, 'gsm_cell_id']
                    place = 'E'  # End
                    # print('\nCell', cell, '\nPlace', place)

                    # Define filters
                    cell_filter = handover_df['Cell'] == cell
                    place_filter = handover_df['Place'] == place

                    # Combine filters using the & (and) operator
                    combined_filter = place_filter & cell_filter

                    # Subset the DataFrame
                    df = handover_df[combined_filter]
                    del df['Cell']
                    del df['Place']

                    if not df.empty:
```

```
                    # Find index of max value in 'Counts' column
                    max_index = df['Counts'].idxmax()
                    # Get corresponding value in the 'PK' column
                    pk_handover_e = df.loc[max_index, 'PK']
                    Dictionary().add(dic, key, cell, place)

                    if key < pk_handover_e - 0.1:
                        alarm.append(i)
                        # print('Handover happened sooner.')
                    elif key > pk_handover_e + 0.1:
                        alarm.append(i)
                        # print('Handover happened later.')

                next_key = serving_cell.loc[next_i, 'pk_rounded']
                next_cell = serving_cell.loc[next_i, 'gsm_cell_id']
                next_place = 'S'  # Start
                # print('\nCell',next_cell,'\nPlace',next_place)

                # Define filters
                cell_filter = handover_df['Cell'] == next_cell
                place_filter = handover_df['Place'] == next_place

                # Combine filters using the & (and) operator
                combined_filter = place_filter & cell_filter

                # Subset the DataFrame
                df = handover_df[combined_filter]
                del df['Cell']
                del df['Place']

                if not df.empty:
                    max_index = df['Counts'].idxmax()
                    pk_handover_s = df.loc[max_index, 'PK']
                    Dictionary().add(dic, next_key, next_cell,
                        next_place)

                    if next_key < pk_handover_s - 0.1:
                        alarm.append(next_i)
                        # print('Handover happened sooner.')
                    elif next_key > pk_handover_s + 0.1:
                        alarm.append(next_i)
                        # print('Handover happened later.')
        if len(alarm) > 0:
            return True, alarm
        else:
```

```
145             return False, alarm
146
147     def populate(self):
148
149         csv_files = [file for file in os.listdir(
150             'Journey/Train') if file.endswith('.csv')]
151
152         for file in csv_files:
153             df = pd.read_csv('Journey/Train/' + file,
154                              delimiter=',',
155                              decimal='.')
156             journeys = np.unique(df['new_journey']).tolist()
157
158             if 0 in journeys:
159                 journeys.remove(0)
160
161             print(f'\nJourneys in file {file}:', journeys)
162
163             for j in journeys:
164                 subset = df[df['new_journey'] == j]
165                 line = mode(subset['Line'])
166                 direct = mode(subset['new_direction'])
167                 # print(f'\nJourney {j} \nLine {line} \nDirection {
                        direct}')
168
169                 if line == 451:
170                     cells = self.cells_451
171                 if line == 452:
172                     cells = self.cells_452
173
174                 # Eliminate invalid cells
175                 subset = subset[subset['gsm_cell_id'].isin(cells)]
176
177                 # Keep only 'SERVING_CELL'
178                 condition = subset['cell_type'] == 'SERVING_CELL'
179                 to_process = subset.index[condition]
180                 serving_cell = subset.drop(
181                     subset[subset['cell_type'] == 'NEIGHBOUR_CELL'].
                            index)
182
183                 # Load
184                 filename = f'H{line}_{direct}.pkl'
185                 array = Dictionary().load(filename)
186
187                 # Get time increment
188                 df['date_time'] = pd.to_datetime(df['date_time'])
```

```
                df['date_time_diff'] = Handover(
                ).timestamp_difference(df['date_time'])

                # Include handover if conditions are met
                array = Handover().include(array, serving_cell,
                                           to_process, df['
                                              date_time_diff'])
                print(f'Journey {j} processed.')
                Dictionary().save(array, filename)

    def plot(self, df, cell, place):

        # Define filters
        cell_filter = df['Cell'] == cell
        place_filter = df['Place'] == place

        # Combine filters using the & (and) operator
        combined_filter = place_filter & cell_filter

        # Subset the DataFrame
        df = df[combined_filter]
        del df['Cell']
        del df['Place']

        if len(df['PK']) > 0:

            data = {
                'PK': df['PK'],
                'Counts': df['Counts']
            }
            df = pd.DataFrame(data)

            # Define the range of PK values to ensure all values are
               included
            min_val = df['PK'].min()
            max_val = df['PK'].max()
            step = 0.1  # Define the step size

            all_values = np.arange(min_val, max_val + step, step)

            # Create dictionary to hold counts for all possible values
            counts_dict = {val: 0 for val in all_values}
            counts_dict.update(df.set_index('PK')['Counts'].to_dict())

            # Extract updated PK and Counts
            updated_pk = list(counts_dict.keys())
```

```
            updated_counts = list(counts_dict.values())

            # Set up the plot
            plt.figure(figsize=(10, 6))

            # Plot the bar chart
            plt.bar(updated_pk,
                    updated_counts,
                    color='lightblue',
                    edgecolor='dodgerblue',
                    width=0.06,
                    label=cell)

            # Set the x-ticks (adjust tick frequency and rotation as
               needed)
            plt.xticks(np.arange(min_val, max_val + step, step),
               rotation=45)

            # Labels and title
            plt.xlabel('PK')
            plt.ylabel('Frequency')
            plt.title(f'Histogram of cell {cell} {place}')

            # Show the plot
            plt.show()

    def visualize(self, line, filename):

        array = Dictionary().load(filename)
        handover_df = pd.DataFrame()

        if line == 451:
            cells = self.cells_451
        if line == 452:
            cells = self.cells_452

        # Iterate through each key-value pair in the dictionary
        for key, value in array.items():
            for cell in cells:
                if value is None:
                    continue
                elif cell in value:
                    unique, counts = np.unique(array[key][cell],
                                               return_counts=True)
                    # Add row
                    new_row = pd.DataFrame({'PK': [key],
```

```
                                            'Cell': cell,
                                            'Place': str(unique[0]),
                                            'Counts': counts[0]})
                    handover_df = pd.concat([handover_df, new_row],
                                            ignore_index=True)
        for cell in cells:
            for place in ['S', 'E']:
                Handover().plot(handover_df, cell, place)
```

Main

```
import os
import numpy  as np
import pandas as pd
from statistics import mode
from dictionary import Dictionary
from sig        import Signal
from handover   import Handover

def test_new_journeys():
    csv_files = [file for file in os.listdir('Journey/New')
                 if file.endswith('.csv')]

    for file in csv_files:
        df = pd.read_csv('Journey/New/' + file, delimiter=',', decimal
           ='.')
        journeys = np.unique(df['new_journey']).tolist()
        if 0 in journeys:
            journeys.remove(0)
        print(f'\nJourneys in file {file}:', journeys[-1])

        for j in journeys:
            subset = df[df['new_journey'] == j]
            line = mode(subset['Line'])
            direct = mode(subset['new_direction'])
            print(f'\nJourney {j} \nLine {line} \nDirection {direct}')

            # Keep only 'SERVING_CELL'
            condition = subset['cell_type'] == 'SERVING_CELL'
            to_process = subset.index[condition]
            serving_cell = subset.drop(
                subset[subset['cell_type'] == 'NEIGHBOUR_CELL'].index)

            # Signal

            filename = f'S{line}_{direct}.pkl'
```

```
            dictionary = Dictionary().load(filename)

            raise_alarm, alarm_signal, alarm_error = Signal(
            ).raise_alarm(dictionary, serving_cell, to_process)

            if raise_alarm:
                print(f'Journey {j} not added.')
                user_input = input('View? (yes/no): ').strip().lower()
                if user_input == 'yes':
                    print('Alarm (signal):')
                    print(serving_cell.loc[alarm_signal])
                    print('Alarm (error):')
                    print(serving_cell.loc[alarm_error])
                else:
                    print('Not displaying.')
            else:
                Signal().include(dictionary, serving_cell, to_process)
                print(f'Journey {j} added.')

            Dictionary().save(dictionary, filename)

            # Handover
            filename = f'H{line}_{direct}.pkl'
            dictionary = Dictionary().load(filename)

            serving_cell['date_time'] = pd.to_datetime(
                serving_cell['date_time'])
            serving_cell['date_time_diff'] = Handover(
            ).timestamp_difference(serving_cell['date_time'])

            raise_alarm_h, alarm = Handover().raise_alarm(dictionary,
               serving_cell, to_process, line)

            if raise_alarm_h:
                print(f'Handover happened out of time {len(alarm)}
                   time(s).')
                user_input = input('View? (yes/no): ').strip().lower()
                if user_input == 'yes':
                    print('Alarm (handover):')
                    print(serving_cell.loc[alarm])
                else:
                    print('Not displaying.')
            else:
                print('Handover happened smoothly.')
            Dictionary().save(dictionary, filename)
            # If needed, add condition to delete files from the folder
```

```
79              # after being processed
80          print(f'\nFinished processing file {file}.')
81
82  def main(choice):
83      if choice == '1':
84          # Signal
85          Dictionary().create('S')
86          Signal().populate()
87          # Handover
88          Dictionary().create('H')
89          Handover().populate()
90
91      if choice == '2':
92          test_new_journeys()
93
94  # '1' - Restart both dictionaries - Signal and Handover - and populate
        with the journeys contained inside the folder 'Train'
95  # '2' - Test new journeys and add them to the dictionary, if they are
        valid
96  main('2')
```

Bibliography

[1] Abid Ali Awan. What is bagging in machine learning? a guide with examples. https://www.datacamp.com/tutorial/what-bagging-in-machine-learning-a-guide-with-examples?dc_referrer=https%3A%2F%2Fwww.google.com%2F#:~:text=Bagging%20(bootstrap%20aggregating)%20is%20an,predictions%20through%20voting%20or%20averaging, 2023. Accessed: 6 September 2024.

[2] Gérard Biau, Luc Devroye, and Gábor Lugosi. Consistency of random forests and other averaging classifiers. *Journal of Machine Learning Research*, 9:2015–2033, 2008.

[3] Michel Bierlaire. *Optimization: Principles and Algorithms*. EPFL Press, Lausanne, 2nd edition, 2018.

[4] Leo Breiman. Bagging predictors. *1996 Kluwer Academic Publishers, Boston*, Machine Learning(24):123–140, 1996.

[5] Leo Breiman. Random forests. *2001 Kluwer Academic Publishers, Netherlands*, Machine Learning(45):5–32, 2001.

[6] Archana Chaudhary, Savita Kolhe, and Raj Kamal. An improved random forest classifier for multi-class classification. *Information Processing in Agriculture*, 3:215–222, 2016.

[7] IP Infraestruturas de Portugal. Linhas ip. `LinhasIP.xlsx`, 2024. Excel spreadsheet.

[8] Scikit Learn Developers. 1.10.7. mathematical formulation. https://scikit-learn.org/stable/modules/tree.html#tree-mathematical-formulation, 2024. Accessed: September 12, 2024.

[9] INTF Instituto Nacional do Transporte Ferroviário. Instrução de exploração técnica nº 50. Technical report, I.E.T. 50, 2005.

[10] EIRENE. Eirene system requirements specification, version 16.0.0. Technical report, GSM-R Operators Group, 21 December 2015. Available as PDF.

[11] Chollet F. *Deep Learning with Python.* Manning Publications Co., Shelter Island NY, 2021.

[12] Sandhya Krishnan. How do determine the number of layers and neurons in the hidden layer? https://medium.com/geekculture/introduction-to-neural-network-2f8b8221fbd3, 2023. Accessed: September 24, 2023.

[13] Mahdi. Dictionary as data structure: What, when, how? https://mahdisaeedi.medium.com/dictionary-as-data-structure-what-when-how-54af30b6ed21, 2024. Accessed: September 20, 2023.

[14] Sujatha Mudadla. Deep neural networks vs dense neural networks. https://medium.com/@sujathamudadla1213/deep-neural-networks-vs-dense-neural-networks-bad5918a5b2a, 2023 December 3. Accessed: September 14, 2023.

[15] Kizito Nyuytiymbiy. Parameters and hyperparameters in machine learning and deep learning. https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac, 2020 December 30. Accessed: November 15, 2024.

[16] Angshuman Paul, Dipti Prasad Mukherjee, Prasun Das, Abhinandan Gangopadhyay, Appa Rao Chintha, and Saurabh Kundu. Improved random forest for classification. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 27(8):4012–4024, August 2018.

[17] Frank Pfenning. Binary search. Lecture 6 Notes, Spring 2016. 15-122: Principles of Imperative Computation.

[18] Natassha Selvaraj. Python pickle tutorial: Object serialization. https://www.datacamp.com/tutorial/pickle-python-tutorial?utm_source=google&utm_medium=paid_search&utm_campaignid=21374847033&utm_adgroupid=165153430282&utm_device=c&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=716161020391&utm_targetid=aud-2274077226600:dsa-2218886984100&utm_loc_interest_ms=&utm_loc_physical_ms=20873&utm_content=&utm_campaign=240617_1-sea~dsa~tofu_2-b2c_3-ptbr-lang-en_4-prc_5-na_6-na_7-le_8-pdsh-go_9-nb-e_10-na_11-na-oct24&gad_source=1&gbraid=0AAAADQ9WsFyQRpQRDQBqCKj4M3pfMZfx&gclid=Cj0KCQjwveK4BhD4ARIsAKy6pMLfWht7PAJZyleJeXLNevDfing7wYWa0xPECCFPP_2iXOzpKWZXtKoaAvxLEALw_wcB, 2024. Accessed: September 27, 2024.

[19] Thomas H Cormen. Charles E Leiserson. Ronald L Rivest. Clifford Stein. *Introduction To Algorithms (Third Edition)*. The MIT Press. Cambridge, Massachusetts. London, England., 2009.

[20] CFI Team. Bagging (bootstrap aggregation). https://corporatefinanceinstitute.com/resources/data-science/bagging-bootstrap-aggregation/. Accessed: 7 September 2024.