INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Victor Hugo de Oliveira Bastos

PREDICTION OF AIR TRAFFIC IN CONTROL SECTORS USING MACHINE LEARNING FOR ATFM ANALYSIS

Final Paper
2024

Course of Aerospace Engineering

CDU 681.3:16

Victor Hugo de Oliveira Bastos

PREDICTION OF AIR TRAFFIC IN CONTROL SECTORS USING MACHINE LEARNING FOR ATFM ANALYSIS

Advisor

Prof. Dr. José Maria Parente de Oliveira (ITA)

AEROSPACE ENGINEERING

São José dos Campos
Instituto Tecnológico de Aeronáutica

2024

Cataloging-in Publication Data
Documentation and Information Division

de Oliveira Bastos, Victor Hugo
Prediction of Air Traffic in Control Sectors using Machine Learning for ATFM Analysis / Victor Hugo de Oliveira Bastos.
São José dos Campos, 2024.
140f.

Final paper (Undergraduation study) – Course of Aerospace Engineering– Instituto Tecnológico de Aeronáutica, 2024. Advisor: Prof. Dr. José Maria Parente de Oliveira.

1. Cupim. 2. Dilema. 3. Construção. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

DE OLIVEIRA BASTOS, Victor Hugo. **Prediction of Air Traffic in Control Sectors using Machine Learning for ATFM Analysis**. 2024. 140f. Final paper (Undergraduation study) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

Victor Hugo de Oliveira Bastos
Rua H8B, Ap. 231
12.228-461 – São José dos Campos–SP

PREDICTION OF AIR TRAFFIC IN CONTROL SECTORS USING MACHINE LEARNING FOR ATFM ANALYSIS

This publication was accepted like Final Work of Undergraduation Study

---

Victor Hugo de Oliveira Bastos

Author

José Maria Parente de Oliveira (ITA)

Advisor

Prof. Dr. Cristiane Aparecida Martins
Course Coordinator of Aerospace Engineering

São José dos Campos: NOVEMBER 23, 2024.

To all the ”merely players” who, in their exits and entrances, played their parts, rendering this stage unforgettable.

Acknowledgments

First and foremost, I would like to express my gratitude to God for His infinite mercy and guidance that transcends our understanding, ”straightening my paths.”

To my family. My father Arilson, my mother Giovana, and my brother Vinicius, for being part of every step until today and truly making my dreams theirs.

To my beloved grandparents who defied destiny and demonstrated to Albert Einstein their values by the measure and sense in which they attained liberation from the self, freeing my parents.

To other relatives who were my anchor in times of storm, and whom I hope to offer the same refuge, thus completing the cycle of support described by Khalil Gibran.

To my college friends who turned moments of ’carefree laughter’ into what Guimarães Rosa pointed out as happiness.

To other friends who proved to be true, as defined by Saint Thomas Aquinas through Idem Velle, Idem Nolle.

To some teachers who handed me the weapon that Nelson Mandela considered the most powerful for changing the world.

Finally, I express my sincere appreciation to all who, throughout life’s encounters and partings, enriched my journey with what no misfortune can deprive, in the name of my grandfather José Hugo, who always endeavored to impart all the knowledge that was incumbent upon him, demonstrating beyond words that ”the impossible is done immediately. Miracles take a little longer.” as he quoted David Berglas.

"There are more things in heaven and earth, Horatio,
Than are dreamt of in your philosophy."
— William Shakespeare, *Hamlet*

Resumo

A previsão eficaz do tráfego aéreo dentro das Regiões de Informação de Voo (FIR) é crucial para gerenciar e otimizar as operações de tráfego aéreo. Esta pesquisa utiliza técnicas computacionais e de aprendizado de máquina, juntamente com recursos em nuvem, para desenvolver um pipeline sofisticado destinado a prever o número de aeronaves sobrevoando essas seções de FIR, para uso na Análise ATFM do ICEA. Ao integrar múltiplos conjuntos de dados, incluindo informações de voos, dados meteorológicos e observações de radar, e refiná-los para performar bem em modelos de aprendizado de máquina, o estudo visa aprimorar a precisão e a confiabilidade das previsões de tráfego aéreo. O algoritmo final desenvolvido prevê as posições das aeronaves com um erro médio de menos de 18 km uma hora à frente, o que leva à confiabilidade das contagens regionais de aeronaves. Isso é evidenciado pela comparação com um modelo de referência baseado na velocidade média das aeronaves, que apresenta um erro médio 2,9 vezes maior nas contagens regionais de aeronaves. Potenciais melhorias e limitações foram identificados e discutidos para que a futura implementação possa ser otimizada pela equipe do ICEA.

Abstract

The effective prediction of air traffic within Flight Information Regions (FIR) is crucial for managing and optimizing air traffic operations. This research leverages computational and machine learning techniques, along with cloud-based resources, to develop a sophisticated pipeline for predicting the number of aircraft flying over these FIR sections, intended for use in ICEA's ATFM Analysis. By integrating multiple datasets, including flight information, weather data, and radar observations, and refining them to perform well in machine learning models, the study aims to enhance the accuracy and reliability of air traffic predictions. The final algorithm developed predicts aircraft positions with an average error of less than 18 km one hour ahead, which leads to the reliability of the regional aircraft counts. This is evidenced by the comparison with a baseline model based on the average speed of the aircraft, which has an average error 2.9 times greater in regional aircraft counts. Potential improvements and limitations have been identified and discussed so that future implementation can be optimized by the ICEA team.

List of Figures

FIGURE 2.1 – Map of FIR sectors in Brazil. (Departamento de Controle do Espaço Aéreo (DECEA), 2024) 25

FIGURE 2.2 – Diagram illustrating the concept of RESTful API. (Contentful, 2024) 27

FIGURE 2.3 – Processed satellite image showing detected storm areas. (ICEA, 2023) 31

FIGURE 2.4 – Number of flights per hour of the day. 33

FIGURE 2.5 – Using ThreadPool to speed up API requests. Source: (CARTER, 2020) 34

FIGURE 2.6 – Illustration of the Feature Engineering process (FLAXMAN, 2022). . 36

FIGURE 2.7 – Illustration of SMOTE oversampling technique (ORELLANA, 2020). . 38

FIGURE 2.8 – Example of Spatial Data Representation (SIGLES Programme, 2018) . 39

FIGURE 2.9 – Structure of an MLP with its layers (ENDEAVOURS, 2019). 42

FIGURE 2.10 –Illustration of a Random Forest structure (MASTER, 2019). 44

FIGURE 2.11 –Structure of XGBoost with its decision trees. (GEEKSFORGEEKS, 2023) 46

FIGURE 2.12 –Illustration of a Voting Classifier combining multiple models. (RASCHKA, 2019). 49

FIGURE 2.13 –Structure of a neural network with input, hidden, and output layers (SPICEWORKS, 2021). 56

FIGURE 2.14 –Illustration of the Cross-Validation process (WIJAYA, 2023). 60

FIGURE 3.1 – Flights by Origin and Destination Airport 68

FIGURE 3.2 – Histogram of Flight Durations 68

FIGURE 3.3 – Relation Between Average Distance and Maximum Flight Altitude . 69

FIGURE 3.4 – Average Waiting Time by Hour of the Day 69

FIGURE 3.5 – Original and Thresholded Images 70

FIGURE 3.6 – Detected Polygons in Satellite Image 72
FIGURE 3.7 – Confusion Matrix for the MLP Model 86
FIGURE 3.8 – Structure of the MLP Classifier used 88
FIGURE 3.9 – Predicted vs Actual Coordinates for Latitude and Longitude 91
FIGURE 3.10 –Predicted Aircraft Locations within FIR Sectors 96
FIGURE 3.11 –Mean Absolute Error (MAE) of Aircraft Count Predictions in FIR Sectors 98
FIGURE 3.12 –Production Workflow for Predicting Aircraft Positions 98

FIGURE 4.1 – Confusion Matrix for MLP Classifier 100
FIGURE 4.2 – Confusion Matrix for Random Forest Classifier 101
FIGURE 4.3 – Confusion Matrix for XGBoost Classifier 102
FIGURE 4.4 – Confusion Matrix for Logistic Regression 103
FIGURE 4.5 – Confusion Matrix for Voting Classifier 103
FIGURE 4.6 – Mean RMSE over the last 10 days for XGBoost model. 105
FIGURE 4.7 – Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h with RMSE lat 0.23 and RMSE lon 0.20 for XGBoost model. 106
FIGURE 4.8 – Flight trajectories depicting actual (full lines) and predicted (dotted lines) paths. 106
FIGURE 4.9 – Mean RMSE over the last 10 days for Lasso Regression model. . . . 107
FIGURE 4.10 –Real latitudes versus predicted values for date reference 2023-05-01 15:00h for Lasso Regression model with RMSE lat 2.06. 107
FIGURE 4.11 –Mean RMSE over the last 10 days for Gradient Boost model. 108
FIGURE 4.12 –Real latitudes versus predicted values for date reference 2023-05-01 15:00h for Gradient Boost model with RMSE lat 0.94. 108
FIGURE 4.13 –Mean RMSE over the last 10 days for Elastic Net model. 109
FIGURE 4.14 –Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Elastic Net model with RMSE lat 2.04. . 109
FIGURE 4.15 –Mean RMSE over the last 10 days for Neural Network 1 model. . . . 110

FIGURE 4.16 –Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Neural Network 1 model with RMSE lat 0.43 and RMSE lon 1.68. 110

FIGURE 4.17 –Mean RMSE over the last 10 days for Neural Network 2 model. . . . 111

FIGURE 4.18 –Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Neural Network 2 model with RMSE lat 0.49 and RMSE lon 1.72. 111

FIGURE 4.19 –Mean RMSE over the last 10 days for Neural Network 3 model. . . . 112

FIGURE 4.20 –Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Neural Network 3 model with RMSE lat 0.47 and RMSE lon 6.93. 112

FIGURE 4.21 –Mean RMSE over the last 10 days for Neural Network 4 model. . . . 113

FIGURE 4.22 –Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Neural Network 4 model with RMSE lat 0.34 and RMSE lon 0.48. 114

FIGURE 4.23 –Estimate of Longitudinal span of Distrito Federal. Adapted from Google Maps 116

FIGURE 4.24 –FIR sector map with blue points representing predicted positions and red points representing actual positions of aircraft. 117

FIGURE 4.25 –Comparison of FIR sector predictions with baseline model (green points), predicted positions (blue points), and actual positions (red points). 118

FIGURE 4.26 –MAE over time for the validation period. 119

FIGURE 4.27 –MAE over time with the number of planes. 119

FIGURE 4.28 –Pearson correlation coefficient over time. 119

FIGURE 4.29 –Pearson correlation over time with the number of planes. 120

FIGURE 4.30 –Average distance per hour in a scatter plot, showing an average of 17.73 km of distance. 120

FIGURE 4.31 –Illustration of varied flight trajectories highlighting the non-linear paths observed during exploratory data analysis. 121

FIGURE A.1 –Wait times by hour of the day 139

FIGURE A.2 –Mean distance for highest flight traffic times 140

FIGURE A.3 –Mean duration of flights by hour of the day 140
FIGURE A.4 –Number of flights by day of the week 140
FIGURE A.5 –Number of flights by hour of the day 140

List of Tables

TABLE 3.1 – Summary of Saved Data Files 66
TABLE 3.2 – Coordinates of Major Brazilian Airports. Source: (AIRPORT...,) . . 68
TABLE 3.3 – Comparison of Radar Data, Predicted Data, and Baseline Model Data for FIR Sectors 97

TABLE 4.1 – Classification Report for MLP Classifier 100
TABLE 4.2 – Classification Report for Random Forest Classifier 101
TABLE 4.3 – Classification Report for XGBoost Classifier 102
TABLE 4.4 – Classification Report for Logistic Regression 102
TABLE 4.5 – Classification Report for Voting Classifier 104
TABLE 4.6 – Performance Comparison of Classification Models 104
TABLE 4.7 – Performance Comparison of Regression Models 114
TABLE 4.8 – Evaluation Metrics for the Selected Timestamp 116
TABLE 4.9 – Predictions for FIR sectors on 2023-05-12 11:00h 117
TABLE 4.10 – Summary of Performance Metrics Over the Validation Period 121
TABLE 4.11 – Performance Metrics for Exceptional Test Cases 123

List of Abbreviations and Acronyms

API	Application Programming Interface
ATFM	Air Traffic Flow Management
AWS	Amazon Web Services
CAT-62	Radar Data Category 62
CSV	Comma-Separated Values
EDA	Exploratory Data Analysis
FIR	Flight Information Region
GPU	Graphics Processing Unit
ICEA	Instituto de Controle do Espaço Aéreo
Lasso	Least Absolute Shrinkage and Selection Operator
MAE	Mean Absolute Error
MLP	Multi-Layer Perceptron
NN	Neural Network
PIL	Python Imaging Library
PILLOW	Python Imaging Library Fork
RMS	Root Mean Square
RMSE	Root Mean Square Error
SMOTE	Synthetic Minority Over-sampling Technique
SQL	Structured Query Language
TPU	Tensor Processing Unit
UR	Unified Robot
XGBoost	Extreme Gradient Boosting

List of Symbols

a	Average Distance
$\mathbf{a}$	Vector of Distances
$\boldsymbol{D}$land	Vector of flights landed before the prediction time
$\boldsymbol{D}$takeoff	Vector of flights taking off before the prediction time
$\boldsymbol{D}$dur	Vector of flight durations for each date
dij	Distance between points i and j
$\hat{y}$	Predicted Value
y	Actual Value
n_{landed}	Number of flights that landed
n_{departed}	Number of flights that departed
n	Number of Samples
δ_{k-k_f}	Kronecker delta at time k_f
Δt	Time interval
$\mathbf{e}_j$	Unit vector of dimension n with the j-th component equal to 1
$\mathbf{K}$	Stiffness Matrix
$\mathbf{X}$	Feature Matrix
$\mathbf{y}$	Target Vector
λ	Learning Rate
μ	Mean
σ	Standard Deviation

Contents

1 Introduction 20

1.1 Motivation 20

1.2 Problem Definition 21

1.2.1 Dynamic Nature of FIRs 21

1.2.2 Challenges in Air Traffic Prediction 21

1.3 Objectives 22

1.3.1 Development of a Robust Predictive Model 22

1.3.2 Model Testing, Evaluation, and Replicability 23

2 Theoretical Foundation 24

2.1 Control Sector 24

2.1.1 Definition and Purpose 24

2.1.2 Structure and Management 24

2.1.3 Dynamic Nature of FIRs 25

2.2 API 26

2.2.1 Introduction to APIs 26

2.2.2 ICEA API 26

2.3 Image Processing 29

2.3.1 Introduction to Image Processing 29

2.3.2 Processing Meteorological Satellite Images 30

2.3.3 Identifying Flights Passing Through Storm Areas 32

2.4 Exploratory Data Analysis 33

2.5 Data Preparation 33

2.5.1 Data Fetching . . . 34
2.5.2 Efficient Data Fetching with ThreadPool . . . 34
2.5.3 Data Integration . . . 35
2.5.4 Feature Engineering . . . 36
2.5.5 Geographic Data Processing . . . 38
2.6 Machine Learning . . . 40
2.6.1 Classification Models . . . 41
2.6.2 Regression Models . . . 50
2.6.3 Neural Networks . . . 55
2.6.4 Hyperparameter Tuning . . . 57
2.6.5 Cross-Validation . . . 59
2.6.6 Evaluating Models . . . 60

3 Methodology . . . 64
3.1 Environment . . . 64
3.2 Data Fetching . . . 65
3.3 Exploratory Data Analysis . . . 67
3.3.1 Data Loading and Preprocessing . . . 67
3.3.2 Airports Coordinates Enrichment . . . 67
3.3.3 Data Visualization . . . 67
3.4 Weather Image Processing . . . 69
3.4.1 Image Download and Preprocessing . . . 69
3.4.2 Image Decoding and Polygon Construction . . . 70
3.4.3 Efficient Data Processing with Apache Spark . . . 73
3.4.4 Polygon Intersection and SQL Queries . . . 74
3.4.5 Output and Integration . . . 75
3.5 Chosen Approach for the Problem . . . 75
3.6 Feature Engineering . . . 76
3.6.1 Flight Data Processing . . . 77
3.6.2 Airport Waiting Data Processing . . . 78

3.6.3 METAR Data Processing . . . 79
3.6.4 Runway Change Data Processing . . . 81
3.6.5 Combining Features . . . 82
3.6.6 Last Processing Steps . . . 83
3.7 Classification Models . . . 85
3.7.1 Training Models . . . 85
3.7.2 Models Trained . . . 87
3.7.3 Final Classification Model Selection . . . 88
3.8 Regression Models . . . 90
3.8.1 Training Models . . . 90
3.8.2 Models Trained . . . 91
3.8.3 Final Regression Model Selection . . . 93
3.8.4 Final Model . . . 94
3.9 Counting Aircraft in FIR Sectors . . . 94
3.9.1 Identification of Contained Regions . . . 94
3.9.2 Counting Aircraft . . . 95
3.10 Evaluating the Final Predictions . . . 96
3.10.1 Obtaining True Values . . . 96
3.10.2 Baseline Model . . . 96
3.10.3 Evaluating the Final Predictions . . . 97
3.11 Production Workflow . . . 98
4 Results . . . 100
4.1 Classification Models . . . 100
4.1.1 MLP Classifier . . . 100
4.1.2 Random Forest Classifier . . . 101
4.1.3 XGBoost Classifier . . . 101
4.1.4 Logistic Regression . . . 102
4.1.5 Voting Classifier . . . 103
4.1.6 Discussion . . . 104

4.2 Regression Models . . . 105
4.2.1 XGBoost . . . 105
4.2.2 Lasso Regression . . . 107
4.2.3 Gradient Boost . . . 108
4.2.4 Elastic Net . . . 109
4.2.5 Neural Network 1 . . . 110
4.2.6 Neural Network 2 . . . 111
4.2.7 Neural Network 3 . . . 112
4.2.8 Neural Network 4 . . . 112
4.3 Discussion . . . 114
4.4 Sector Grouping . . . 116
4.4.1 Specific Case Illustration . . . 116
4.4.2 General Results in Validation Window . . . 118
4.4.3 Evaluation in Specific Conditions . . . 122
4.4.4 Limitations and Next Steps . . . 124

5 Conclusion . . . 126

Bibliography . . . 127

Appendix A – Development of Custom Algorithms for Specialized Tasks . . . 132
A.1 Computer Vision Algorithm . . . 132
A.2 Algorithm to Correct Duplicate or Intersecting Regions in FIR Shapefiles 135
A.3 Association of Flights and Storm Areas . . . 136
A.4 Additional Charts from Exploratory Analysis . . . 139

1 Introduction

1.1 Motivation

The ability to accurately predict air traffic is a crucial component of modern aviation logistics, particularly in a country as vast and dynamically complex as Brazil. As air travel continues to grow, the need for real-time, precise forecasting of aircraft movements within control regions becomes increasingly important. Effective air traffic prediction allows for better resource allocation, improved safety measures, and enhanced overall efficiency in airspace management.

The Brazilian Airspace Control Institute (ICEA) currently utilizes a comprehensive monitoring system to oversee air traffic within predefined control regions. These regions are dynamically managed, with a controller responsible for flights within their assigned area. However, this system faces significant challenges when air traffic exceeds the capacity of a single controller, necessitating the division of regions into smaller segments to ensure manageable workloads.

In this context, predicting the number of aircrafts in each control region on an hourly basis is essential. Such predictions enable proactive adjustments to control region configurations, ensuring optimal management of airspace and minimizing potential delays. The integration of predictive models into the ICEA's monitoring framework can provide controllers with advanced warnings, allowing for strategic planning and more efficient air traffic control.

The databases provided by ICEA, such as BIMTRA, CAT-62, Esperas, METAF, METAR, and meteorological satellite data, offer a wealth of information that can be leveraged to develop accurate predictive models. These databases contain critical data points, including hourly air traffic movements, radar synthesis data, flight wait quantities, meteorological forecasts, and runway change histories. By utilizing this comprehensive dataset, we aim to create a robust predictive model capable of estimating air traffic in control regions with a one-hour lead time.

Accurate air traffic prediction is not only vital for the operational efficiency of the

airspace management but also has significant implications for the logistics sector. Timely and precise forecasts enable better coordination of ground services, such as baggage handling, passenger boarding, and aircraft fueling. Additionally, it enhances the ability to manage unexpected events, such as adverse weather conditions or technical issues, by providing a buffer time to implement contingency plans.

This work aims to address these challenges by developing a predictive model that utilizes the smallest viable configuration of control regions. By investigating the underlying principles and practical challenges of air traffic prediction, we seek to contribute to the body of knowledge that supports the future of air traffic management in Brazil. The ultimate goal is to enhance the ICEA's capability to monitor and manage air traffic efficiently, ensuring safe and smooth operations across the nation's airspace.

1.2 Problem Definition

The primary problem addressed in this work is the accurate prediction of hourly air traffic within Brazilian FIRs. This prediction is crucial due to the dynamically changing nature of the control sectors and the limitations of the current monitoring and control systems.

1.2.1 Dynamic Nature of FIRs

FIRs in Brazil are predefined areas managed by controllers responsible for overseeing flights within their regions (WRAGG, 1973). However, these regions are not fixed; they must adapt to variations in air traffic volume. When the number of aircraft exceeds a controller's capacity, the region must be subdivided into smaller regions, increasing complexity and the need for precise management.

1.2.2 Challenges in Air Traffic Prediction

- **High Volume of Data:** The ICEA databases contain a vast amount of data, including hourly air traffic movements (BIMTRA), radar synthesis data (CAT-62), flight wait quantities (Esperas), meteorological forecasts (METAF and METAR), and runway change information. This volume of data presents a significant challenge in terms of processing and analyzing to extract meaningful predictions (ICEA, 2023).
- **Integration of Multiple Data Sources:** To develop an accurate predictive model, it is essential to integrate diverse data sources. Each data type has different characteristics and update frequencies, complicating the prediction process. Combining

these datasets effectively to produce reliable forecasts is a major challenge.

- **Timely Predictions:** The necessity for timely predictions adds another layer of complexity. ICEA requires air traffic forecasts with at least a one-hour lead time to make proactive adjustments and ensure efficient air traffic control. Achieving this level of timeliness while maintaining high accuracy is a significant challenge.

- **Hourly Prediction Challenges:** Predicting hourly air traffic poses a significant challenge due to the dynamic nature of flight arrivals and departures. Within the span of an hour, numerous flights may arrive at their destinations, potentially altering the composition of air traffic within FIRs. It is imperative to distinguish between ongoing flights and those that have completed their journeys to avoid overestimating air traffic levels. Developing algorithms capable of accurately tracking flight statuses and adjusting predictions accordingly is essential to ensure the reliability of hourly forecasts.

- **Unforeseen Events:** Predictive models must also be robust enough to handle unforeseen events such as sudden weather changes, technical issues, or unexpected increases in air traffic. Ensuring the model can adapt to these conditions without significant degradation in performance is critical.

1.3 Objectives

The primary objective of this work is to develop an accurate and reliable predictive model for hourly air traffic within Brazilian FIRs. This involves several key sub-objectives:

1.3.1 Development of a Robust Predictive Model

- **Data Analysis and Integration:** Analyze and integrate data from ICEA-provided databases, including BIMTRA, CAT-62, Flight Waits, METAF, METAR, and historical runway change data. This integration must consider the temporal and spatial characteristics of the data to ensure comprehensive coverage and reliability (ICEA, 2023).

- **Algorithm Development:** Develop sophisticated algorithms capable of handling the high volume and diversity of data. These algorithms must produce timely and accurate air traffic forecasts with a one-hour lead time. The focus will be on leveraging machine learning techniques and advanced statistical methods to enhance prediction accuracy.

- **Handling Unforeseen Events:** Ensure that the predictive model is robust and adaptable to unforeseen events such as sudden weather changes or unexpected spikes in air traffic. The model should maintain high performance under various scenarios, providing reliable forecasts even in the face of uncertainty.

1.3.2 Model Testing, Evaluation, and Replicability

- **Testing and Evaluation:** Implement rigorous testing procedures to evaluate the performance of predictive models. This includes testing the models on unseen data to assess their generalization capabilities and ensure they can accurately forecast air traffic under various conditions. Multiple models will be tested, and thorough evaluations will be conducted to identify the most effective approach.

- **Model Selection and Tuning:** Compare the performance of different predictive models and select the most suitable one based on predefined evaluation metrics. Parameters of the chosen model will be fine-tuned to optimize performance, ensuring the model's reliability and effectiveness in real-world scenarios.

- **Replicability and Transparency:** Ensure that the entire development process, including data preprocessing, model training, testing, and evaluation, is well-documented and replicable. This transparency is crucial for validating the robustness of the predictive model and facilitating future improvements or adaptations.

By addressing these challenges and meeting the objectives, this work aims to significantly enhance the ICEA's capability to manage air traffic effectively, ensuring safe, efficient, and reliable operations across Brazil's vast airspace.

2 Theoretical Foundation

2.1 Control Sector

Control sectors, also known as Flight Information Regions (FIRs), are specified regions of airspace in which a flight information service and an alerting service are provided. These sectors are essential for the organization and management of air traffic, ensuring safety and efficiency within the designated airspace. The International Civil Aviation Organization (ICAO) delegates the responsibility of operational control of a given FIR to specific countries, which then manage these regions according to international standards (International Civil Aviation Organization (ICAO), 2022).

2.1.1 Definition and Purpose

FIRs are the largest regular division of airspace in use worldwide and have been in existence since at least 1947 (International Civil Aviation Organization (ICAO), 2024). The primary purpose of an FIR is to provide flight information services (FIS) and alerting services (ALRS) to all aircraft operating within the region. This involves the dissemination of important information related to flight safety and the coordination of search and rescue operations when necessary. Each FIR is managed by a designated authority that ensures compliance with international regulations and standards set by the ICAO.

2.1.2 Structure and Management

In smaller countries, a single FIR typically encompasses the entire territorial airspace. In contrast, larger countries, such as Brazil, have their airspace divided into multiple regional FIRs to manage the high volume of air traffic more effectively (International Civil Aviation Organization (ICAO), 2022). Each FIR is further subdivided into smaller control sectors, each managed by a dedicated air traffic controller. These controllers are responsible for ensuring the safe and efficient flow of air traffic within their assigned sectors by providing instructions and information to pilots.

The boundaries of FIRs are not fixed and can extend into oceanic airspace. Oceanic information regions (OIRs) are similarly managed by controlling authorities based on international agreements facilitated by the ICAO (International Civil Aviation Organization (ICAO), 2022). The division of airspace into FIRs and control sectors allows for a structured approach to air traffic management, which is crucial for maintaining safety, especially in regions with high traffic density.

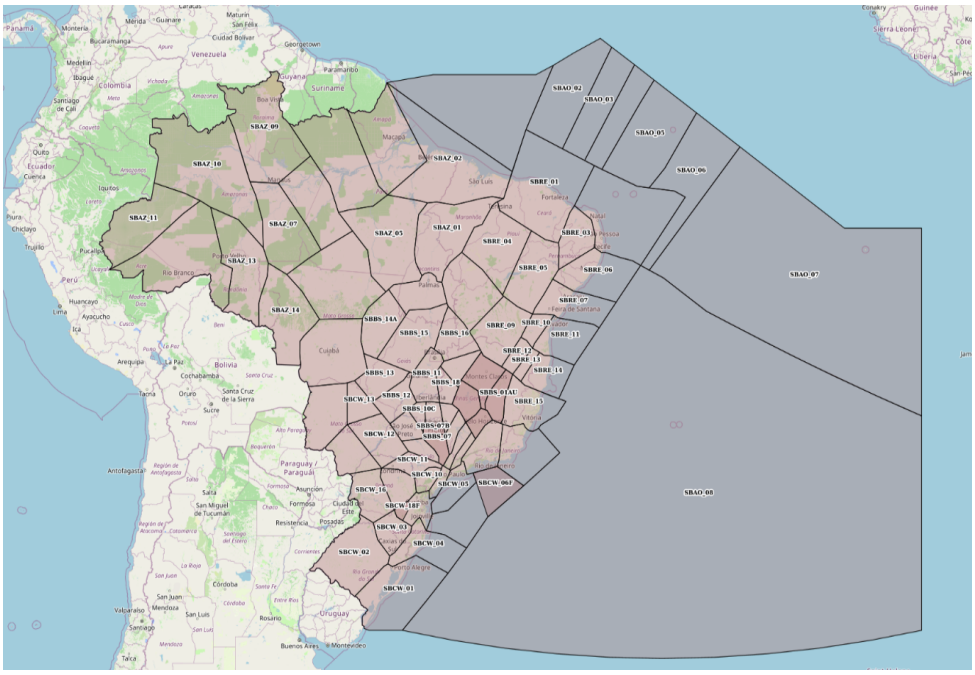

FIGURE 2.1 – Map of FIR sectors in Brazil. (Departamento de Controle do Espaço Aéreo (DECEA), 2024)

The image in Figure 2.1 illustrates the FIR sectors in Brazil. These sectors are delineated to efficiently manage air traffic within the country's airspace. Each FIR sector is responsible for a specific geographic area and is further subdivided into control sectors to facilitate the coordination of air traffic control operations.

By visualizing the FIR sectors, air traffic controllers can effectively monitor and manage air traffic flow, ensuring the safe and efficient movement of aircraft throughout Brazil's airspace.

2.1.3 Dynamic Nature of FIRs

The dynamic nature of air traffic requires FIRs to be flexible in their management. When air traffic within a sector exceeds the capacity of a single controller, the sector must be subdivided to ensure manageable workloads. This flexibility is vital for maintaining safety and efficiency in air traffic management. The need for accurate predictions of air traffic volume within FIRs is thus critical, as it allows for proactive adjustments to sector boundaries and controller assignments (International Civil Aviation Organization (ICAO), 2022).

By leveraging sophisticated predictive models and integrating comprehensive data from various sources, it is possible to enhance the management of FIRs significantly. Accurate hourly predictions of air traffic can help in making timely adjustments to control sectors, thus ensuring that air traffic controllers are not overwhelmed and that the flow of air traffic remains smooth and safe.

2.2 API

2.2.1 Introduction to APIs

An Application Programming Interface (API) is a set of rules and protocols for building and interacting with software applications. APIs enable different software systems to communicate with each other, facilitating the integration of various functions and data sources (FIELDING, 2000). They define the methods and data formats that applications can use to request and exchange information. This allows developers to access specific features or data of an application, service, or other system without needing to understand the internal workings of that system.

APIs are crucial in modern software development, especially when dealing with large datasets and multiple data sources. By using APIs, developers can automate the process of data retrieval, ensuring that applications are both efficient and up-to-date with the latest information. For this thesis, the ICEA API was instrumental in fetching comprehensive historical data on air traffic, which forms the foundation of the predictive models developed.

2.2.2 ICEA API

The Brazilian Airspace Control Institute (ICEA) provides a comprehensive API that facilitates access to a wide range of aviation data. This API is essential for the development of accurate predictive models for air traffic management, as it offers real-time and historical data critical for analysis (ICEA, 2023).

2.2.2.1 Overview of ICEA API

The ICEA API is designed to provide users with seamless access to various datasets relevant to air traffic management. These datasets include information on air traffic movements, radar synthesis, flight wait quantities, meteorological forecasts, and runway change histories. By integrating these diverse data sources, the API enables the development of

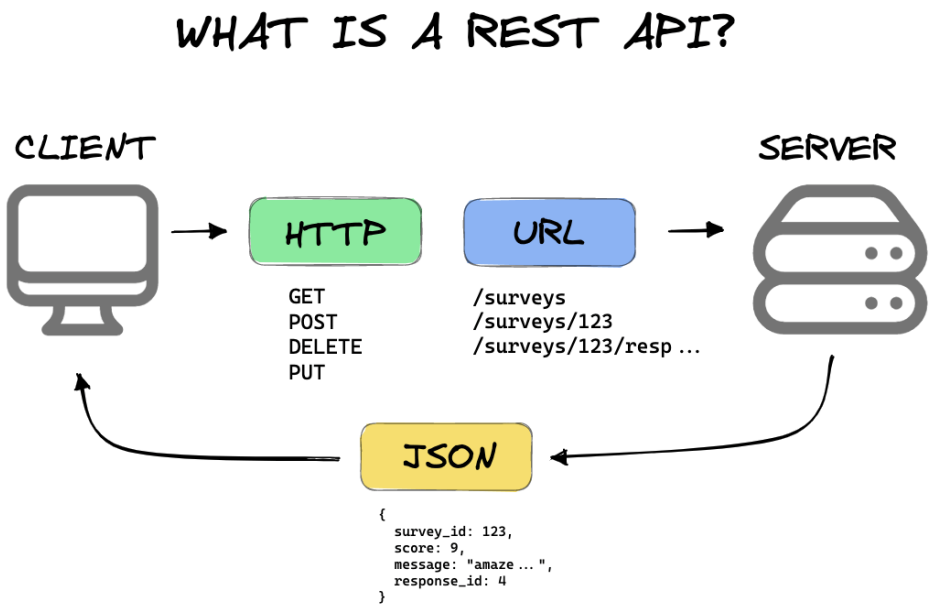

FIGURE 2.2 – Diagram illustrating the concept of RESTful API. (Contentful, 2024)

sophisticated predictive models that can enhance the efficiency and safety of air traffic operations.

2.2.2.2 Key Datasets

The ICEA API provides access to several key datasets:

- **BIMTRA (Air Traffic Movement Information Bank)**
  - Endpoint: `/bimtra`
  - Description: Contains information about air traffic movement. Includes basic flight data such as departure time and arrival forecast.
  - Frequency: Hourly
- **CAT-62 (Radar Synthesis Data)**
  - Endpoint: `/cat-62`
  - Description: Provides historical radar synthesis data.
  - Frequency: Every 2 minutes
- **Esperas (Flight Wait Quantities Data per Hour)**
  - Endpoint: `/esperas`
  - Description: Contains historical information about the quantity of flight waits per hour at aerodromes.

  - Frequency: Hourly

- **METAF (Terminal Aerodrome Forecast)**
  - Endpoint: `/metaf`
  - Description: Offers meteorological forecasts at aerodromes.
  - Frequency: Hourly

- **METAR (Aerodrome Meteorological Report)**
  - Endpoint: `/metar`
  - Description: Contains meteorological reports at aerodromes.
  - Frequency: Hourly

- **Meteorological Satellite**
  - Endpoint: `/satelite`
  - Description: Provides meteorological satellite data in images
  - Frequency: Hourly

- **Aerodrome Runway Change Forecast**
  - Endpoint: `/tc-prev`
  - Description: Offers (historical) forecasts on aerodrome runway changes.
  - Frequency: Hourly

- **Aerodrome Runway Change History**
  - Endpoint: `/tc-real`
  - Description: Contains information about historical aerodrome runway changes.
  - Frequency: Hourly

2.2.2.3 Accessing the ICEA API

To access the ICEA API, users must obtain the necessary credentials and API keys from the ICEA website. The API documentation, available at (Departamento de Controle do Espaço Aéreo (DECEA), 2024), provides detailed instructions on how to authenticate and interact with the API endpoints. Each dataset is accessible through specific endpoints, which return data in a structured format suitable for analysis.

2.2.2.4 Usage Example

Here is a basic example of how to fetch data from the ICEA API for the BIMTRA dataset using a simple curl command:

```
# Define variables for the URL parts
base_url="http://montreal.icea.decea.mil.br:5002/api/v1/"
token="YOUR_API_KEY"
idate="2022-05-01"
fdate="2022-05-02"

# Concatenate the URL using variables
url="$base_url/bimtra?token=$token&idate=$idate&fdate=$fdate"

# Use the URL variable in the curl command
curl -X GET "$url" -H "accept: application/json"
```

Listing 2.1 – Fetching BIMTRA Data with Curl

This command demonstrates how to make an authenticated request to the BIMTRA endpoint to retrieve data for a specific date range. Similar methods can be applied to access other datasets provided by the ICEA API.

2.3 Image Processing

In this section, the processing of meteorological satellite data, which is crucial for understanding weather patterns that impact aviation, is delved into. As explained previously, meteorological satellite data was accessed via the ICEA API's '/satelite' endpoint, providing images such as those shown in Figure 2.3.

Analyzing satellite images is crucial because they provide real-time and historical data on atmospheric conditions like cloud cover, precipitation, and temperature distributions, which are vital for predicting and understanding weather patterns. By incorporating these extracted features into machine learning models, weather-related predictions can be significantly enhanced, leading to more accurate and reliable forecasts. These improvements in predictive capabilities optimize flight operations, mitigate risks associated with adverse weather conditions, and ensure the safety and efficiency of aviation activities.

2.3.1 Introduction to Image Processing

Image processing involves the manipulation and analysis of images to extract meaningful information. It is widely used in various fields, including meteorology, where satellite images are processed to monitor weather conditions (GONZALEZ; WOODS, 2002).

1. **Reading and Cropping Images:**

Satellite images are read into the system using image processing libraries such as OpenCV (OpenCV, 2024). The images are cropped to focus on the region of interest, removing unnecessary borders to ensure that only the relevant area is analyzed.

2. **Converting to Grayscale and Applying Morphological Transformations:**

The images are converted to grayscale to simplify the data and enhance important features. Grayscale conversion reduces the complexity of the image by eliminating color information, retaining only intensity information. Morphological transformations, such as closing, are then applied. Closing involves dilating the image and then eroding it, which helps to fill small holes and connect adjacent regions, enhancing the features that will be used for contour detection.

3. **Thresholding and Contour Detection:**

Thresholding is applied to the grayscale images to create binary images. This process involves setting a threshold value, and all pixel values above this threshold are set to the maximum value (typically 255), while all values below the threshold are set to zero. This simplifies the image to highlight storm areas as distinct regions. Contour detection algorithms are then used to identify the boundaries of these regions. Contours are curves joining all the continuous points along a boundary with the same color or intensity.

4. **Polygon Approximation and Centroid Calculation:**

The detected contours are approximated using polygons to simplify their shapes. Polygon approximation reduces the number of vertices of the contours based on a specified accuracy. The centroids of these polygons are calculated by finding the geometric center of each polygon. These centroids are then converted from pixel coordinates to geographical coordinates using a transformation function. This conversion involves mapping pixel coordinates to real-world latitude and longitude coordinates, enabling accurate geographical mapping of storm areas.

The described process enables the identification and geographical mapping of weather phenomena that can impact flight operations, providing crucial information for predictive models.

2.3.2 Processing Meteorological Satellite Images

The processing of meteorological satellite images is essential for identifying storm regions and other weather phenomena that could affect aviation. The following steps outline the approach used to analyze these images (GONZALEZ; WOODS, 2002):

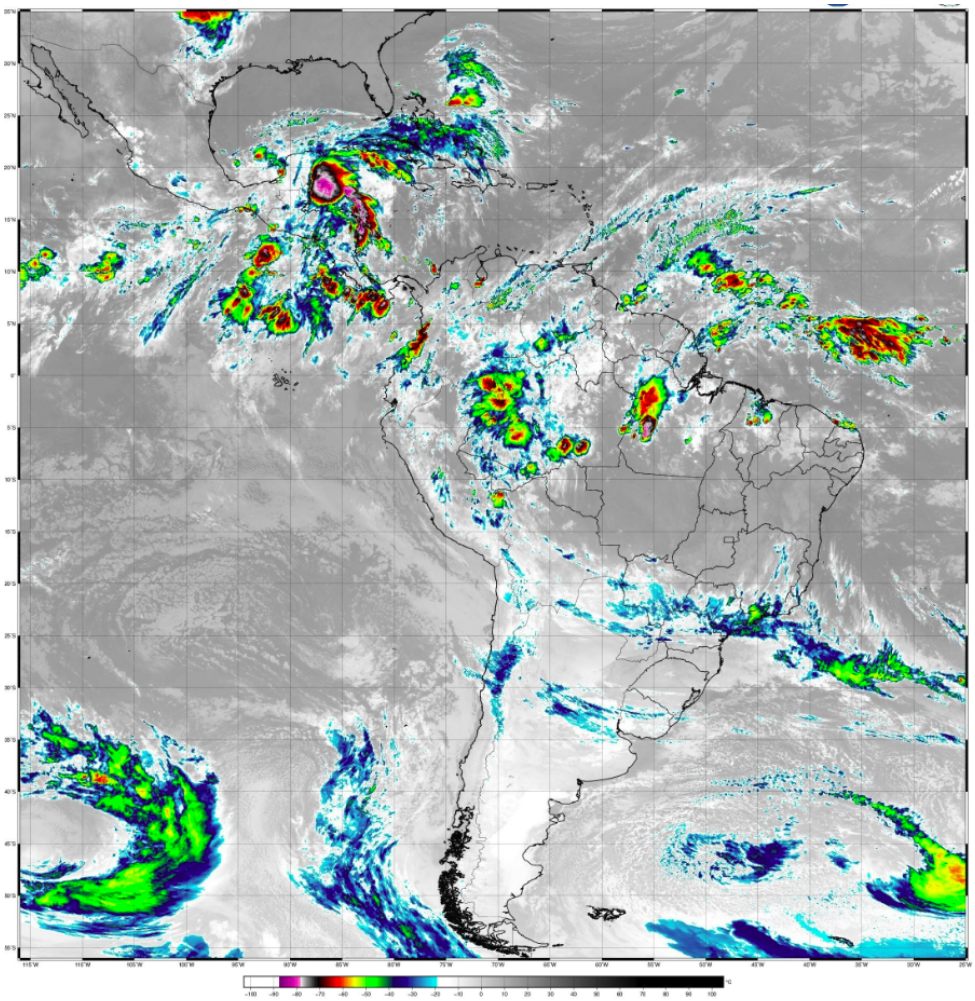

FIGURE 2.3 – Processed satellite image showing detected storm areas. (ICEA, 2023)

1. **Acquisition and Preprocessing of Images:**

   Meteorological satellite images are acquired through APIs such as the ICEA API's `/satelite` endpoint. The images are then preprocessed using OpenCV to focus on the region of interest by removing extraneous borders. This preprocessing step ensures that only the relevant areas are analyzed, enhancing the efficiency of subsequent processing steps (OpenCV, 2024).

2. **Grayscale Conversion and Morphological Operations:**

   To reduce the complexity of the image data, the satellite images are converted to grayscale. This conversion retains only the intensity information, making it easier to highlight significant features. Morphological operations, such as the closing technique, are applied to the grayscale images. This involves dilation followed by erosion, which helps to close small gaps and connect disjointed regions, making feature detection more robust.

3. **Binary Thresholding and Feature Extraction:**

After grayscale conversion, binary thresholding is applied to create a binary image. This step involves setting a threshold value where pixel values above the threshold are set to the maximum value (typically 255), and values below the threshold are set to zero. This simplification helps to isolate storm regions as distinct entities. Feature extraction techniques, including contour detection, are then used to delineate the boundaries of these regions, enabling further analysis.

4. **Simplification and Geolocation of Features:**

   The contours detected in the binary images are approximated with polygons to simplify their shapes and reduce the number of vertices. This simplification makes it easier to work with the data and perform further analysis. The centroids of these polygons are calculated to find the geometric centers. These centroids are then transformed from pixel coordinates to geographical coordinates using mapping functions, allowing for accurate geospatial analysis of the identified weather phenomena.

2.3.3 Identifying Flights Passing Through Storm Areas

Determining whether a flight passes through a storm area is crucial for enhancing the accuracy and reliability of air traffic predictions. This process involves several key steps to ensure accurate spatial analysis and integration with flight data:

1. **Transforming Coordinates:** The first step is to convert the pixel coordinates of detected storm polygons into geographical coordinates (latitude and longitude). This transformation uses known reference points within the satellite images and interpolation methods. Accurate geolocation of these storm polygons is essential for aligning them correctly with real-world geographical features.

2. **Spatial Joins:** Spatial joins are then performed to compare the flight paths with the storm polygons. A spatial join is a process that matches rows from two datasets based on their spatial relationship. In this case, it involves checking whether any part of a flight path intersects with the storm polygons. Geospatial libraries, such as Shapely and GeoPandas, provide tools for handling geometric operations and spatial relationships efficiently, facilitating this analysis (GeoPandas Development Team, 2013–2024).

3. **Enhancing Predictive Models:** By incorporating the results of these spatial joins into predictive models, we can account for the impact of weather patterns on flight operations. This integration helps to improve the accuracy and reliability of air traffic predictions by providing a more comprehensive understanding of potential disruptions caused by adverse weather conditions.

2.4 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a critical step in the data science process that involves summarizing the main characteristics of a dataset, often using visual methods. It is essential for understanding the underlying patterns, detecting anomalies, and testing hypotheses before proceeding to more complex analyses or modeling (TUKEY, 1977).

EDA serves multiple purposes:

- **Understanding Data Distribution:** Visualizing data helps in understanding the distribution of variables, identifying central tendencies, and detecting any skewness or kurtosis.

- **Identifying Outliers and Missing Values:** Through visual and statistical methods, EDA helps in spotting outliers and missing values that need to be addressed in subsequent data preprocessing steps (TUKEY, 1977).

- **Discovering Relationships:** EDA enables the identification of relationships and correlations between variables, which can inform feature engineering and model selection.

- **Generating Hypotheses:** By exploring the data, analysts can generate and test new hypotheses, refining their understanding of the data.

In this thesis, EDA was performed to gain insights into the air traffic dataset. Figure 2.4 exemplifies what can be discovered about the air traffic throughout the day.

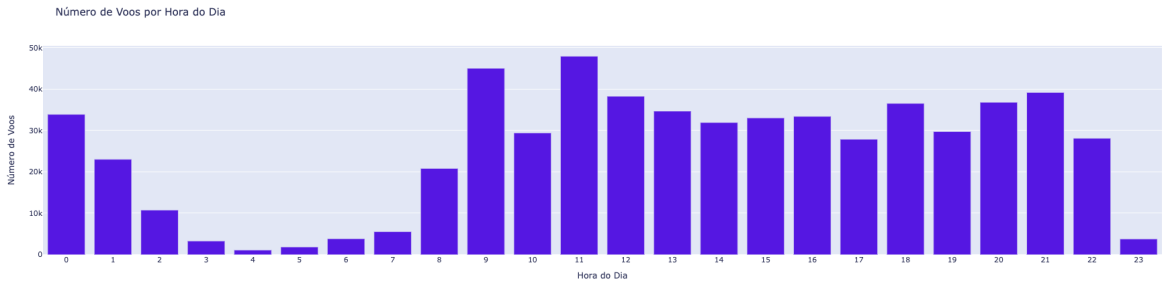

FIGURE 2.4 – Number of flights per hour of the day.

2.5 Data Preparation

Data preparation is a crucial step in any machine learning workflow, involving the collection, cleaning, and transformation of raw data into a suitable format for model training and evaluation. Effective data preparation can significantly enhance the performance and accuracy of machine learning models. This section details the specific methods and techniques used for data preparation.

2.5.1 Data Fetching

Data fetching is a critical step in any data analysis workflow, involving the acquisition of data from various sources. This data can be obtained through downloading files directly or extracting data from APIs. Depending on the project's requirements, data can be saved either locally or in cloud storage for further processing.

Downloading Files Files can be downloaded from websites or FTP servers using various tools and libraries, such as `wget` for command-line downloading or Python's `requests` (REITZ, 2023) library for programmatic access. The downloaded files are then saved to a specified directory for subsequent use.

Extracting Data from APIs APIs (Application Programming Interfaces) provide a structured way to access and retrieve data from web services. Using Python libraries like `requests`, data can be fetched in formats such as JSON or XML and then processed into a usable format like CSV or directly into a database. Depending on the volume and frequency of the data, it can be stored locally or uploaded to cloud storage services such as Google Drive or AWS S3 for better accessibility and management.

Saving Data Once fetched, the data can be stored locally on disk or in cloud storage. Cloud storage solutions provide advantages such as scalability, remote access, and data backup. Tools like Google Colab (Google, 2024) integrate seamlessly with Google Drive, allowing for easy storage and retrieval of data files.

2.5.2 Efficient Data Fetching with ThreadPool

Fetching large volumes of data can be time-consuming, especially when dealing with numerous API requests or large files. To expedite this process, concurrent programming techniques such as using a ThreadPool can be employed (Python Software Foundation, 2023).

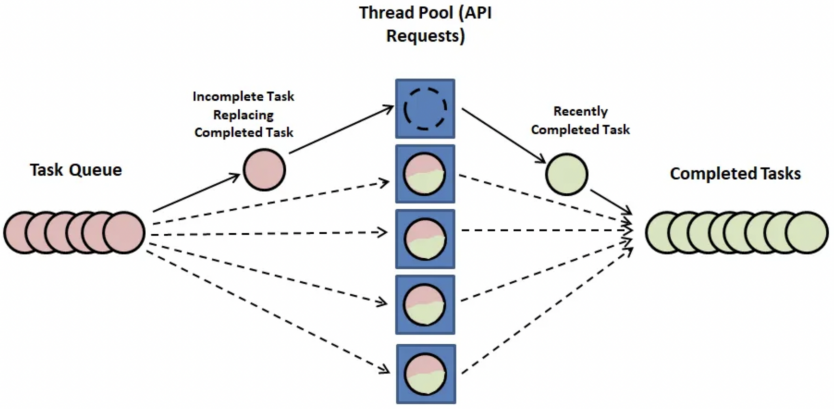

FIGURE 2.5 – Using ThreadPool to speed up API requests. Source: (CARTER, 2020)

A ThreadPool is a method of parallelizing tasks by utilizing multiple threads. This is particularly useful for I/O-bound tasks like data fetching, where the speed is limited by the time it takes to read or write data rather than by computational complexity.

In Python, the `concurrent.futures` module provides a high-level interface for asynchronously executing callables. The `ThreadPoolExecutor` class within this module allows for managing a pool of threads to execute tasks concurrently.

```
from concurrent.futures import ThreadPoolExecutor, as_completed
import requests

# Function to fetch data from a URL
def fetch_data(url):
    response = requests.get(url)
    return response.content

# List of URLs to fetch data from
urls = [
    'https://example.com/data1.csv',
    'https://example.com/data2.csv',
    # more URLs
]

# Using ThreadPoolExecutor to fetch data concurrently
with ThreadPoolExecutor(max_workers=5) as executor:
    futures = [executor.submit(fetch_data, url) for url in urls]
    for future in as_completed(futures):
        data = future.result()
        # Process the fetched data
```

Listing 2.2 – Using ThreadPoolExecutor for Concurrent Data Fetching

The use of `ThreadPoolExecutor` allows multiple data fetching tasks to be executed simultaneously, significantly reducing the total time required for the operation. This approach is especially beneficial when dealing with a large number of requests or when the data source has rate limits.

2.5.3 Data Integration

The first step involves integrating data from multiple sources. This includes merging datasets on common keys, such as flight IDs, to create a comprehensive dataset that combines information from different databases. Techniques like SQL joins and pandas merge functions in Python are typically used for this purpose.

2.5.4 Feature Engineering

Feature engineering is the process of transforming raw data into features that can be used in machine learning algorithms. This step is crucial as it directly impacts the performance of predictive models.

2.5.4.1 Overview

The feature engineering pipeline typically involves the following steps (FLAXMAN, 2022):

Feature Creation: Creating features involves identifying and generating new variables that capture underlying patterns in the data. This process often requires human intervention and creativity. For example, combining existing features through addition, subtraction, multiplication, and ratios can produce derived features with greater predictive power.

Transformations: Transformations manipulate predictor variables to improve model performance by ensuring variables are on the same scale and within an acceptable range for the model. Common transformations include log transformations, polynomial features, or interaction terms.

Feature Extraction: Feature extraction automatically reduces the volume of data into a more manageable set by extracting new variables from raw data. Techniques such as cluster analysis, text analytics, edge detection algorithms, and principal component analysis are often used for this purpose.

Feature Selection: Feature selection algorithms analyze, judge, and rank various features to determine their relevance. Irrelevant and redundant features are removed, while the most useful features are prioritized for the model, enhancing its accuracy and efficiency.

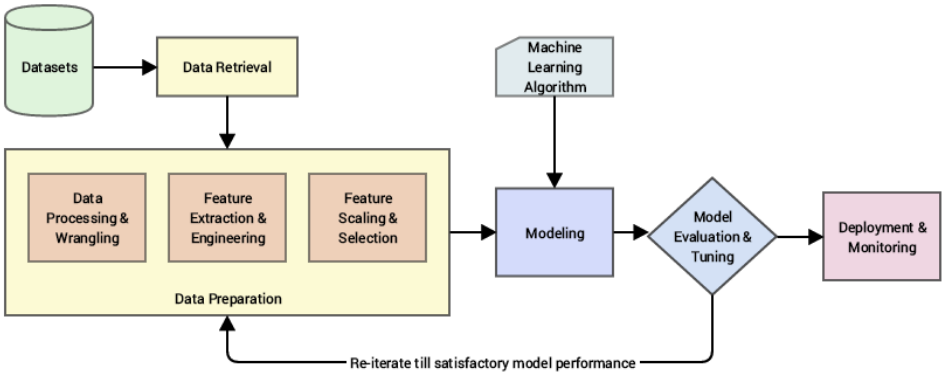

FIGURE 2.6 – Illustration of the Feature Engineering process (FLAXMAN, 2022).

2.5.4.2 Data Integrity Verification

Ensuring the integrity of data is crucial for reliable model training. This involves several key steps, which are not exhaustive but highlight the main procedures relevant to the problem at hand (LITTLE; RUBIN, 2014):

- **Handling Missing Values:** Missing data can lead to biased results and inaccurate models. It is essential to address this issue because models trained on incomplete data may not generalize well to new data. By identifying and appropriately handling missing values—either through imputation (filling in the gaps with plausible values) or by removing records with missing data—the dataset remains representative and complete. This process improves the accuracy and robustness of the model.

- **Outlier Detection and Removal:** Outliers are data points that deviate significantly from the majority of the data. These anomalies can distort statistical analyses and model predictions, leading to unreliable results. Detecting and managing outliers ensures that the model's performance is not unduly influenced by these extreme values. Removing or appropriately treating outliers helps in building a model that better reflects the underlying data distribution, resulting in more reliable and accurate predictions.

Ensuring data integrity through these methods helps in building robust and reliable machine learning models by providing clean and accurate data for training.

2.5.4.3 Final Data Preparation Steps

The final steps in data preparation may include:

Data Scaling with Standard Scaler

Standardizing features ensures that all features contribute equally to the model's performance. In this thesis, standard scaling was applied, transforming features to have zero mean and unit variance:

$$z = \frac{x - \mu}{\sigma}$$

where x is the feature value, μ is the mean of the feature, and σ is the standard deviation (LITTLE; RUBIN, 2014). This scaling is crucial because it ensures that the model does not become biased towards features with larger magnitudes, leading to better model performance and convergence during training.

Additionally, it is essential to save the trained scaler used during the training phase. The same scaler must be applied to any new data that the model will predict on. Retraining the scaler on new data is not advisable, as the statistical properties of the data may

change, leading to different mean and standard deviation values. If the original scaler is not used, the transformed feature values will differ, causing the model to produce incorrect predictions. Therefore, saving the scaler ensures consistency in feature scaling between the training and prediction phases, maintaining the model's accuracy and reliability.

Data Splitting

Dividing the dataset into training and testing sets is essential to evaluate the model's performance. The training set is used to train the model, while the testing set is used to assess how well the model generalizes to new, unseen data. This step helps in understanding the model's performance and identifying any overfitting issues.

Balancing the Dataset

Addressing class imbalance is crucial for models that are sensitive to the distribution of classes. Techniques such as SMOTE (Synthetic Minority Over-sampling Technique) are used to ensure that the model is trained on a balanced dataset. SMOTE generates synthetic samples for the minority class by interpolating between existing samples, which helps in improving the model's performance on imbalanced datasets (CHAWLA *et al.*, 2002).

FIGURE 2.7 – Illustration of SMOTE oversampling technique (ORELLANA, 2020).

Balancing the dataset ensures that the model does not become biased towards the majority class, leading to more accurate and fair predictions.

2.5.5 Geographic Data Processing

Geographic data processing is a fundamental aspect of spatial data analysis, involving a variety of techniques and tools designed to manage, manipulate, and analyze geographic information. In the context of this thesis, geographic data processing is essential for accurately predicting aircraft counts within different sectors and understanding the spatial distribution of these counts. This section provides an introduction to the core concepts and methodologies of geographic data processing.

2.5.5.1 Spatial Data Representation

Spatial data represents information about the physical location and shape of objects on the Earth. This data can be classified into two main types: vector data and raster data. Vector data represents geographic features as points, lines, and polygons, while raster data represents geographic phenomena as grid cells or pixels. Each type of data has its specific applications and is chosen based on the nature of the spatial analysis to be performed (DEMPSEY, 2024).

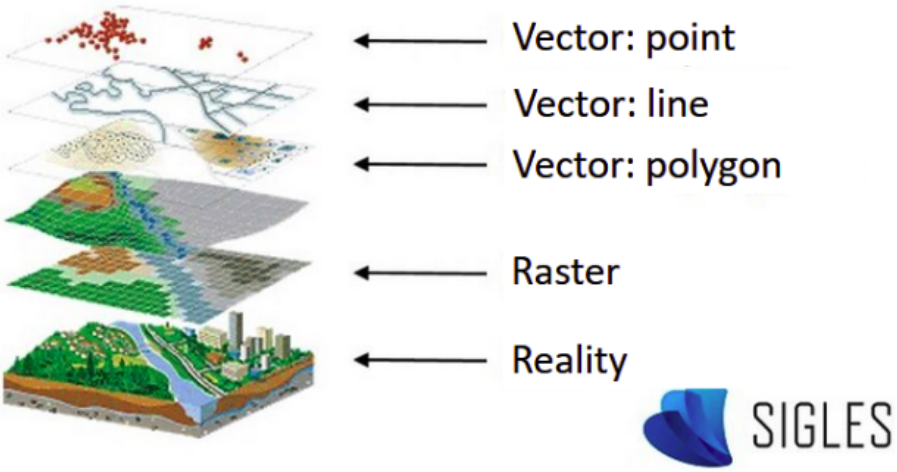

FIGURE 2.8 – Example of Spatial Data Representation (SIGLES Programme, 2018)

2.5.5.2 Spatial

One of the primary tasks in geographic data processing is the execution of spatial operations, which include spatial joins, buffering, overlay, and spatial indexing. Spatial joins involve combining datasets based on their geographic relationship, such as linking points representing aircraft locations with polygons representing airspace sectors. This technique is crucial for identifying which geographic regions contain specific data points, enabling comprehensive spatial analysis.

In addition to spatial joins, spatial buffering is used to create zones around geometric features, which is essential for determining proximity and influence areas. Overlay operations, such as union, intersection, and difference, allow the combination and comparison of different spatial layers to derive new information, such as determining the overlap between airspace zones and weather patterns. Spatial indexing, on the other hand, optimizes the querying and retrieval of spatial data, enhancing the efficiency of spatial operations in large datasets.

GeoPandas is a powerful library in Python that extends the capabilities of pandas to support spatial operations on geometric types. It provides tools for reading, manipulating, and visualizing geographic data, making it an essential tool for spatial data analysis (GeoPandas Development Team, 2013–2024). GeoPandas simplifies the process of performing spatial joins, buffering, overlay, and spatial indexing, facilitating efficient geographic data processing.

2.5.5.3 Distance Calculations

Calculating distances between geographic points is another critical aspect of geographic data processing. The Haversine formula is widely used for this purpose, especially when dealing with large distances on the Earth's surface. The formula calculates the great-circle distance between two points, accounting for the Earth's curvature. (DISTANCES..., 2023).

The Haversine formula is given by:

$$d = 2 \times 6371 \times \arcsin\left(\sqrt{\sin^2\left(\frac{f.lat - a.lat}{2}\right) + \cos(f.lat) \times \cos(a.lat) \times \sin^2\left(\frac{f.lon - a.lon}{2}\right)}\right)$$

where $f.lat$ and $f.lon$ are the latitude and longitude of the first point, $a.lat$ and $a.lon$ are the latitude and longitude of the second point, and 6371 km is the Earth's radius.

2.6 Machine Learning

Machine Learning (ML) is a branch of artificial intelligence (AI) focused on developing algorithms that allow computers to learn from and make decisions based on data. Instead of being explicitly programmed to perform specific tasks, ML algorithms identify patterns in data and use these patterns to make predictions or decisions. This approach is particularly powerful in situations where traditional programming would be infeasible due to the complexity or variability of the task (BISHOP, 2006).

ML can be broadly categorized into three types:

1. **Supervised Learning:** In this approach, the algorithm is trained on a labeled dataset, which means that each training example is paired with an output label. The goal is to learn a mapping from inputs to outputs that can be used to predict labels for new, unseen data. Common applications include classification (e.g., identifying spam emails) and regression (e.g., predicting house prices) (MURPHY, 2012).

2. **Unsupervised Learning:** Here, the algorithm is given data without explicit instructions on what to do with it. Instead, it tries to identify underlying structures or patterns within the data. Clustering (e.g., customer segmentation) and dimensionality reduction (e.g., principal component analysis) are typical examples of unsupervised learning (HASTIE

et al., 2009).

3. **Reinforcement Learning:** This type of learning involves training an agent to make sequences of decisions by rewarding it for desirable actions and penalizing it for undesirable ones. It is often used in scenarios where the learning process involves a series of steps, such as game playing or robotic control (SUTTON; BARTO, 2018).

Applications of Machine Learning: ML is used in a wide range of applications, from simple tasks like email filtering to complex systems like self-driving cars. In the field of air traffic management, ML can be utilized to predict air traffic patterns, identify potential delays, and optimize routing to enhance safety and efficiency. The ability to process and analyze vast amounts of data in real-time makes ML an invaluable tool in this context .

Historical Perspective: The concept of machine learning dates back to the mid-20th century, but it has gained significant traction in recent decades due to advancements in computing power, the availability of large datasets, and the development of sophisticated algorithms. Early works by Alan Turing and later developments by Arthur Samuel laid the groundwork for modern ML techniques (TURING, 1950; SAMUEL, 1959).

Challenges and Considerations: Despite its potential, ML comes with challenges. These include the need for large amounts of high-quality data, the risk of overfitting (scenario in which a model learns the training data too well but fails to generalize to new data), and ethical considerations related to bias and privacy. Addressing these challenges requires careful model selection, rigorous validation techniques, and adherence to ethical standards.

2.6.1 Classification Models

Classification models are used to categorize data into predefined classes. These models learn from labeled training data to make predictions or classifications on new, unseen data. Classification is one of the most fundamental tasks in machine learning and is widely used in various applications, including image recognition, spam detection, and medical diagnosis (SUTTON; BARTO, 2018). In this thesis, the following classification models were employed:

2.6.1.1 MLP Classifier

The Multilayer Perceptron (MLP) is a type of artificial neural network consisting of multiple layers of nodes (neurons). An MLP typically includes an input layer, one or more hidden layers, and an output layer. Each node in a layer is connected to every node in the subsequent layer, forming a fully connected network. The MLP is particularly effective

for various classification tasks due to its ability to capture complex patterns in the data (BISHOP, 2006).

Structure of MLP:

- **Input Layer:** This layer receives the input data. The number of nodes in this layer corresponds to the number of input features.

- **Hidden Layers:** These layers perform nonlinear transformations of the input data through weighted connections and activation functions. The number of hidden layers and nodes per layer can be adjusted based on the complexity of the task.

- **Output Layer:** This layer produces the final classification output. The number of nodes in this layer corresponds to the number of target classes.

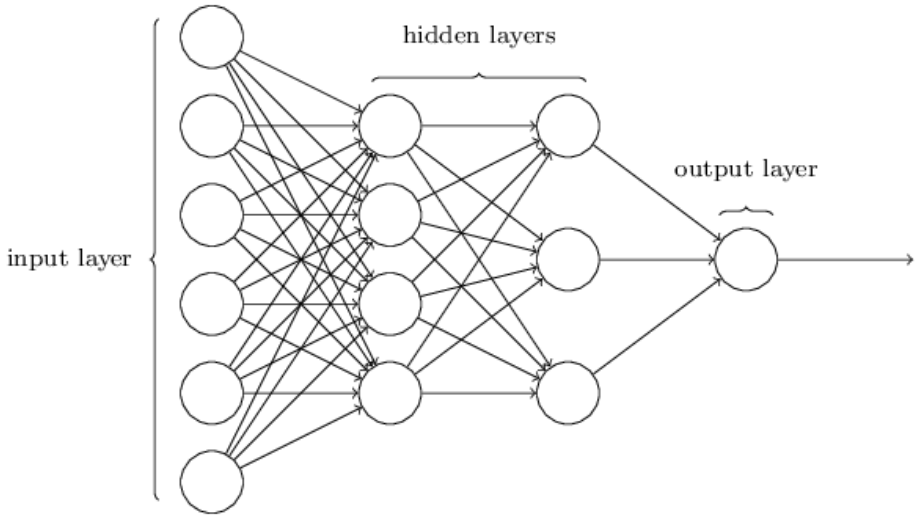

FIGURE 2.9 – Structure of an MLP with its layers (ENDEAVOURS, 2019).

Mathematical Formulation:

The operation of an MLP can be described using the following equations. Let $\mathbf{x}$ be the input vector, $\mathbf{W}^{(l)}$ be the weight matrix for layer l, $\mathbf{b}^{(l)}$ be the bias vector for layer l, and f be the activation function (GOODFELLOW *et al.*, 2016).

1. **Forward Propagation:**

For each layer l from 1 to L (the number of layers), the output $\mathbf{h}^{(l)}$ is computed as:

$$\mathbf{h}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$$

where $\mathbf{h}^{(0)} = \mathbf{x}$ (the input vector).

2. **Activation Function:**

Common activation functions include the sigmoid function, hyperbolic tangent (tanh),

and Rectified Linear Unit (ReLU):

$$\text{Sigmoid: } f(z) = \frac{1}{1 + e^{-z}}$$

$$\text{Tanh: } f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\text{ReLU: } f(z) = \max(0, z)$$

3. **Loss Function:**

The loss function measures the discrepancy between the predicted output and the actual output. For classification tasks, the cross-entropy loss is commonly used:

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k} \log(\hat{y}_{i,k})$$

where N is the number of samples, K is the number of classes, $y_{i,k}$ is the binary indicator (0 or 1) if class label k is the correct classification for sample i, and $\hat{y}_{i,k}$ is the predicted probability of sample i being in class k.

4. **Backpropagation:**

Backpropagation is the process of updating the weights to minimize the loss function. It involves computing the gradient of the loss function with respect to each weight by the chain rule and then updating the weights using gradient descent:

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}$$

where η is the learning rate.

2.6.1.2 Random Forest Classifier

The Random Forest classifier is an ensemble learning method that operates by constructing multiple decision trees during training and outputting the mode of the classes (classification) of the individual trees. It is widely used due to its simplicity and effectiveness in various classification tasks (BISHOP, 2006).

Structure of Random Forest:

- **Decision Trees:** A Random Forest consists of multiple decision trees, each trained on a different subset of the training data. These trees are constructed using a process

called bootstrap aggregation (bagging), which helps in reducing variance and preventing overfitting.

- **Ensemble Method:** The final prediction of the Random Forest is determined by aggregating the predictions of all individual trees, typically through majority voting in classification tasks.

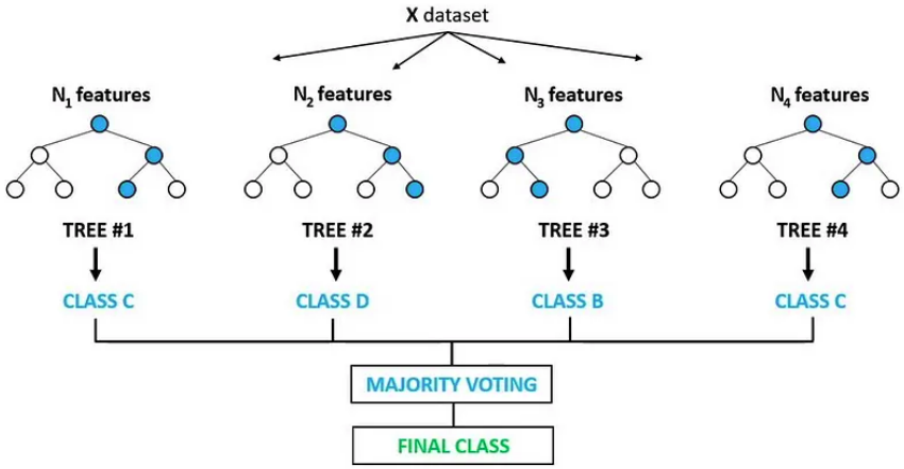

FIGURE 2.10 – Illustration of a Random Forest structure (MASTER, 2019).

Mathematical Formulation:

The operation of a Random Forest can be described using the following steps. Let $T_1, T_2, \ldots, T_B$ be the B decision trees in the forest, and let $\mathcal{D}$ be the training data.

1. **Bootstrap Sampling:**

For each tree T_b (where $b = 1, 2, \ldots, B$), a bootstrap sample $\mathcal{D}_b$ is created by randomly sampling n instances with replacement from the training data $\mathcal{D}$ (BREIMAN, 2001).

2. **Tree Construction:**

Each tree T_b is trained on the bootstrap sample $\mathcal{D}_b$ by recursively splitting the data at each node. The splits are determined by maximizing the information gain or minimizing the Gini impurity (BISHOP, 2006).

The information gain IG for a split is given by:

$$IG = H(D) - \sum_{i=1}^{k} \frac{|D_i|}{|D|} H(D_i)$$

where $H(D)$ is the entropy of the dataset D and D_i are the subsets resulting from the split.

The Gini impurity I_G is given by:

$$I_G = 1 - \sum_{i=1}^{C} p_i^2$$

where p_i is the probability of an instance being classified into class i and C is the total number of classes (BISHOP, 2006).

3. **Prediction:**

To make a prediction for a new instance $\mathbf{x}$, each tree T_b in the forest outputs a class prediction $\hat{y}_b(\mathbf{x})$. The final prediction of the Random Forest is the majority vote among the predictions of the individual trees:

$$\hat{y} = \text{mode}(\{\hat{y}_1(\mathbf{x}), \hat{y}_2(\mathbf{x}), \ldots, \hat{y}_B(\mathbf{x})\})$$

2.6.1.3 XGBoost Classifier

XGBoost (Extreme Gradient Boosting) is an ensemble learning method that uses the boosting technique to create a strong model from a combination of weak models, typically decision trees. It is widely recognized for its efficiency and accuracy in various classification and regression tasks (CHEN; GUESTRIN, 2016a).

Structure of XGBoost:

- **Boosting:** XGBoost utilizes the boosting technique, where sequential models are trained to correct the errors of the previous models. Each new model is adjusted to minimize the error of the combined model.

- **Regularization:** XGBoost includes regularization terms in its objective function to prevent overfitting, thereby improving the model's generalization.

Mathematical Formulation:

The operation of XGBoost can be described using the following steps. Let $\mathcal{D}$ be the training dataset with n examples and m features.

1. **Objective Function:**

The objective function in XGBoost consists of the loss function and a regularization term to control the complexity of the model (CHEN; GUESTRIN, 2016a):

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where l is the loss function, y_i is the true label, $\hat{y}_i$ is the prediction, $\Omega(f_k)$ is the regularization term, and K is the number of trees.

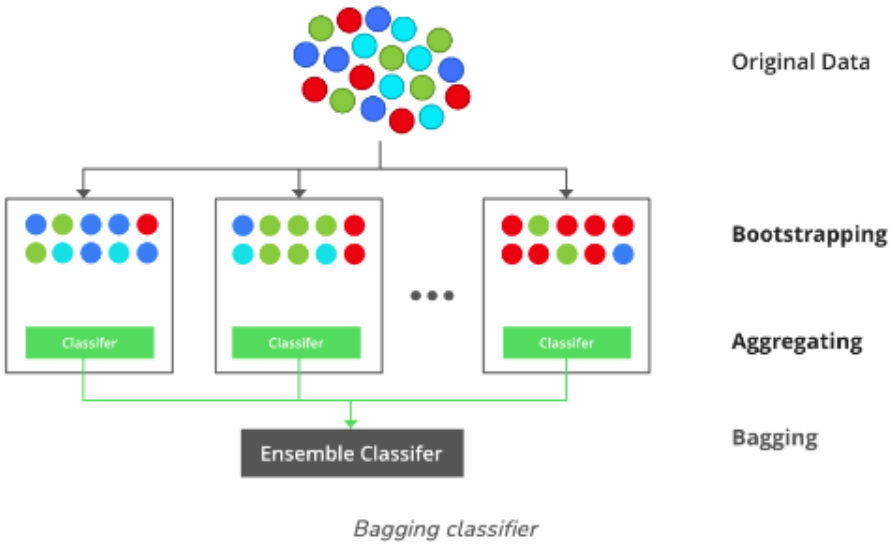

FIGURE 2.11 – Structure of XGBoost with its decision trees. (GEEKSFORGEEKS, 2023)

2. **Loss Function:**

The common loss function for classification problems is logistic loss:

$$l(y_i, \hat{y}_i) = -[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

3. **Regularization Term:**

The regularization term in XGBoost is given by:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where T is the number of leaves in the tree, w_j is the weight of leaf j, γ and λ are regularization parameters.

4. **Additive Training:**

XGBoost adds new models to minimize the loss function. The prediction at iteration t is updated as:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

where f_t is the new tree added at iteration t.

5. **Gradient and Hessian:**

To optimize the objective function, XGBoost uses the gradient and hessian:

$$g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}$$

$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}$$

where g_i and h_i are the gradient and hessian for example i, respectively (CHEN; GUESTRIN, 2016a).

2.6.1.4 Logistic Regression

Logistic Regression is a statistical method used for binary classification problems. It models the probability that a given input belongs to a particular class, making it a classification algorithm rather than a regression algorithm despite its name. Logistic regression is widely used due to its simplicity and effectiveness in various applications (BISHOP, 2006).

Structure of Logistic Regression:

- **Binary Classification:** Logistic regression is primarily used to predict binary outcomes, i.e., whether an event happens (class 1) or not (class 0).

- **Logistic Function:** It employs the logistic function, also known as the sigmoid function, to model the probability of the default class.

Mathematical Formulation:

The logistic regression model operates by estimating the probability that an instance $\mathbf{x}$ belongs to a particular class. The model is represented by the logistic function, which is defined as follows (GOODFELLOW *et al.*, 2016):

1. **Logistic Function:**

The logistic function $\sigma(z)$ is given by:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is the linear combination of the input features, i.e., $z = \mathbf{w}^T\mathbf{X} + b$.

2. **Probability Estimation:**

The probability that the output y_i is 1 given the input $\mathbf{x}_i$ is:

$$P(y_i = 1|\mathbf{x}_i) = \sigma(\mathbf{w}^T\mathbf{x}_i + b)$$

and the probability that y_i is 0 is:

$$P(y_i = 0|\mathbf{x}_i) = 1 - \sigma(\mathbf{w}^T\mathbf{x}_i + b)$$

3. **Loss Function:**

The loss function used in logistic regression is the log-loss (also known as logistic loss or binary cross-entropy) (MURPHY, 2012):

$$\mathcal{L}(\mathbf{w}, b) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\sigma(\mathbf{w}^T\mathbf{x}_i + b)) + (1 - y_i)\log(1 - \sigma(\mathbf{w}^T\mathbf{x}_i + b))\right]$$

where N is the number of training examples.

4. **Gradient Descent:**

The weights $\mathbf{w}$ and bias b are optimized using gradient descent. The gradients of the loss function with respect to $\mathbf{w}$ and b are:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{1}{N}\sum_{i=1}^{N}(\sigma(\mathbf{w}^T\mathbf{x}_i + b) - y_i)\mathbf{x}_i$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{N}\sum_{i=1}^{N}(\sigma(\mathbf{w}^T\mathbf{x}_i + b) - y_i)$$

The weights and bias are updated iteratively as follows (GOODFELLOW *et al.*, 2016):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\frac{\partial \mathcal{L}}{\partial \mathbf{w}}$$

$$b \leftarrow b - \eta\frac{\partial \mathcal{L}}{\partial b}$$

where η is the learning rate.

Logistic regression is particularly valuable in fields such as medical diagnosis, marketing, and social sciences, where understanding the likelihood of an event occurring is crucial.

2.6.1.5 Voting Classifier

The Voting Classifier is an ensemble learning method that combines the predictions from multiple machine learning models to improve overall prediction accuracy. It leverages the strengths of different algorithms by combining their outputs. This approach can be implemented using either hard voting or soft voting (PEDREGOSA *et al.*, 2023b).

Hard Voting: In hard voting, each classifier in the ensemble votes for a class, and the class with the majority of votes is chosen as the final prediction. This method relies on the principle of majority rule.

Soft Voting: In soft voting, each classifier outputs a probability (or confidence) for each class. The final prediction is made based on the weighted average of these probabilities. Soft voting often results in better performance compared to hard voting, especially when the classifiers are well-calibrated.

Advantages:

- **Improved Accuracy:** By combining multiple models, the Voting Classifier can achieve higher accuracy than any individual model.

- **Reduced Overfitting:** Ensemble methods like the Voting Classifier help reduce the risk of overfitting, especially when the individual models are prone to overfitting.

- **Robustness:** The Voting Classifier can be more robust to errors or biases in the individual models.

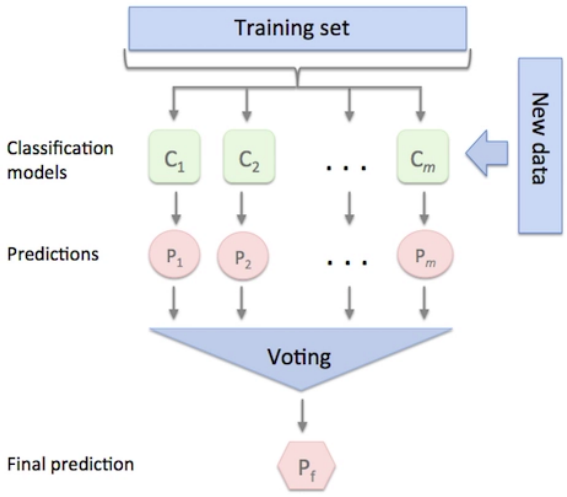

FIGURE 2.12 – Illustration of a Voting Classifier combining multiple models. (RASCHKA, 2019).

Mathematical Formulation:

1. **Hard Voting:**

For a set of classifiers $C_1, C_2, \ldots, C_m$, the final prediction $\hat{y}$ for an instance x is given

by:

$$\hat{y} = \text{mode}\{C_1(x), C_2(x), \ldots, C_m(x)\}$$

2. **Soft Voting:**

For soft voting, the probability P_k of class k for an instance x is the average of the predicted probabilities from all classifiers:

$$P_k = \frac{1}{m} \sum_{j=1}^{m} P_j(k|x)$$

The final prediction $\hat{y}$ is the class with the highest average probability:

$$\hat{y} = \arg\max_k P_k$$

2.6.2 Regression Models

Regression models are used to predict continuous outcomes, making them essential in scenarios where the goal is to forecast a quantity rather than a category. These models find patterns in data by fitting a function that maps input features to a continuous output. In the context of this thesis, regression models were applied to predict aircraft positions, which involves understanding the dynamics and patterns in the flight data (HASTIE *et al.*, 2009).

Regression models can vary significantly in their approach and complexity. Some of the common types include

1. **Linear Regression:** A simple and widely used approach that assumes a linear relationship between the input features and the output.
2. **Polynomial Regression:** An extension of linear regression that models the relationship as an nth degree polynomial.
3. **Ridge and Lasso Regression:** Techniques that add regularization terms to the linear regression to prevent overfitting by penalizing large coefficients.
4. **Decision Tree Regression:** A non-linear method that splits the data into different segments based on certain conditions, fitting a simple model (like a constant) within each segment.

In this thesis, the following classification models were employed:

2.6.2.1 XGBoost Regressor

XGBoost is a powerful gradient boosting algorithm widely used for various regression tasks. It enhances performance by combining the predictions of multiple weak learners, typically decision trees. XGBoost is known for its high efficiency, accuracy, and ability to handle large datasets with complex patterns (CHEN; GUESTRIN, 2016a).

Structure of XGBoost Regressor:

- **Boosting:** XGBoost uses the boosting technique, where models are trained sequentially to correct the errors of previous models. Each new model minimizes the residual errors of the combined model.

- **Regularization:** XGBoost includes regularization terms in its objective function to prevent overfitting and improve model generalization.

Mathematical Formulation:

The operation of XGBoost Regressor can be described using the following steps. Let $\mathcal{D}$ be the training dataset with n examples and m features.

1. **Objective Function:**

The objective function in XGBoost consists of the loss function and a regularization term (CHEN; GUESTRIN, 2016a):

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

where l is the loss function, y_i is the true label, $\hat{y}_i$ is the prediction, $\Omega(f_k)$ is the regularization term, and K is the number of trees.

2. **Loss Function:**

For regression tasks, a common loss function is the mean squared error (MSE):

$$l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$$

3. **Regularization Term:**

The regularization term in XGBoost is given by:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where T is the number of leaves in the tree, w_j is the weight of leaf j, γ and λ are

regularization parameters.

4. **Additive Training:**

XGBoost adds new models to minimize the loss function. The prediction at iteration t is updated as:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

where f_t is the new tree added at iteration t.

5. **Gradient and Hessian:**

To optimize the objective function, XGBoost uses the gradient and hessian:

$$g_i = \frac{\partial l(y_i, \hat{y}_i)}{\partial \hat{y}_i}$$

$$h_i = \frac{\partial^2 l(y_i, \hat{y}_i)}{\partial \hat{y}_i^2}$$

where g_i and h_i are the gradient and hessian for example i, respectively (CHEN; GUESTRIN, 2016a).

2.6.2.2 Lasso Regression

Lasso regression (Least Absolute Shrinkage and Selection Operator) is a linear regression model that includes a regularization term to improve prediction accuracy and interpretability by shrinking some coefficients to zero. This regularization technique helps in feature selection by effectively reducing the number of features used in the model (TIBSHIRANI, 1996).

Structure of Lasso Regression:

- **Regularization:** Lasso regression adds an L_1 penalty to the loss function, which encourages sparsity in the model coefficients. This means that some coefficients can become exactly zero, effectively selecting a subset of the features.

- **Feature Selection:** By shrinking some coefficients to zero, Lasso regression performs feature selection, making the model more interpretable and potentially improving prediction accuracy on new data.

Mathematical Formulation:

The operation of Lasso regression can be described using the following steps. Let $\mathcal{D}$ be the training dataset with n examples and m features.

1. **Objective Function:**

The objective function in Lasso regression includes a regularization term to control the complexity of the model:

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i - b)^2 + \lambda\sum_{j=1}^{m}|w_j|$$

where N is the number of training examples, y_i is the true label, $\mathbf{x}_i$ is the input feature vector, $\mathbf{w}$ is the weight vector, b is the bias term, and λ is the regularization parameter.

2. **Regularization Term:**

The regularization term in Lasso regression is given by:

$$\lambda\sum_{j=1}^{m}|w_j|$$

where λ controls the amount of shrinkage: larger values of λ result in more coefficients being set to zero.

3. **Feature Selection:**

The L_1 penalty causes some of the coefficients to be exactly zero, thus performing automatic feature selection and simplifying the model.

2.6.2.3 Gradient Boosting Regressor

Gradient Boosting is an ensemble technique that builds models sequentially, each new model correcting errors made by the previous ones. It is highly effective for regression tasks due to its ability to improve prediction accuracy by combining multiple weak models to form a strong model (FRIEDMAN, 2001).

Structure of Gradient Boosting:

- **Boosting:** Gradient Boosting trains models sequentially, where each new model aims to correct the prediction errors of the previous models.

- **Loss Function:** The technique optimizes a specified loss function by adding models that minimize this loss.

- **Learning Rate:** Gradient Boosting uses a learning rate to control the contribution of each model to the final prediction, balancing between underfitting and overfitting.

Mathematical Formulation:

The operation of Gradient Boosting Regressor can be described using the following

steps. Let $\mathcal{D}$ be the training dataset with n examples and m features.

1. **Objective Function:**

The objective in Gradient Boosting is to minimize a differentiable loss function $L(y, F(x))$ where y is the true value and $F(x)$ is the model's prediction (FRIEDMAN, 2001):

$$\mathcal{L} = \sum_{i=1}^{n} L(y_i, F(x_i))$$

2. **Initialization:**

The model is initialized with a constant value, typically the mean of the target values (HASTIE *et al.*, 2009):

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

3. **Additive Training:**

New models are added to correct the residuals of the current model. At each stage t, a new model $h_t(x)$ is trained to predict the residuals $r_i^{(t)}$:

$$r_i^{(t)} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{t-1}(x)}$$

The updated model is:

$$F_t(x) = F_{t-1}(x) + \eta h_t(x)$$

where η is the learning rate (FRIEDMAN, 2001).

4. **Learning Rate:**

The learning rate η controls the contribution of each model, with a smaller η generally leading to better performance but requiring more iterations (HASTIE *et al.*, 2009):

$$F_t(x) = F_{t-1}(x) + \eta \sum_{j=1}^{T} \gamma_j h_j(x)$$

2.6.2.4 ElasticNet

ElasticNet is a regularized regression method that linearly combines both the Lasso and Ridge regression penalties. It is particularly useful when there are multiple features

correlated with the target variable, as it can select groups of correlated features effectively (ZOU; HASTIE, 2005).

Structure of ElasticNet:

- **Combination of Penalties:** ElasticNet combines the L_1 penalty of Lasso and the L_2 penalty of Ridge regression, allowing it to handle scenarios where Lasso alone might fail, especially when the number of predictors exceeds the number of observations or when there are highly correlated predictors.

- **Feature Selection and Shrinkage:** The method performs feature selection and shrinkage, which can improve model interpretability and prevent overfitting.

Mathematical Formulation:

The operation of ElasticNet can be described using the following steps. Let $\mathcal{D}$ be the training dataset with n examples and m features.

1. **Objective Function:**

The objective function in ElasticNet combines both L_1 and L_2 penalties (HASTIE *et al.*, 2009):

$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i - b)^2 + \lambda_1 \sum_{j=1}^{m}|w_j| + \lambda_2 \sum_{j=1}^{m} w_j^2$$

2. **Regularization Terms:**

The L_1 and L_2 penalties in ElasticNet are given by:

$$\lambda_1 \sum_{j=1}^{m}|w_j| + \lambda_2 \sum_{j=1}^{m} w_j^2$$

These terms control the amount of shrinkage applied to the coefficients, with larger values of λ_1 and λ_2 leading to more shrinkage (ZOU; HASTIE, 2005).

2.6.3 Neural Networks

Neural networks are computational models inspired by the human brain. They are particularly powerful for capturing complex patterns in data, making them suitable for both classification and regression tasks. Neural networks consist of layers of interconnected nodes (neurons) that process input data and learn to make predictions through training (GOODFELLOW *et al.*, 2016).

General Structure of Neural Networks:

- **Input Layer:** This layer receives the input data. The number of nodes in this layer corresponds to the number of input features.

- **Hidden Layers:** These layers perform nonlinear transformations of the input data through weighted connections and activation functions. The number of hidden layers and nodes per layer can be adjusted based on the complexity of the task.

- **Output Layer:** This layer produces the final output. In classification tasks, the number of nodes corresponds to the number of target classes, while in regression tasks, there is typically one node for the continuous output.

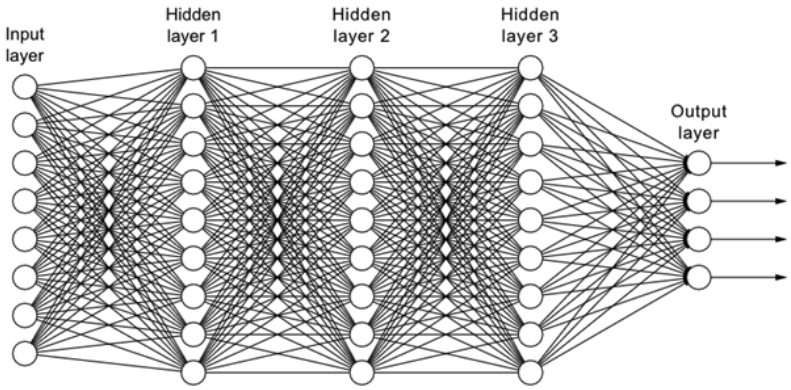

FIGURE 2.13 – Structure of a neural network with input, hidden, and output layers (SPICEWORKS, 2021).

While Multilayer Perceptrons (MLP) are a specific type of neural network characterized by fully connected layers, neural networks encompass a broader range of architectures, including but not limited to convolutional neural networks (CNNs) for image processing and recurrent neural networks (RNNs) for sequence data (GOODFELLOW *et al.*, 2016).

Mathematical Formulation:

The operation of neural networks can be described using the following steps. Let $\mathbf{x}$ be the input vector, $\mathbf{W}^{(l)}$ be the weight matrix for layer l, $\mathbf{b}^{(l)}$ be the bias vector for layer l, and f be the activation function.

1. **Forward Propagation:**

For each layer l from 1 to L (the number of layers), the output $\mathbf{h}^{(l)}$ is computed as:

$$\mathbf{h}^{(l)} = f(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$$

where $\mathbf{h}^{(0)} = \mathbf{x}$ (the input vector).

2. **Activation Functions:**

Common activation functions include the sigmoid function, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU):

$$\text{Sigmoid: } f(z) = \frac{1}{1 + e^{-z}}$$

$$\text{Tanh: } f(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$\text{ReLU: } f(z) = \max(0, z)$$

3. **Loss Function:**

The loss function measures the discrepancy between the predicted output and the actual output. For classification tasks, the cross-entropy loss is commonly used:

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k} \log(\hat{y}_{i,k})$$

For regression tasks, the mean squared error (MSE) is often used:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

4. **Backpropagation:**

Backpropagation is the process of updating the weights to minimize the loss function. It involves computing the gradient of the loss function with respect to each weight by the chain rule and then updating the weights using gradient descent (RUMELHART *et al.*, 1986):

$$\mathbf{W}^{(l)} \leftarrow \mathbf{W}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l)}}$$

where η is the learning rate.

2.6.4 Hyperparameter Tuning

Hyperparameter tuning was performed to optimize model performance. This involves adjusting parameters such as learning rate, number of trees in ensemble methods, and regularization terms.

Hyperparameters:

Hyperparameters are parameters whose values are set before the learning process be-

gins, in contrast to model parameters, which are learned during training. Hyperparameters control the training process and affect the model's performance. Common hyperparameters include the learning rate, the number of hidden layers in a neural network, the number of trees in an ensemble method, and regularization terms (GOODFELLOW *et al.*, 2016).

Importance of Hyperparameter Tuning:

Effective hyperparameter tuning can significantly enhance the performance of a machine learning model by improving accuracy, reducing overfitting, and increasing generalizability to new data. It ensures that the model is well-calibrated to handle the specific characteristics of the dataset. Without proper tuning, models may perform suboptimally, leading to poor predictions and insights.

Methods for Hyperparameter Tuning:

Several methods are commonly used for hyperparameter tuning:

1. **Manual Search:** - Manually selecting hyperparameters based on intuition, experience, or trial and error. While straightforward, this method can be time-consuming and less effective.

2. **Grid Search:** - Systematically explores a predefined set of hyperparameters by training and evaluating the model for each combination. Although exhaustive, it can be computationally expensive, especially with a large number of hyperparameters. Grid Search was utilized in this thesis to tune hyperparameters, involving the definition of a grid of hyperparameters and evaluating model performance using cross-validation for each combination.

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

# Initialize the model
model = RandomForestClassifier()

# Initialize Grid Search
grid_search = GridSearchCV(estimator=model,
                           param_grid=param_grid,
                           cv=5, scoring='accuracy')
```

```
18
19 # Fit the Grid Search to the data
20 grid_search.fit(X_train, y_train)
21
22 # Best hyperparameters
23 best_params = grid_search.best_params_
24 print(f"Best hyperparameters: {best_params}")
25
```

Listing 2.3 – Example of Grid Search for Hyperparameter Tuning

3. **Random Search:** - Random Search selects hyperparameter combinations at random from a specified distribution. It often finds good solutions more efficiently than Grid Search, especially when the search space is large.

4. **Bayesian Optimization:** - Bayesian Optimization builds a probabilistic model of the objective function and uses it to select the most promising hyperparameters to evaluate. This method can be more efficient than Grid and Random Search but is also more complex to implement (SNOEK *et al.*, 2012).

Hyperparameter tuning is crucial in achieving optimal model performance, ensuring that the model can generalize well to new, unseen data. In practice, methods like Grid Search and Random Search help systematically explore the hyperparameter space to find the best combination for a given task.

2.6.5 Cross-Validation

Cross-validation is a technique used to assess the performance and robustness of a machine learning model. It involves splitting the dataset into multiple parts, training the model on some parts while validating it on others, and repeating this process several times. The results are then averaged to provide a more accurate estimate of the model's performance. This approach ensures that the model generalizes well to unseen data and helps prevent overfitting (WIJAYA, 2023)

Importance of Cross-Validation:

Cross-validation is a crucial technique in machine learning for several reasons (BISHOP, 2006).:

1. **Prevents Overfitting:** - Overfitting occurs when a model learns the noise in the training data rather than the actual underlying pattern. This leads to poor generalization on new, unseen data. Cross-validation helps to mitigate this by ensuring that the model is evaluated on different subsets of the data, preventing it from memorizing the training data.

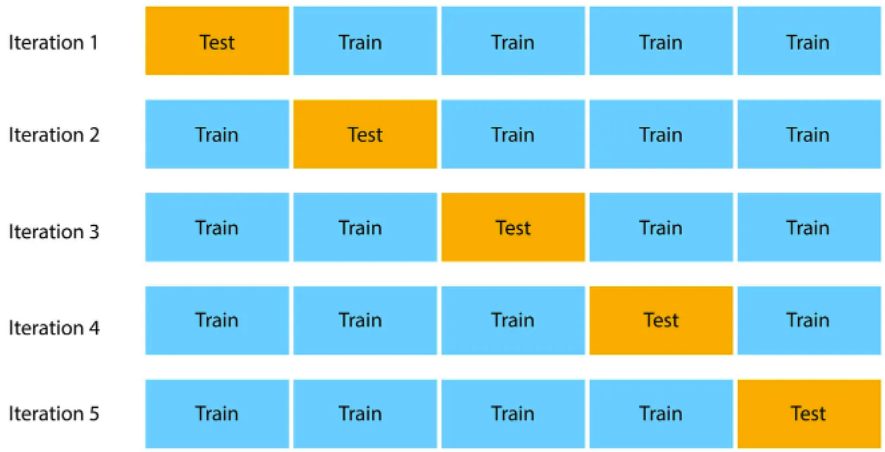

FIGURE 2.14 – Illustration of the Cross-Validation process (WIJAYA, 2023).

2. **Provides Reliable Performance Estimation:** - By using multiple folds for training and validation, cross-validation provides a more accurate and robust estimate of the model's performance. This is especially important in cases where the dataset is not large enough to be split into separate training and validation sets without losing valuable information.

3. **Efficient Use of Data:** - Cross-validation makes efficient use of the available data by ensuring that each data point is used for both training and validation. This maximizes the utility of the dataset and provides a comprehensive evaluation of the model.

2.6.6 Evaluating Models

Evaluating the performance of machine learning models is crucial to understand their effectiveness and reliability. Different metrics are used depending on whether the task is classification or regression.

2.6.6.1 Evaluating Classification Models

For classification models, the following metrics were used:

- **Precision:** Precision is the ratio of true positive predictions to the total number of positive predictions. It indicates the accuracy of the positive predictions. In other words, it measures how many of the predicted positive cases are actually positive.

$$\text{Precision} = \frac{TP}{TP + FP}$$

where: - TP (True Positives) are instances correctly classified as positive. - FP (False

Positives) are instances incorrectly classified as positive.

- **Recall:** Recall is the ratio of true positive predictions to the total number of actual positives. It measures the model's ability to identify all positive instances.

$$\text{Recall} = \frac{TP}{TP + FN}$$

where: - FN (False Negatives) are instances that are positive but incorrectly classified as negative.

- **F1-Score:** The F1-Score is the harmonic mean of precision and recall, providing a single metric that balances both concerns.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Support:** The support is the number of actual occurrences of each class in the dataset. It helps in understanding the distribution of the target variable.

- **Accuracy:** Accuracy is the ratio of correctly predicted instances to the total instances.

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

where: - TN (True Negatives) are instances correctly classified as negative (PEDREGOSA *et al.*, 2023a).

2.6.6.2 Evaluating Regression Models

For regression models, the performance was evaluated using the Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

where y_i is the true value and $\hat{y}_i$ is the predicted value. RMSE provides a measure of the differences between predicted and observed values, penalizing larger errors more significantly. RMSE is particularly useful because it is in the same units as the target variable, making interpretation straightforward (AGRAWAL, 2024).

In general, regression models can also be evaluated using several other metrics:

- **Mean Absolute Error (MAE):** This metric calculates the average magnitude of the errors in a set of predictions, without considering their direction. It is the average over the

test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

- **Mean Squared Error (MSE):** This metric calculates the average of the squares of the errors. It is more sensitive to outliers than MAE because it squares the error before averaging.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

- **R-squared ($\mathbf{R^2}$):** This metric indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It provides an indication of goodness of fit and can be used to compare the predictive power of different models.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

These metrics provide a comprehensive evaluation of model performance, enabling the selection of the most appropriate model for the given data (HIREGOUDAR, 2020).

2.6.6.3 Evaluating Final Aircraft Counting in Sectors

For predicting the counts of aircraft in sectors, we employed metrics typically used for regression models. This is because the count data can be organized in a tabular format, allowing us to analyze the prediction errors in a manner similar to regression problems, where each predicted count is compared against its actual value.

- **Mean Absolute Error (MAE):**

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

MAE measures the average magnitude of errors in a set of predictions, without considering their direction.

- **Pearson Correlation:**

$$r = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2 \sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

Pearson Correlation measures the linear correlation between predicted and actual counts, ranging from -1 to 1. A value closer to 1 indicates a strong positive correlation, while a value closer to -1 indicates a strong negative correlation (WILLMOTT; MATSUURA, 2005).

3 Methodology

3.1 Environment

In this research, the computational experiments were conducted using Google Colaboratory (Colab), a cloud-based platform that provides a free environment for running Python code. Colab is particularly advantageous for its integration with Google Drive, allowing seamless access to and storage of files.

To access Google Drive within Colab, the following code snippet was utilized:

```
from google.colab import drive
drive.mount('/content/drive')
```

Listing 3.1 – Mounting Google Drive in Colab

This command mounts the user's Google Drive to the Colab environment, facilitating the management of datasets and other resources directly from the cloud. Each Colab session operates as an independent Jupyter Notebook, an interactive computational environment that combines code execution, rich text, mathematics, plots, and media.

One significant aspect of using Colab is the need to install necessary libraries at the beginning of each new session. Since Colab does not retain the installed libraries between sessions, the `!pip install` command is employed to reinstall any required Python packages. For example, to install the `metar` library, the following command is used:

```
!pip install metar
```

Listing 3.2 – Installing the metar library

Despite this minor inconvenience, running computations in the cloud offers substantial benefits, such as leveraging Google's powerful hardware, including GPUs and TPUs, which significantly enhance computational efficiency and performance as discussed in the corresponding documentation (Google, 2024).

The following libraries were imported and utilized in this research, categorized by their functionality:

- **Standard Libraries**: `os`, `re`, `json`, `math`, `warnings`, `datetime`, `requests`, `urllib`, `concurrent.futures`, `tqdm`
- **Data Manipulation**: `numpy`, `pandas`
- **Visualization**: `matplotlib.pyplot`, `seaborn`, `mpl_toolkits.basemap`, `geopandas`, `shapely.geometry`, `plotly.graph_objects`, `plotly.express`, `plotly.subplots`, `plotly.figure_factory`
- **Machine Learning**: `sklearn.preprocessing`, `sklearn.metrics`, `sklearn.model_selection`, `sklearn.linear_model`, `sklearn.ensemble`, `sklearn.svm`, `sklearn.neural_network`, `imblearn.over_sampling`, `joblib`, `xgboost`
- **Deep Learning**: `tensorflow`, `tensorflow.keras.models`, `tensorflow.keras.layers`, `tensorflow.keras.callbacks`
- **Geospatial Libraries**: `geopandas`, `mpl_toolkits.basemap`, `shapely.geometry`
- **Miscellaneous**: `PIL.Image`, `metar.Metar`, `zipfile`, `io`

These libraries facilitated various aspects of data processing, analysis, visualization, and modeling, ensuring that the research was comprehensive and utilized state-of-the-art techniques.

3.2 Data Fetching

This section details the process used to fetch and store data from the ICEA API, as discussed in previous sections. The data was retrieved using a custom Python script that efficiently managed the retrieval process while considering the constraints of the Google Colaboratory environment.

The endpoints utilized in this study were: ”cat-62”, ”bimtra”, ”tc-real”, ”tc-prev”, ”metaf”, ”metar”, ”esperas”, and ”satelite”. These endpoints provided a range of datasets critical to the analysis. Additionally, an authentication token provided by ICEA was used to access the data.

Given the significant volume of data and the limitations of Google Colab, particularly concerning runtime and RAM, the data fetching process was designed to be both efficient and effective. The entire dataset was divided into four parts to prevent exceeding these limitations. This segmentation was particularly important for handling radar data, which is notably large.

The data retrieval script initiated by setting up a session with the API and defining the necessary parameters, such as the date range and endpoint-specific settings. For each endpoint, the script processed a range of dates, fetching data in parallel using a `ThreadPoolExecutor` to speed up the process. After fetching, the data was stored in CSV files within Google Drive, leveraging its ample storage capacity and avoiding the need to use local storage.

Each endpoint's data was divided into four parts, with each part processed and saved separately to ensure manageability. Except for CAT-62 data due to its size, it was possible to concatenate the files into one. Below is a summary of the saved files, including their sizes:

File Name	**Size**
cat-62_part_1.csv	623.9 MB
cat-62_part_2.csv	881.3 MB
cat-62_part_3.csv	941.1 MB
cat-62_part_4.csv	617.6 MB
bimtra.csv	40.6 MB
esperas.csv	4 MB
metaf.csv	6.6 MB
metar.csv	15.6 MB
satelite.csv	2.3 MB
tc-prev.csv	3.6 MB
tc-real.csv	1.5 MB

TABLE 3.1 – Summary of Saved Data Files

The following steps outline the data fetching and processing workflow:

1. **Initial Setup:** The script set up necessary configurations, including API endpoint lists, base URL, authentication token, and headers.

2. **Session Management:** A session was initiated to maintain persistent connections with the API.

3. **Data Fetching:** For each endpoint, data was fetched for a specified date range, divided into four parts to manage resource constraints.

4. **Parallel Processing**: Data retrieval was parallelized using a `ThreadPoolExecutor`, optimizing the fetching time.

5. **Data Storage**: Retrieved data was converted to Pandas DataFrames and saved as CSV files in Google Drive.

By utilizing Google Colab's cloud infrastructure, data storage and processing were conducted efficiently and scalably, taking full advantage of the platform's capabilities while mitigating its limitations. This approach enabled the handling of large datasets without overwhelming local resources.

3.3 Exploratory Data Analysis

The exploratory data analysis (EDA) was performed to uncover patterns and insights from the dataset, which would inform the feature engineering and modeling process. The EDA aimed to identify significant trends, relationships, and anomalies within the flight data.

3.3.1 Data Loading and Preprocessing

The analysis started with loading the primary dataset, `bimtra.csv`, which contains detailed flight information. The date columns were converted to datetime format to facilitate temporal analysis.

```
# Base directory of the data
base = '/content/drive/MyDrive/TG/Data/'

# Reading the 'bimtra.csv' file and converting date columns to datetime
bimtra = pd.read_csv(base + 'bimtra.csv')
bimtra['dt_dep'] = pd.to_datetime(bimtra['dt_dep'], unit='ms')
bimtra['dt_arr'] = pd.to_datetime(bimtra['dt_arr'], unit='ms')
```

Listing 3.3 – Loading and Preprocessing bimtra.csv

3.3.2 Airports Coordinates Enrichment

A dictionary containing coordinates for major Brazilian airports was used to map the latitude and longitude of the origin and destination airports in the dataset. This information was crucial for visualizing flight paths and understanding the geographical distribution of flights.

The coordinates were mapped to the respective airport codes in the `bimtra` DataFrame to enable geospatial visualization of flight paths and the Table 3.2 represents the information gathered.

3.3.3 Data Visualization

Interactive visualizations were created using Plotly to explore the data and identify patterns. Plotly is a powerful graphing library that enables the creation of interactive and publication-quality graphs online (Plotly, 2024).

Airport Code	Latitude	Longitude
SBSP	-23.4323	-46.4691
SBRJ	-22.9105	-43.1636
SBGR	-23.4323	-46.4691
SBBR	-15.8692	-47.9208
SBRF	-8.1264	-34.9240
SBCF	-19.6336	-43.9686
SBPA	-29.9935	-51.1711
SBKP	-23.0074	-45.0539
SBSV	-12.9134	-38.3317
SBCT	-25.5317	-49.1758
SBFL	-27.6705	-48.5503
SBGL	-22.8093	-43.2506

TABLE 3.2 – Coordinates of Major Brazilian Airports. Source: (AIRPORT...,)

3.3.3.1 Examples of Visualizations

Several visualizations were generated to explore the data, such as the distribution of flights by origin and destination airports, a histogram of flight durations, the relationship between average distance and maximum flight altitude, and the average waiting time by the hour of the day. These visualizations were instrumental in identifying key trends and relationships within the data, providing valuable insights for the subsequent modeling phase.

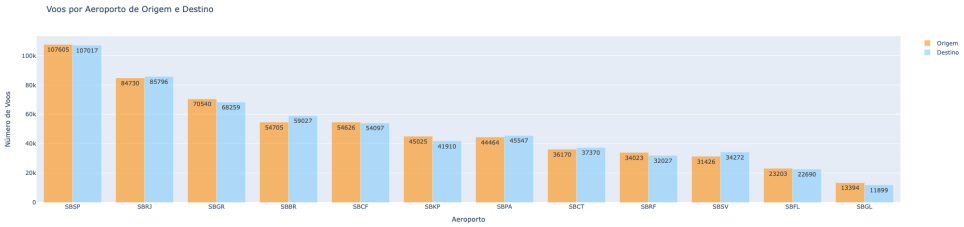

FIGURE 3.1 – Flights by Origin and Destination Airport

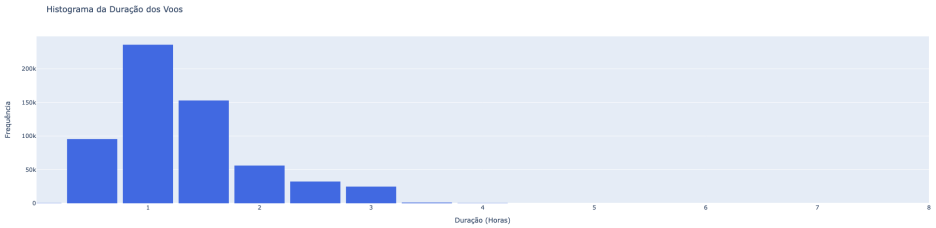

FIGURE 3.2 – Histogram of Flight Durations

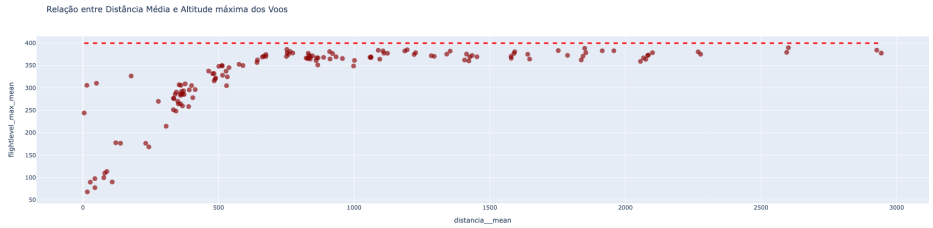

FIGURE 3.3 – Relation Between Average Distance and Maximum Flight Altitude

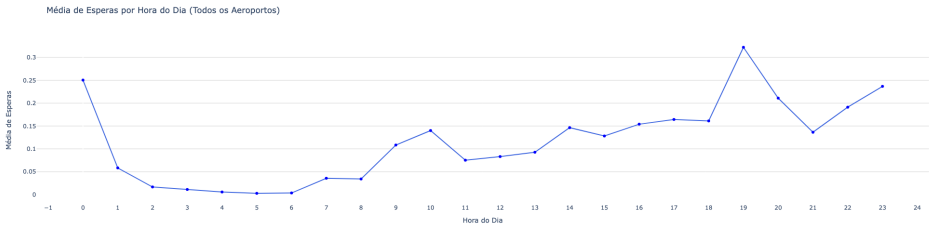

FIGURE 3.4 – Average Waiting Time by Hour of the Day

3.4 Weather Image Processing

This section outlines the methods and tools used to process weather images, download satellite data, construct polygons for analysis, and leverage Spark for efficient data processing. The aim was to extract meaningful insights from a large volume of weather-related images and integrate them with flight data for comprehensive analysis.

3.4.1 Image Download and Preprocessing

To begin with, satellite images were downloaded from specified URLs and saved locally. The function `download_image` was used to fetch images using the `urllib` library and save them with the `Pillow` library. Multithreading was implemented using the `ThreadPoolExecutor` to speed up the download process, allowing multiple images to be downloaded concurrently.

```
def download_image(data, path, destination_folder):
    try:
        with urllib.request.urlopen(path) as response:
            image = Image.open(response)
        image_save_path = f'{destination_folder}{data}.jpg'
        image.save(image_save_path)
        print(data)
    except Exception as e:
        print(f"Error loading or saving image {path.split('/')[-1]}: {e}")
```

Listing 3.4 – Downloading and Saving Satellite Images

3.4.2 Image Decoding and Polygon Construction

Once the images were downloaded, they were processed to extract relevant information using the OpenCV library. This processing involved several steps to identify and outline significant features within the images. The approach used was inspired by methods found in the repository (VILELA, 2023).

First, each image was read using OpenCV's `cv2.imread` function, which loads the image in color. The image was then cropped to remove unnecessary borders and converted to grayscale to simplify the analysis. Grayscale conversion helps in reducing computational complexity and highlights the key features needed for further processing (OpenCV, 2024).

Morphological transformations were applied to the grayscale image using a closing operation. This operation helps in closing small holes inside the foreground objects, making the features more pronounced. The transformation was achieved using a kernel matrix and the `cv2.morphologyEx` function.

To enhance the contrast and prepare the image for contour detection, the pixel values were inverted and thresholding was applied. Thresholding converts the grayscale image into a binary image, where the pixels are either black or white, based on a threshold value. This binary image makes it easier to detect contours, which are continuous lines or curves that bound or cover the full boundary of an object in an image.

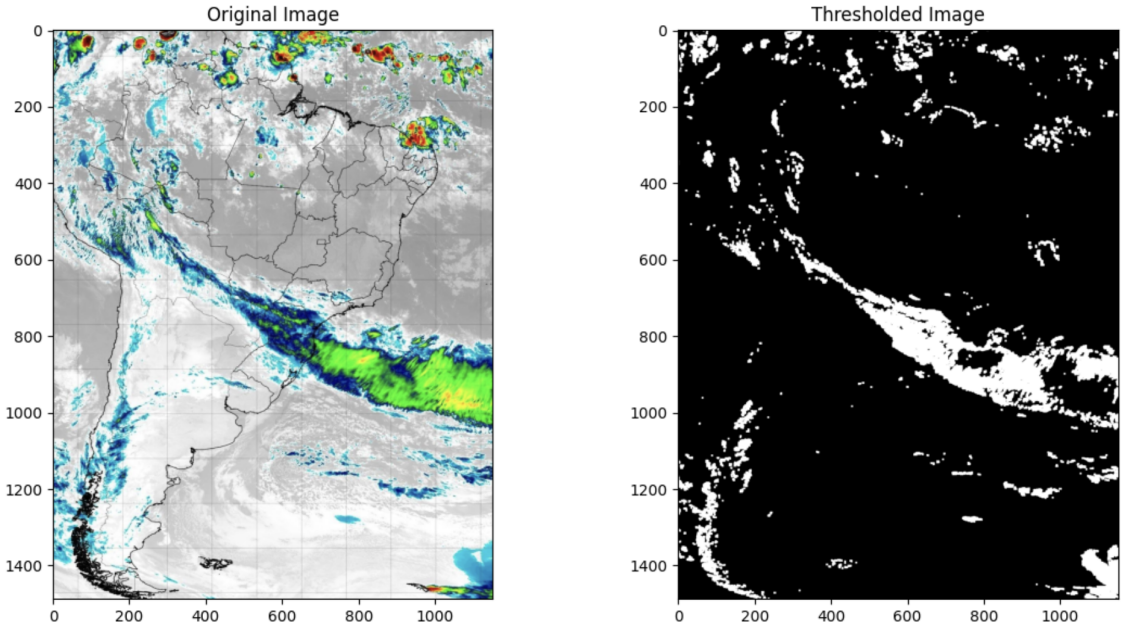

FIGURE 3.5 – Original and Thresholded Images

Contours were detected using the `cv2.findContours` function, which retrieves the contours from the binary image. These contours were then approximated to polygons using the Douglas-Peucker algorithm, implemented in OpenCV as the `cv2.approxPolyDP`

function. This step simplifies the contours to a series of vertices, which represent the polygons outlining the significant features in the image.

Here is an illustrative snippet of the code used for this process:

```
def process_image(img_path, output_path):
    # Load the image
    imageRoot = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

    # Load the image
    image = cv2.imread(img_path, cv2.IMREAD_COLOR)[740:len(imageRoot) - 100, 940:len(
    imageRoot[0]) - 100]

    kernel = np.ones((5, 5), np.uint8)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    closing = cv2.morphologyEx(gray, cv2.MORPH_CLOSE, kernel)
    closing = (closing * -1) + np.max(closing)
    closing[closing < 100] = 0
    _, thresh = cv2.threshold(closing, 50, 255, cv2.THRESH_BINARY)

    # Sort contours by their area in descending order
    thresh = np.array(thresh, np.uint8)
    contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contours = sorted(contours, key=cv2.contourArea, reverse=True)

    # Copy the original image for drawing rectangles
    image_with_rectangles = image.copy()

    # Create a blank mask to fill polygons
    mask = np.zeros_like(image)

    # Define the distance threshold to determine proximity for union
    distance_threshold = 20

    # Utility function to check proximity between two points
    def is_within_distance(point1, point2):
        return np.linalg.norm(point1 - point2) <= distance_threshold

    # Function to merge contours representing polygons
    def merge_polygons(contour1, contour2):
        merged_contour = np.vstack((contour1, contour2))
        hull = cv2.convexHull(merged_contour)
        return hull

    poly_vertices = []

    # Iterate through each contour
```

```
    for i in range(len(contours)):
        epsilon = 0.01 * cv2.arcLength(contours[i], True)  # Aumentado para reduzir a sensibilidade
        approx = cv2.approxPolyDP(contours[i], epsilon, True)

        # Union adjacent polygons based on proximity
        for j in range(i + 1, len(contours)):
            epsilon = 0.01 * cv2.arcLength(contours[j], True)  # Aumentado para reduzir a sensibilidade
            approx_j = cv2.approxPolyDP(contours[j], epsilon, True)

            # Check proximity and merge polygons
            for point in approx:
                if any(is_within_distance(point[0], p[0]) for p in approx_j):
                    approx = merge_polygons(approx, approx_j)
                    break
        poly_vertices.append(approx.tolist())
        # Draw a red rectangle around the merged polygon
        x, y, w, h = cv2.boundingRect(approx)
        cv2.rectangle(image_with_rectangles, (x, y), (x + w, y + h), (0, 0, 255), 2)
    return poly_vertices
```

Listing 3.5 – Decoding Image and Constructing Polygons

This method was inspired by techniques used in similar image processing tasks, such as those found in the Latam Challenge repository.

The following figure shows the result of the polygon detection algorithm, highlighting the identified polygons with red rectangles.

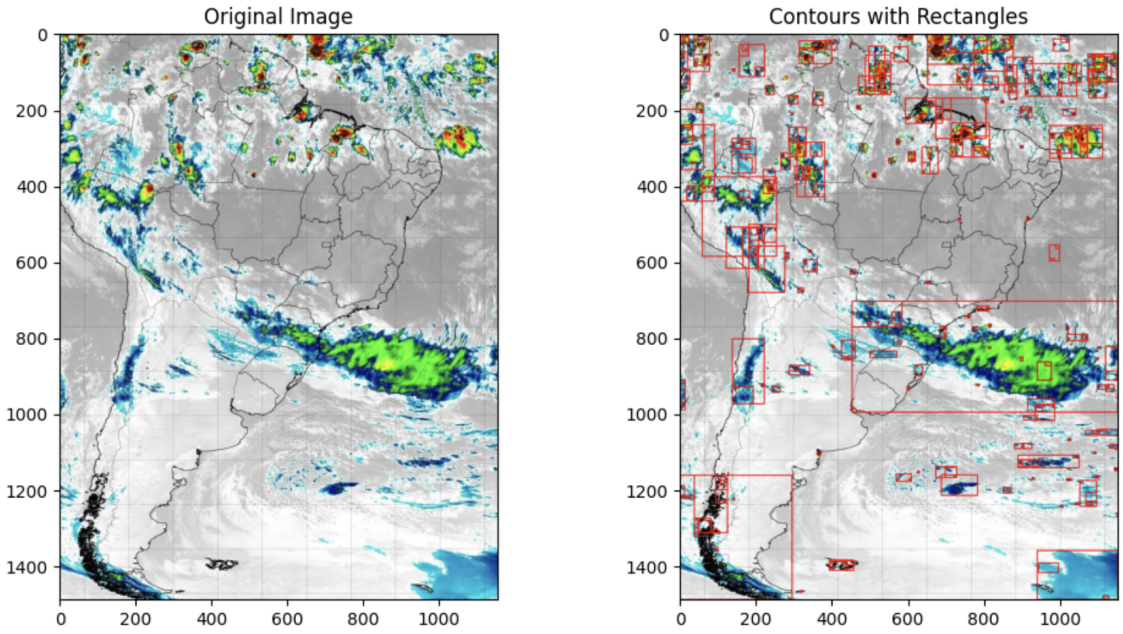

FIGURE 3.6 – Detected Polygons in Satellite Image

As shown in Figure 3.6, the algorithm successfully identifies and outlines the significant features within the image. The red rectangles denote the detected polygons, which represent the key areas of interest. The entire algorithm developed to handle this is exposed in Appendix A1 and Appendix A3.

3.4.3 Efficient Data Processing with Apache Spark

Given the large volume of data, Apache Spark was utilized to enhance processing efficiency. Apache Spark is an open-source distributed computing system that provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. It is designed to process large datasets quickly and efficiently by distributing the data and computations across many nodes in a cluster.

To set up Spark for our analysis, a Spark session was initiated. The Spark session is the entry point for programming Spark with the Dataset and DataFrame API. It allows you to create DataFrame and Dataset objects and provides an interface for executing SQL queries, data manipulation, and various machine learning operations (Apache Spark, 2024).

The following code snippet illustrates the setup of the Spark session:

```
# Setting up Spark session
spark = SparkSession.builder \
    .master('local[*]') \
    .config("spark.driver.memory", "12g") \
    .appName('my-app') \
    .getOrCreate()
spark.conf.set('spark.sql.crossJoin.enabled', 'true')
```

Listing 3.6 – Setting up Spark Session

In this configuration:

- `.master('local[*]')` specifies that Spark runs locally with as many worker threads as logical cores on the machine.

- `.config("spark.driver.memory", "12g")` allocates 12GB of memory to the Spark driver, ensuring sufficient memory for handling large datasets.

- `.appName('my-cool-app')` assigns a name to the Spark application for easier identification in the Spark UI.

- `spark.conf.set('spark.sql.crossJoin.enabled', 'true')` enables cross joins, which are necessary for certain types of SQL queries that involve combining every row of one DataFrame with every row of another.

Using Spark's distributed computing capabilities, we were able to efficiently read,

process, and analyze large volumes of data stored in CSV files. Data from various sources, such as weather images and flight paths, were loaded into Spark DataFrames, enabling complex transformations and SQL queries to be executed efficiently.

3.4.4 Polygon Intersection and SQL Queries

To determine the intersection of flight paths with weather polygons, the coordinates of the polygons were transformed into latitude and longitude. Using Spark SQL, a query was formulated to check if flight paths intersected with the weather-affected areas. This involved calculating distances and performing spatial joins to identify overlaps.

The following code snippet illustrates the SQL query used for intersection analysis:

```
query = """
    SELECT
        f.flightid, f.dt_radar,
        MAX(CASE
            WHEN (2 * 6371 * ASIN(SQRT(POW(SIN((f.lat - a.lat) * .0174532925 / 2), 2)
                                    + COS(f.lat * .0174532925) * COS(a.lat *
    .0174532925) *
                                    POW(SIN((f.lon - a.lon) * .0174532925 / 2), 2))))
                <= (a.area_estimate / 2) THEN 1
            ELSE 0
        END) as flight_through_area
    FROM (
        SELECT *,
        SUBSTRING(CAST(dt_radar AS STRING), 1, 13) AS rounded_event_time
        FROM df_flight
    ) f
    LEFT JOIN df_areas a
    ON f.rounded_event_time = SUBSTRING(CAST(a.polygon AS STRING), 3, 13)
    GROUP BY f.flightid, f.dt_radar
"""
result_df = spark.sql(query)
result_df.show()
```

Listing 3.7 – Executing SQL Query for Intersection Analysis

This SQL query performs several critical operations to identify whether flight paths intersect with weather polygons:

1. **Distance Calculation:** The query calculates the great-circle distance between the flight path coordinates (`f.lat` and `f.lon`) and the polygon coordinates (`a.lat` and `a.lon`). This distance calculation uses the Haversine formula, which is suitable for spherical geometry. The formula is given by:

$$d = 2 \times 6371 \times \arcsin\left(\sqrt{\sin^2\left(\frac{f.lat - a.lat}{2}\right) + \cos(f.lat) \times \cos(a.lat) \times \sin^2\left(\frac{f.lon - a.lon}{2}\right)}\right)$$

Here, 6371 Km is the Earth's radius in kilometers (DISTANCES..., 2023).

2. **Intersection Check:** The `CASE` statement checks if the calculated distance is less than or equal to half of the polygon's estimated area (`a.area_estimate / 2`). If the condition is met, it indicates that the flight path intersects with the weather-affected area, returning `1`. Otherwise, it returns `0`.

3. **Time Matching:** The `LEFT JOIN` operation matches flight records and polygon records based on a rounded event time. The `SUBSTRING` function extracts the relevant time component from the datetime strings to ensure accurate matching.

4. **Grouping and Aggregation:** The query groups the results by `flightid` and `dt_radar`, aggregating the results to determine if any part of the flight path intersects with a weather polygon. The `MAX` function ensures that if any segment of the flight path intersects, the result will be `1`.

The method described above was inspired by techniques found in the Latam Challenge repository (VILELA, 2023).

3.4.5 Output and Integration

The output of this query is a DataFrame, `result_df`, which contains the `flightid`, `dt_radar`, and a boolean-like variable `flight_through_area`. This variable indicates whether the flight path intersects with a weather-affected area (1 for intersection, 0 for no intersection).

The resulting DataFrame is then joined with the overall flight dataset, adding a new feature that can be used in further analysis or modeling. The boolean variable `flight_through_area` provides valuable insight into the potential impact of weather conditions on flight operations.

By leveraging tools such as OpenCV, Pillow, and Apache Spark, the process of downloading, decoding, processing, and analyzing weather images was both efficient and robust. This integration of weather data with flight paths provided critical insights into how weather affects flight operations, which can be used to improve flight safety and efficiency.

3.5 Chosen Approach for the Problem

The initial problem posed by the thesis was entirely open-ended: assist ICEA in predicting air traffic within the FIR sectors. This included decisions on which data to

use and what specific predictions to make. After conducting an exploratory data analysis (EDA) and considering practical aspects of the problem, it was determined that making hourly predictions was the most sensible approach. This decision was driven by the need to balance computational power, practicality, and ICEA's demand.

To forecast the traffic, several potential targets were considered: the number of aircraft in each sector, entry and exit times within regions (as discussed in (BRITO MAYARA C. ROCHA MURCA; OLIVEIRA, 2021)), among others. Ultimately, predicting the latitude and longitude of aircraft one hour into the future was deemed the most practical choice. This decision leverages radar readings taken immediately before the reference time (t) to predict aircraft positions at time (t+1) hour. From these predicted positions, it is possible to aggregate aircraft counts within specific regions, thus providing meaningful insights into air traffic for ICEA.

After deciding on the 1-hour prediction window for latitude and longitude, two main problems arose:

1. **Analyzing Flights That Have Not Yet Taken Off:** How to analyze flights that have not yet taken off at time t, given that the goal is to predict their positions for t+1 hour? For these flights, there will be no radar data available immediately before time t, and we do not have initial position data to predict their future positions.

2. **Handling Flights That Land Between t and t+1 Hour:** How to handle flights that land between t and t+1 hour? These flights will not have continuous radar data throughout the prediction period.

To address the first problem, it was decided to define the initial positions of flights that have not yet taken off as the latitude and longitude of their respective airports (which are available in our dataset). Other parameters, such as speed and altitude, were set to zero, which also makes physical sense for aircraft that are still on the ground.

The second problem will be addressed as detailed in Section 3.7, where specific methodologies for handling such cases will be discussed.

3.6 Feature Engineering

In transforming raw data into meaningful features that enhance the predictive capabilities of machine learning models, the methodology adopted for this project focuses on aggregating historical data based on specific dimensions such as route (origin to destination), time of day, and day of the week. This approach leverages the inherent correlations within the data to create robust features.

The primary strategy for feature engineering includes condensing historical data of the relevant magnitudes and grouping them based on various contextual factors. By doing so, we generate features that capture the seasonal and route-specific patterns observed in the flight data. For example:

- **Average Temperature**: The average temperature experienced on flights for a given route at a particular time of day.
- **Average Waiting Times**: The average number of waits at the destination airport on specific days of the week.
- **Average Runway Changes**: The average number of runway changes at the destination airport during particular times of the day.

The described methodology was chosen based on an initial analysis of the data, which indicated that flight behaviors are highly correlated with seasonal parameters (such as day of the week and time of day) and, most notably, with the specific routes. By incorporating these correlations into the feature set, we aim to improve the predictive power of the models.

3.6.1 Flight Data Processing

The flight data processing script is designed to load and process flight data from multiple CSV files to create a unified dataset. This dataset includes comprehensive information about flights and their routes, sourced from the Bimtra and Cat62 datasets. The key steps involved in this process are detailed below.

- **Data Loading and Concatenation**: The flight data is loaded from multiple CSV files. The Bimtra dataset, which includes metadata such as departure and arrival times, is read and date columns are converted to datetime format. The Cat62 dataset, which provides detailed flight trajectory information, is also loaded and concatenated from multiple parts into a single DataFrame.
- **Feature Creation and Duplicate Removal**: New features are derived from the raw data to enhance its utility for analysis. For example, the total flight duration is calculated by computing the difference between arrival and departure times. Additionally, a unique identifier for each flight route is created by combining the origin and destination codes. Duplicate entries are identified and removed based on the flight identifier to ensure data integrity.

- **Data Merging and Sorting**: The Cat62 data is merged with the Bimtra data using the flight identifier as the key. This combined dataset is then sorted by flight identifier and radar timestamp to maintain chronological order, which is essential for accurate temporal analysis.

- **Distance Calculation**: To understand the spatial dynamics of the flights, the Haversine formula is applied to calculate the great-circle distance between the starting and ending points of each flight. This calculation uses the latitude and longitude coordinates of the flights, converted to degrees for accuracy.

- **Aggregation and Data Restructuring**: The data is grouped by flight identifier, and various aggregated statistics are computed. This includes the mean, minimum, maximum, and standard deviation of flight level and speed, as well as the total flight duration. The aggregated data is then restructured to enhance readability and usability, involving steps such as flattening multi-level indices and renaming columns for clarity.

- **Data Saving**: The final processed and aggregated data is saved to a CSV file for subsequent analysis. This ensures that the dataset is ready for further steps in the predictive modeling pipeline.

By following these steps, the flight data is thoroughly prepared and enhanced, ready to be used for predictive modeling and detailed analysis. The integration of Bimtra and Cat62 datasets, coupled with the creation of new features and calculation of relevant statistics, provides a comprehensive view of the flight operations.

For further details on the libraries used, refer to the following:

- `pandas` for data manipulation and analysis (Pandas Development Team, 2024).

- `math` for mathematical operations such as the Haversine formula (Python Software Foundation, 2024a).

- `re` for regular expression operations (Python Software Foundation, 2024b).

3.6.2 Airport Waiting Data Processing

The script for this section processes airport waiting data and flight information to assess waiting patterns and integrate this data with flight itineraries. The script calculates waiting statistics per hour, day of the week, and airport, and merges this data with flight details.

- **Data Loading**: Airport waiting data is loaded, and timestamps are converted to datetime format. Information such as the day of the week and hour is extracted.
- **Grouping and Aggregation**: Waiting data is grouped by different combinations of airport, hour, and day of the week, and averages and sums of waiting times are calculated.
- **Flight Data Integration**: Flight data is processed to extract similar temporal information and merged with waiting data to create a detailed overview of each flight's conditions.

```
# Process bimtra data in the same way to extract time information
bimtra["diaSemana"] = bimtra["dt_dep"].dt.dayofweek
bimtra["hora"] = bimtra["dt_dep"].dt.hour
bimtra['HoraDataDest'] = bimtra['dt_arr'].dt.strftime('%Y-%m-%d %H')
bimtra['HoraData'] = bimtra['dt_dep'].dt.strftime('%Y-%m-%d %H')

# Merge 'bimtra' and 'waits' dataframes using the "HoraData"
df_merge = pd.merge(bimtra, waits, on='HoraData', how='left')
df_merge['esperas'] = df_merge['esperas'].fillna(0)  # Handle missing values
bimtra = pd.merge(bimtra, df_merge.groupby("flightid").agg({"esperas": "sum"
}), on="flightid", how="left")

```

Listing 3.8 – Integrating Flight Data with Waiting Data

- **Column Renaming and Data Saving**: Column names of aggregated DataFrames are adjusted to simplify data manipulation and interpretation. Processed DataFrames are saved to CSV files for future analyses or reporting.

External Dependencies: `pandas` (Pandas Development Team, 2024).

3.6.3 METAR Data Processing

The script in the present section reads and processes weather data from METAR records to analyze weather conditions related to specific flights. It uses custom functions to parse METAR data, extract essential information, and integrate it with flight data for detailed weather analysis.

- **Data Loading and Parsing**: METAR weather data is loaded from CSV files. A custom function, `parse_metar`, is used to parse each METAR record and extract information such as temperature, dew point, wind speed, wind direction, visibility, pressure, weather description, and sky conditions.

```
# Function to parse METAR data and convert it into useful columns
def parse_metar(metar):
    metar = metar.replace('METAF', 'METAR')
    try:
        metar_obj = Metar(metar)
        return pd.Series([
            metar_obj.temp.value() if metar_obj.temp else None,
            metar_obj.dewpt.value() if metar_obj.dewpt else None,
            metar_obj.wind_speed.value() if metar_obj.wind_speed else None,
            metar_obj.wind_dir.value() if metar_obj.wind_dir else None,
            metar_obj.vis.value() if metar_obj.vis else None,
            metar_obj.press.value() if metar_obj.press else None,
            ', '.join([str(c) for c in metar_obj.weather]) if metar_obj.
weather else None,
            ', '.join([str(c) for c in metar_obj.sky]) if metar_obj.sky else
None
        ])
    except Exception:
        return pd.Series([None] * 8)

```

Listing 3.9 – Parsing METAR Data

- **Feature Extraction and Data Integration**: The parsed data is integrated with flight data by merging on matching timestamps and locations. This allows the enrichment of flight records with detailed weather conditions at the time of departure and arrival.
- **Aggregation and Statistical Analysis**: For each flight, aggregated statistics such as mean, minimum, maximum, and standard deviation of the weather parameters are calculated. These statistics provide a comprehensive overview of the weather conditions experienced during each flight.

```
grouped_df = df_merge.groupby('flightid')

# Aggregate statistics for each flight
stats_df = grouped_df.agg({
    'Temperature': ['mean', 'min', 'max', 'std'],
    'Dew Point': ['mean', 'min', 'max', 'std'],
    'Wind Speed': ['mean', 'min', 'max', 'std'],
    'Wind Direction': ['mean', 'min', 'max', 'std'],
    'Visibility': ['mean', 'min', 'max', 'std'],
    'Pressure': ['mean', 'min', 'max', 'std']
})
stats_df.columns = [''.join(col) for col in stats_df.columns]
stats_df.reset_index(inplace=True)

```

```
15      # Update 'bimtra' with aggregated data
16      bimtra = bimtra.merge(stats_df, on="flightid", how="left")
17
```

Listing 3.10 – Aggregating Weather Data

- **Additional Data Processing**: Additional METAF data is processed similarly, ensuring that all relevant weather information is integrated into the flight dataset. The same parsing, feature extraction, and integration steps are applied to the METAF data.

External Dependencies: `pandas`, `Metar` (Pandas Development Team, 2024; Python Metar Library, 2024).

3.6.4 Runway Change Data Processing

The script developed for analysing Runway Change data processes and analyzes forecast and actual data of runway changes at aerodromes, integrating them with flight data to assess the accuracy of forecasts and understand air traffic dynamics at different airports.

- **Time Data Preprocessing**: A function converts milliseconds into date/time format and extracts relevant information such as day of the week and hour.
- **Data Loading**: Forecast (tc-prev) and actual (tc-real) data on runway changes at aerodromes are loaded.
- **Aggregation**: Data is grouped and aggregated by airport, hour, and day of the week, calculating averages and sums.
- **Data Saving**: Aggregated data is saved to CSV files for future analyses.
- **Flight Data Integration**: Flight data is prepared, airport subcodes are extracted from origin, and merged with aggregated runway change data.

External Dependencies: `pandas`

The steps for processing and integrating runway change data with flight data are detailed below:

Data Loading: Forecast (‘tc-prev‘) and actual (‘tc-real‘) runway change data are loaded from CSV files.

Aggregation: The runway change data is grouped by airport, hour, and day of the week. Aggregated statistics, such as mean and sum, are calculated for each group.

Data Saving: The aggregated runway change data is saved to CSV files for future analyses.

Flight Data Integration: The ‘bimtra‘ flight data is prepared by extracting airport subcodes from the origin column. This data is then merged with the aggregated runway change data.

```
# Load and prepare bimtra data
bimtra = pd.read_csv("/content/drive/MyDrive/TG/Dados/bimtra.csv")

# Extract airport subcode from origin
bimtra["aero"] = bimtra["origem"].str[2:]

# Merge bimtra with all aggregated datasets
def merge_with_bimtra(bimtra, df, merge_cols, how='left'):
    bimtra = bimtra.merge(df, on=merge_cols, how=how)
    return bimtra

bimtra = merge_with_bimtra(bimtra, tcprev_hora, ["aero", "hora"])
bimtra = merge_with_bimtra(bimtra, tcprev_sem, ["aero", "diaSemana"])
bimtra = merge_with_bimtra(bimtra, tcprev_aero, ["aero"])
bimtra = merge_with_bimtra(bimtra, tcreal_hora, ["aero", "hora"])
bimtra = merge_with_bimtra(bimtra, tcreal_sem, ["aero", "diaSemana"])
bimtra = merge_with_bimtra(bimtra, tcreal_aero, ["aero"])
```

Listing 3.11 – Integrating Flight Data with Runway Change Data

External Dependencies: `pandas` (Pandas Development Team, 2024).

3.6.5 Combining Features

The present section displays how all the data from different sources were integrated together.

- **Data Merging**: The `bimtra` DataFrame is merged with summary data, waiting data, and weather data to form the training dataset.
- **Categorical Encoding**: The `origin` and `destination` columns are factorized into numerical codes.

```
# Factorize 'origin' and 'destination' columns to numerical codes
train["origin"], origin = pd.factorize(train["origin"])
train["destination"], destination = pd.factorize(train["destination"])
train["linha"], linha = pd.factorize(train["linha"])
```

Listing 3.12 – Factorizing Categorical Columns

- **Data Cleaning**: Unnecessary columns are removed, and column names are cleaned by removing special characters.
- **Flight Data Integration**: The training dataset is further merged with additional flight data, such as flight paths through weather-affected areas.
- **Final Adjustments**: Data is sorted by $flightid$ and dt_radar, future coordinates are calculated, and missing values are imputed using group means.

```
# Data is sorted by flightid and dt_radar
train.sort_values(by=['flightid', 'dt_radar'], inplace=True)

# Calculate future coordinates
train['lat_futuro'] = train.groupby('flightid')['lat'].shift(-1)
train['lon_futuro'] = train.groupby('flightid')['lon'].shift(-1)

# Impute missing values using group means
cols_to_impute = [col for col in train.select_dtypes(include=[np.number]).columns
    if train[col].isnull().any()]
for col in cols_to_impute:
    train[col] = train.groupby(['linha', 'hora'])[col].transform(lambda x: x.
    fillna(x.mean()))
```

Listing 3.13 – Final Adjustments to Data

- **Data Saving**: The final training dataset is saved to a CSV file for use in predictive modeling.

External Dependencies: `pandas` (Pandas Development Team, 2024), `numpy` (NumPy Community, 2024).

3.6.6 Last Processing Steps

The final stage of feature engineering involves a series of steps designed to refine and consolidate the training dataset for predictive modeling in flight data analysis. This section outlines the comprehensive steps taken to prepare the dataset, ensuring it is ready for tasks such as air traffic prediction and flight trajectory forecasting.

Features:

1. **Data Loading and Sorting**: The process begins with loading the flight data from a CSV file and sorting it by `flightid` and `dt_radar`. Sorting ensures efficient manipulation of data in subsequent steps.

2. **Handling Multiple Occurrences**: To maintain data consistency, the script identifies flights with multiple entries and retains only the penultimate occurrence for each flight. This mitigates potential biases introduced by duplicate entries.

3. **Geographical Enrichment**: The dataset is enriched with geographical coordinates (latitude and longitude) of origin and destination airports in the same manner done in the EDA section.

4. **Temporal Adjustments**: Temporal attributes within the dataset are adjusted to ensure accurate analysis of flight timelines. This includes calculating the time differences between key flight events such as departure and arrival.

5. **Future Trajectory Prediction**: Future latitude and longitude coordinates are calculated using the `shift()` method. This allows for predictive analysis of flight trajectories, enabling the development of models that forecast future positions based on historical data.

6. **Conditional Imputation and Data Cleaning**: Missing values are filled using conditional imputation with group means, and unnecessary columns are removed. This step streamlines the dataset and improves computational efficiency.

7. **Temporal Simulation**: New rows are added to simulate flight scenarios where departures occur one hour before the initial recorded time. This augmentation enhances the dataset's completeness for temporal analyses and model training purposes.

8. **Statistical Analysis Preparation**: The dataset is prepared for machine learning or statistical analyses by calculating various statistics and applying transformations. This includes imputing missing values, removing columns with excessive missing data, and ensuring data integrity.

The final code snippet demonstrates handling multiple occurrences and sorting the data:

```
# Identify and filter multiple occurrences of flightid
multiple_occurrences = train.groupby('flightid').filter(lambda x: len(x) > 1)
one_occurrence = train.groupby('flightid').filter(lambda x: len(x) == 1)
drop_later = one_occurrence.drop_duplicates(subset='flightid', keep='last').index

# Retain only the penultimate occurrence of each flight
last_occurrences = multiple_occurrences.drop_duplicates(subset='flightid', keep='last'
    ).index
train = train[~train.index.isin(last_occurrences)]
train.sort_values(by=['flightid', 'dt_radar'], inplace=True)
```

Listing 3.14 – Handling Multiple Occurrences and Sorting

By combining features from various data sources and performing thorough preprocessing, the resulting dataset is well-prepared for predictive modeling and further analysis.

External Dependencies: `pandas` (Pandas Development Team, 2024), `numpy` (NumPy Community, 2024).

3.7 Classification Models

Before proceeding with the prediction of latitude and longitude coordinates, it is crucial to address a fundamental step: determining whether an aircraft has landed or not. This step is essential because the main goal is to focus on aircraft that are in flight, excluding those that have already landed, as they do not directly contribute to solving the problem at hand. Furthermore, by training the regression models with data exclusively from ongoing flights, it is possible to ensure greater specialization in predicting these cases, minimizing the occurrence of possible modeling errors associated with flights that have already concluded. Thus, this approach allows for directing analysis and prediction efforts towards the most relevant and significant scenarios for the context of the work.

In the Feature Engineering section, flights that have already landed were defined through the last radar reading. The aim is to predict the status of the aircraft in a one-hour interval. This is particularly important because it is common for aircraft to be in flight at the reference time but land before the prediction time. This scenario poses challenges, especially for flights with estimated landing times close to the prediction hour, as misclassifying these can negatively impact the accuracy of subsequent predictions and counts.

During the Exploratory Data Analysis (EDA), it was observed that a significant portion of flights lasts close to one hour. This is especially true for short-distance routes, such as those between Rio de Janeiro and São Paulo. Given this context, it becomes evident that accurately determining the landing status within the one-hour prediction interval is critical to ensuring the precision of the models.

3.7.1 Training Models

This section describes the general approach for training the classification models, focusing on data preprocessing, model training, and evaluation.

To prepare the data for classification, several key steps were performed:

1. **Data Splitting**: The dataset was divided into features and labels, excluding irrelevant columns such as `data_ref`, `flightid`, `lat_futuro`, and `lon_futuro`.

2. **Feature Standardization**: `StandardScaler` was used to standardize the features, ensuring that all features have the same scale. This step is crucial for models that

rely on distance calculations.

3. **Dealing with Class Imbalance**: The oversampling technique `SMOTE` (Synthetic Minority Over-sampling Technique) was applied to balance the classes by increasing the number of samples in the minority class.

For model training and evaluation, the following steps were followed:

1. **Data Splitting**: The data was split into training and testing sets using `train_test_split` from `scikit-learn`.

2. **Performance Evaluation**: The performance of each model was evaluated using metrics such as accuracy, recall, and F1-score. These metrics were obtained from the classification report generated by `scikit-learn`.

3. **Confusion Matrix Visualization**: To better understand the model performance across different classes, the confusion matrix for each model was visualized. An example of a confusion matrix for the MLP model is shown in Figure 3.7.

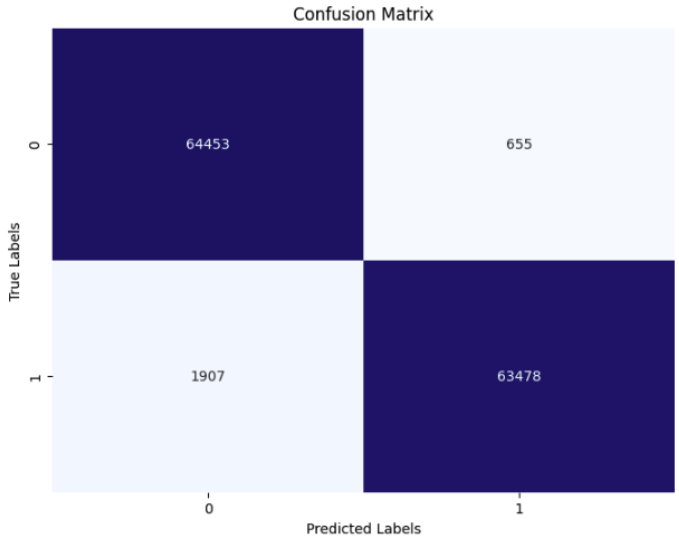

FIGURE 3.7 – Confusion Matrix for the MLP Model

Additionally, the trained models were saved to files using the `joblib` library. This allows for easy deployment of the best model into production and making predictions on new data. The scaler was also saved to ensure consistency in preprocessing future data.

Finally, saving the scaler is crucial to avoid discrepancies in feature scaling when applying the model to new data. The scaler object contains information about the mean and standard deviation of the training features. By saving and reusing the scaler, it is ensured that any future data undergoes the same scaling transformation as the training data, maintaining consistency and preventing potential prediction inaccuracies.

3.7.2 Models Trained

In this section, the classification models that were trained to predict whether an aircraft has landed or not are outlined. Each model was configured with specific parameters to optimize its performance. Below are the models and their key parameters:

- **Random Forest**:
  - **Number of estimators**: 100
  - **Random state**: 42
- **XGBoost**:
  - **Number of estimators**: 100
  - **Random state**: 42
- **Logistic Regression**:
  - **Random state**: 42
- **MLP Classifier**:
  - **Hidden layer sizes**: (150, 100, 50)
  - **Activation**: relu
  - **Solver**: adam
  - **Alpha**: 0.0001
  - **Learning rate**: adaptive
  - **Maximum iterations**: 500
  - **Random state**: 42

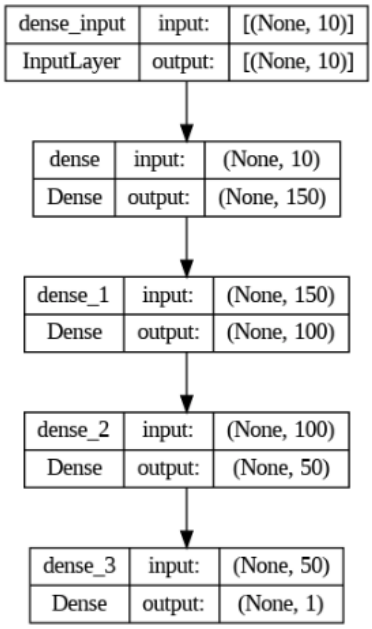

FIGURE 3.8 – Structure of the MLP Classifier used

3.7.3 Final Classification Model Selection

This section details the process of selecting the final model for predicting whether an aircraft has landed or not. Based on the performance metrics from the initial training phase, two top-performing models were identified: the MLP Classifier and the XGBoost model. To further enhance the performance of the XGBoost model, hyperparameter tuning was conducted using a grid search.

3.7.3.1 Hyperparameter Tuning for XGBoost

For the XGBoost model, a grid search was implemented to find the best combination of hyperparameters. The parameter grid used for tuning is as follows:

```
param_grid = {
    'n_estimators': [50, 100, 150, 200],
    'max_depth': [3, 5, 7, 8],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 0.9],
    'colsample_bytree': [0.8, 0.9]
}
```

Listing 3.15 – Hyperparameter Grid for XGBoost

The presented comprehensive grid allowed for a systematic exploration of a range of values for each hyperparameter, ensuring that the XGBoost model was fine-tuned for

optimal performance. Despite the longer training times associated with grid search, it was feasible within the available computational resources and provided a thorough examination of the hyperparameter space. This method proved to be effective for the problem at hand, balancing the need for precise tuning with the available computational power. The grid search approach, while inherently slower, was practical given the analysis of training times and the computational capabilities at disposal.

Other methods such as random search, Bayesian optimization, and genetic algorithms offer different approaches to hyperparameter tuning and may be useful in various scenarios. Random search randomly samples hyperparameter combinations, which can be more efficient than grid search when dealing with a large number of parameters. Bayesian optimization builds a probabilistic model of the function mapping hyperparameters to the objective and uses this model to choose hyperparameters that are expected to improve performance. Genetic algorithms use principles of natural selection to iteratively improve the set of hyperparameters. Each of these methods provides unique benefits and can be particularly useful depending on the specific constraints and requirements of the problem being addressed.

3.7.3.2 Ensemble Method - Voting Classifier

Given the strong performance of both the MLP Classifier and the tuned XGBoost model, an ensemble method was employed to leverage the strengths of both models. Specifically, a Voting Classifier was implemented, which combines the predictions from both models to make a final decision. This approach is known to improve predictive accuracy and robustness (RASCHKA, 2019).

The Voting Classifier was configured as follows:

```
mlp_model_path = base_ + 'mlp_classifier.joblib'
xgb_model_path = base_ + 'xgboost_classifier.joblib'
model_mlp = load(mlp_model_path)
model_xgb = load(xgb_model_path)

# Ensemble Method - Voting Classifier
ensemble_model = VotingClassifier(estimators=[('MLP', model_mlp), ('XGBoost',
    model_xgb)], voting='hard')
ensemble_model.fit(X_train, y_train)
```

Listing 3.16 – Voting Classifier Configuration

In the setup of Listing 3.16, the Voting Classifier uses a hard voting strategy, meaning it takes the majority vote from the individual classifiers (MLP and XGBoost) for the final prediction. This ensemble method benefits from the complementary strengths of the individual models, leading to improved overall performance.

External Dependencies: `pandas` (Pandas Development Team, 2024), `numpy` (NumPy Community, 2024), `scikit-learn` (AL., 2021b), `imblearn` (AL., 2021a), `joblib` (GRISEL; TEAM, 2021), `xgboost` (XGBOOST..., 2016).

3.8 Regression Models

Following the classification task, the focus now shifts to developing regression models to predict the latitude and longitude coordinates of aircraft. The goal is to create accurate predictions of aircraft positions one hour into the future, based on current flight data. By forecasting these coordinates, the air traffic management system's ability to anticipate and manage aircraft movements efficiently is enhanced.

For such task, separate models are generally employed for predicting latitude and longitude to ensure precision and focus on each coordinate's unique predictive patterns. However, in one of the advanced neural network models, a unified approach was experimented with, training a single model to predict both coordinates simultaneously.

The following sections detail the various regression models used, the preprocessing steps involved, and the evaluation metrics applied to assess their performance.

3.8.1 Training Models

This section describes the general approach for training the regression models, focusing on data preprocessing, model training, and evaluation.

To prepare the data for regression, several key steps were performed:

1. **Data Splitting**: The dataset was divided into features and labels for predicting latitude and longitude. Irrelevant columns such as `data_ref`, `flightid`, `lat_futuro`, and `lon_futuro` were excluded.

2. **Feature Standardization**: `StandardScaler` was used to standardize the features to ensure that all features had the same scale. This step is crucial for models that rely on distance calculations.

3. **Excluding Landed Flights**: The previously trained classification model was applied to filter out flights that had already landed, focusing solely on ongoing flights.

For model training and evaluation, the following steps were followed:

1. **Data Splitting**: The data was split into training and testing sets using `train_test_split` from `scikit-learn`.

2. **Model Training**: Separate models were trained for predicting latitude and longitude. The models were configured with specific hyperparameters to optimize performance.

3. **Performance Evaluation**: The models were evaluated using metrics such as RMSE (Root Mean Squared Error) to measure the accuracy of the predictions.

4. **Visualization of Results**: To better understand the model performance, the predicted versus actual coordinates for each test instance were plotted. An example plot is shown in Figure 3.9.

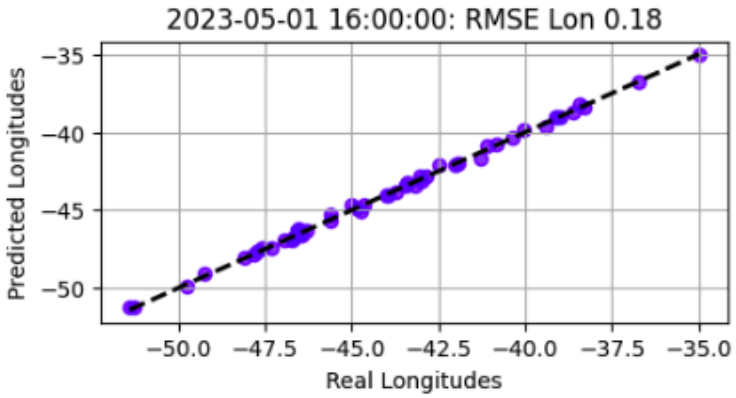

FIGURE 3.9 – Predicted vs Actual Coordinates for Latitude and Longitude

Additionally, the trained models and the scaler were saved to ensure consistency in preprocessing future data. This is crucial to avoid discrepancies in feature scaling when applying the model to new data, as the scaler object contains information about the mean and standard deviation of the training features.

By combining features from various data sources and performing thorough preprocessing, the resulting dataset is well-prepared for predictive modeling and further analysis.

3.8.2 Models Trained

In this section, the regression models that were trained to predict the latitude and longitude coordinates of aircraft are outlined. Each model was configured with specific parameters to optimize its performance. Below are the models and their key parameters:

- **XGBoost for Latitude and Longitude**:
  - **Objective**: reg:squarederror
  - **Number of estimators**: 100
  - **Learning rate**: 0.1

  - **Random state**: 42

- **Lasso Regression for Latitude and Longitude**:
  - **Alpha**: 0.1
  - **Random state**: 42

- **Gradient Boosting for Latitude and Longitude**:
  - **Random state**: 42

- **Elastic Net for Latitude and Longitude**:
  - **Alpha**: 0.1
  - **L1 ratio**: 0.7
  - **Random state**: 42

- **MLP Regressor for Latitude and Longitude**:
  - **Hidden layer sizes**: (128, 64, 32)
  - **Activation**: relu
  - **Solver**: adam
  - **Alpha**: 0.0001
  - **Learning rate**: adaptive
  - **Maximum iterations**: 1000
  - **Random state**: 42
  - **Early stopping**: True

- **Neural Network for Joint Latitude and Longitude Prediction**:
  - **Layers**:
    * Dense(256, activation='relu')
    * BatchNormalization()
    * Dropout(0.3)
    * Dense(256, activation='relu')
    * BatchNormalization()
    * Dropout(0.3)
    * Dense(128, activation='relu')
    * BatchNormalization()
    * Dropout(0.3)

    * Dense(2)
  - **Optimizer**: adam
  - **Loss function**: mean squared error (mse)
  - **Batch size**: 32
  - **Epochs**: 100

In the context of predicting aircraft coordinates, several regression models were trained, each designed to predict either latitude or longitude independently. The XGBoost models were fine-tuned to optimize performance, while the MLP Regressor models used a multi-layer perceptron architecture to capture complex patterns in the data.

Additionally, a more sophisticated approach was explored by implementing a neural network model that predicts both latitude and longitude simultaneously. This joint prediction model leverages a deep learning architecture to provide comprehensive predictions for aircraft coordinates.

By utilizing a combination of traditional machine learning algorithms and advanced neural network models, the goal is to enhance the predictive capabilities and accuracy of the air traffic management system.

External Dependencies: `pandas` (Pandas Development Team, 2024), `numpy` (NumPy Community, 2024), `scikit-learn` (AL., 2021b), `joblib` (GRISEL; TEAM, 2021), `xgboost` (XGBOOST..., 2016), `tensorflow` (ABADI *et al.*, 2015).

3.8.3 Final Regression Model Selection

The current section details the process of selecting the final models for predicting the latitude and longitude coordinates of aircraft. Based on the performance metrics from the initial training phase, the top-performing model identified was XGBoost. To further enhance the performance of the XGBoost models, hyperparameter tuning was conducted using a grid search.

3.8.3.1 Hyperparameter Tuning for XGBoost

For the XGBoost models, a grid search was implemented to find the best combination of hyperparameters. The parameter grid used for tuning is as follows:

```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 8, 10],
    'learning_rate': [0.01, 0.05, 0.1],
```

```
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9],
    'gamma': [0, 0.1, 0.2]
}
```

Listing 3.17 – Hyperparameter Grid for XGBoost

Through this process, the XGBoost models for both latitude and longitude predictions were fine-tuned, enhancing their accuracy and reliability.

3.8.4 Final Model

After thorough evaluation and testing of various models, including both individual regressors and ensemble methods, the decision was made to proceed with the XGBoost model as the final model for predicting the latitude and longitude coordinates of aircraft. The XGBoost model consistently demonstrated superior performance across key metrics, and its accuracy surpassed that of the other models tested.

While ensemble methods can often enhance predictive performance by combining the strengths of multiple models, the gains in accuracy achieved by the XGBoost model alone were significant enough to render the ensemble approach unnecessary.

By selecting the XGBoost model as the final model, predictions are ensured to be both precise and reliable, meeting the rigorous demands of air traffic prediction and providing a solid foundation for future applications and analyses.

3.9 Counting Aircraft in FIR Sectors

First, the shapefiles of the FIR sectors were downloaded to obtain the necessary geospatial base for the analyses. Next, regions that contain other regions in the downloaded file were identified and excluded to avoid duplicates.

3.9.1 Identification of Contained Regions

GeoPandas was used to load and manipulate the geospatial data. Initially, the flight data was converted to a GeoDataFrame, and the shapefile of the FIR sectors was loaded. To ensure that all FIR regions were in the same coordinate system, the CRS (Coordinate Reference System) of the FIR sectors was adjusted to match the CRS of the aviation data. FIR regions that contain other regions were then identified, and a column was created to store these subregions.

```
# Initialize a column to store the subregions of each FIR sector
sectors_fir_gdf['contained_firs'] = None
# Check for contained in relation to every other FIR sector
for index, fir in sectors_fir_gdf.iterrows():
    # Create a temporary GeoDataFrame for other sectors, excluding the current sector
    other_firs = sectors_fir_gdf.drop(index)
    # Identify sectors that are contained within the current sector
    contained_firs = other_firs[other_firs['geometry'].within(fir['geometry'])]
    # If there are contained sectors, save their identifiers
    if not contained_firs.empty:
        sectors_fir_gdf.at[index, 'contained_firs'] = contained_firs['ident'].tolist()
# Filter sectors that contain other FIR sectors
sectors_with_subregions = sectors_fir_gdf.dropna(subset=['contained_firs'])
```

Listing 3.18 – Converting Flight Data to GeoDataFrame and Loading FIR Sectors

The resulting table shows the FIR regions that contain other regions. Cases were observed where region A is contained within B, and simultaneously, B is contained within A, implying that A and B are equal. These regions were grouped and each aircraft was counted only once.

The entire algorithm developed to handle this is exposed in Appendix A2.

3.9.2 Counting Aircraft

For counting the aircraft, a dictionary mapping child sectors to their parent sectors was built, and the count was adjusted to accumulate in the parent sectors. The following code snippet demonstrates the spatial join and counting process:

```
    gdf = gpd.GeoDataFrame(
        df,
        geometry=[Point(xy) for xy in zip(df[lon_col], df[lat_col])],
        crs="EPSG:4326"
    )
    # Load the .shp file of FIR sectors
    sectors_fir_gdf = gpd.read_file(fir_shp_path)
    # Perform spatial intersection for all FIR sectors
    result = gpd.sjoin(gdf, sectors_fir_gdf, how='left', op='intersects')
    # Identify flights that are not in any FIR sector
    result['fir_sector'] = result[fir_name_col].fillna('No Correspondence')
    # Count how many flights per FIR sector, including unmapped ones
    count_per_fir = result['fir_sector'].value_counts()
```

Listing 3.19 – Counting Aircraft in FIR Sectors

To visualize the results, a map showing the predicted locations of aircraft within the FIR sectors was included, as shown in Figure 3.11.

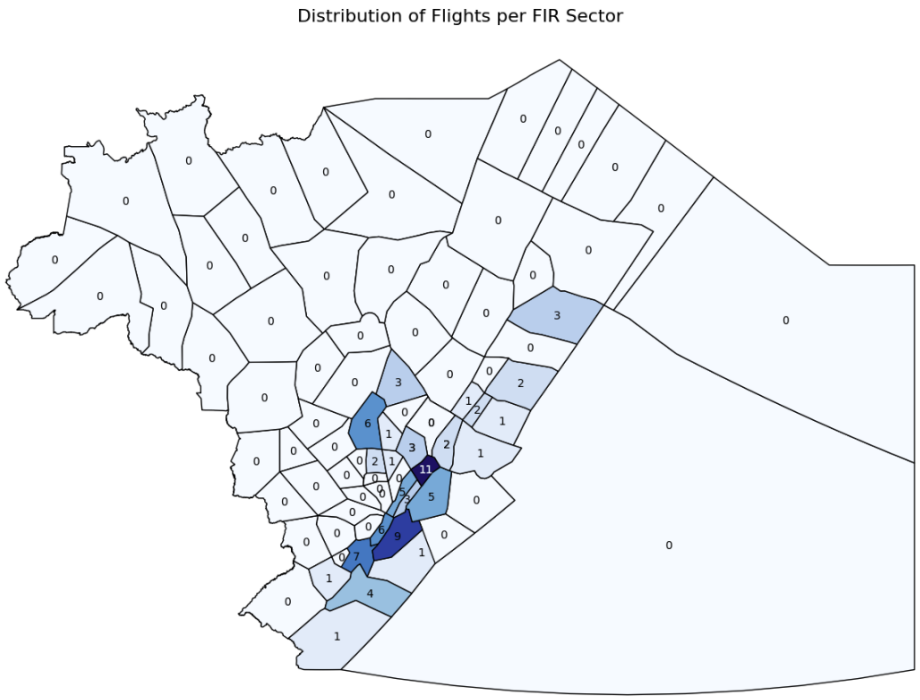

FIGURE 3.10 – Predicted Aircraft Locations within FIR Sectors

3.10 Evaluating the Final Predictions

3.10.1 Obtaining True Values

To evaluate the performance of the model, its predictions were compared with actual aircraft positions obtained from radar data. First, radar data was gathered to capture the actual positions of aircraft at the times of interest. It was assumed that if an aircraft is in flight, it will be detected by the radar. Using this radar data, a spatial join was performed to determine which FIR sector each aircraft was located in, similar to the process used for the predicted data. This allowed for counting the number of aircraft in each FIR sector based on radar readings.

The spatial join and counting process for the radar data was performed similarly to the predicted data, ensuring consistency in the comparison. By comparing the model's predicted aircraft counts with the actual counts from the radar data, the model's performance was assessed, providing valuable insights into its effectiveness.

3.10.2 Baseline Model

For meaningful evaluation, it is essential to compare the model's performance against a baseline model. The average velocity of each aircraft was calculated and used to predict future positions. This baseline model helps to understand the relative performance of the

developed model and serves as a benchmark.

3.10.3 Evaluating the Final Predictions

To evaluate the final predictions, a comparative table was created that included radar-based counts, model-predicted counts, and baseline model counts for each FIR sector. An example of such a comparative table is shown below:

Main Region	Radar	Predicted	Mean Velocity
SBBS_03	10	11.0	1.0
SBCW_05	10	9.0	2.0
SBBS_11	7	6.0	0.0
SBCW_18	6	7.0	4.0
SBCW_09	6	6.0	5.0
SBCW_04	4	4.0	5.0
SBBS_02	4	3.0	3.0
SBCW_08	4	5.0	4.0
SBRE_06	3	3.0	0.0
SBRE_13	3	3.0	1.0
SBBS_10	3	3.0	0.0

TABLE 3.3 – Comparison of Radar Data, Predicted Data, and Baseline Model Data for FIR Sectors

The challenge in evaluating the performance arises from the variability in the number of aircraft across different FIR sectors and times of the day. The total number of predictions can vary significantly, especially during off-peak hours when only a few aircraft are in flight. Given these variations, it is crucial to select appropriate metrics for evaluation that consider these factors.

To address these challenges, the following evaluation metrics were employed:

1. **Mean Absolute Error (MAE)**: MAE measures the average magnitude of errors in the predictions, providing a straightforward interpretation of the model's accuracy.

2. **Pearson Correlation Coefficient**: This metric evaluates the linear relationship between the predicted and actual counts, indicating how well the model's predictions correlate with the true values.

3. **Average Distance Between Predicted and Actual Positions**: This metric assesses the spatial accuracy of the model by calculating the average distance between the predicted and actual positions of aircraft.

These metrics collectively offer a comprehensive evaluation of the model's performance, accounting for both numerical accuracy and spatial reliability. By considering multiple

aspects of prediction quality, a robust and thorough assessment of the model's effectiveness in predicting aircraft positions within FIR sectors is ensured.

To evaluate the final performance of the model, the last 10 days of the dataset, which were not seen by the models during training, were used. The entire algorithm was run hour by hour for these days, and the results were saved for analysis. The figure below illustrates the Mean Absolute Error (MAE) results for all hours from May 1 to May 10, 2023.

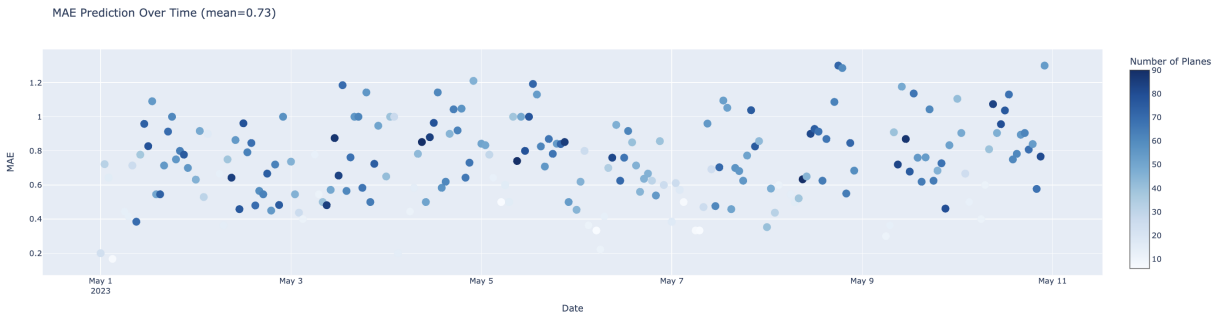

FIGURE 3.11 – Mean Absolute Error (MAE) of Aircraft Count Predictions in FIR Sectors

3.11 Production Workflow

In the final section, the steps required to transition the developed models and methodologies into a production environment are detailed. This workflow ensures consistent and accurate predictions by leveraging the previously discussed cloud infrastructure and tools. The overall process is visualized in the provided flowchart via Figure 3.12.

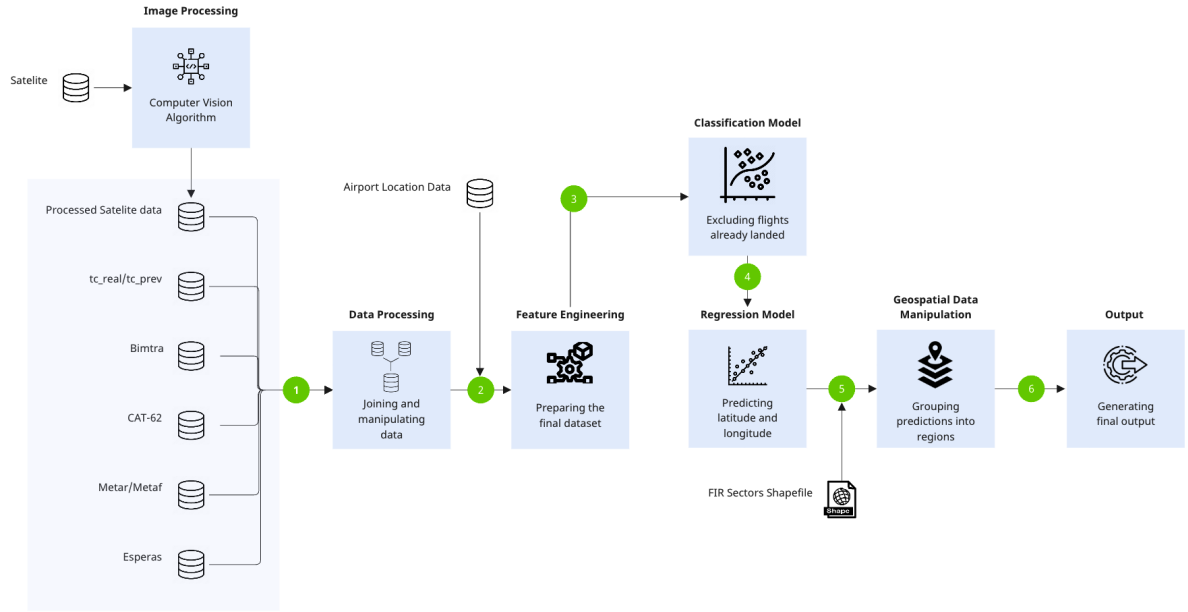

FIGURE 3.12 – Production Workflow for Predicting Aircraft Positions

The steps required to transition into a production environment are as follows:

1. **Data Acquisition:** Satellite images are acquired and preprocessed using a Computer Vision Algorithm to extract relevant features. The processed satellite data, along with other related datasets (`tc_real/tc_prev`, `Bimtra`, `CAT-62`, `Metar/Metaf`, and `Esperas`), are stored in a central database.

2. **Data Integration:** Different datasets are joined and manipulated to create a unified dataset. This integrated dataset is stored in a central database, making it accessible for feature engineering and model training.

3. **Feature Engineering:** The integrated data is processed to create features for the machine learning models. This includes handling missing values, encoding categorical variables, and scaling numerical features.

4. **Model Training:** Train the classification model to exclude flights that have already landed, ensuring only ongoing flights are considered. Train the regression model to predict the latitude and longitude of aircraft using the engineered features.

5. **Geospatial Data Manipulation:** Map the predicted latitude and longitude coordinates to specific FIR sectors using geospatial data manipulation techniques, ensuring each aircraft is correctly assigned to a region.

6. **Output Generation:** Generate the final predictions, including the number of aircraft in each FIR sector. This output is used to inform air traffic management and ensure safe and efficient flight operations.

7. **Automation:** Automate the entire workflow using cloud-based tools and services. This automation ensures real-time predictions, providing timely insights for air traffic management.

8. **Monitoring and Maintenance:** Continuously monitor the production environment to ensure smooth operation. Address any issues promptly and periodically retrain the models with new data to maintain accuracy.

By following this workflow, the methodologies and models developed during this research can be effectively transitioned into a production environment. The integration of automation and continuous monitoring guarantees that the system remains robust and adaptive to new data.

4 Results

4.1 Classification Models

4.1.1 MLP Classifier

The Multi-Layer Perceptron (MLP) classifier was evaluated for its performance in classifying the flight status. The confusion matrix and the classification report for the MLP classifier are shown in Figure 4.1 and Table 4.1, respectively.

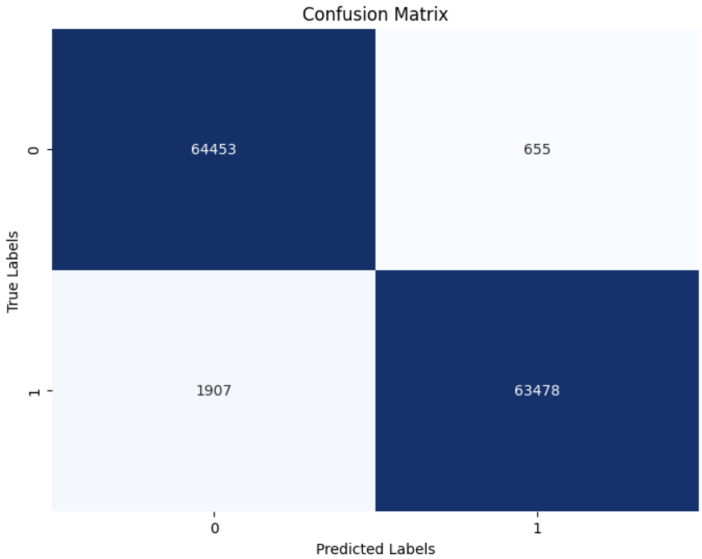

FIGURE 4.1 – Confusion Matrix for MLP Classifier

Class	Precision	Recall	F1-Score	Support
0	0.97	0.99	0.98	65108
1	0.99	0.97	0.98	65385
Accuracy		0.98		
Macro Avg	0.98	0.98	0.98	130493
Weighted Avg	0.98	0.98	0.98	130493

TABLE 4.1 – Classification Report for MLP Classifier

The MLP classifier demonstrated high accuracy, precision, and recall, indicating its effectiveness in correctly classifying both landed and in-flight aircraft. The slight difference

in precision and recall for class 0 and class 1 suggests a balanced performance across both classes.

4.1.2 Random Forest Classifier

The Random Forest classifier was also evaluated for its performance. The confusion matrix and the classification report for the Random Forest classifier are shown in Figure 4.2 and Table 4.2, respectively.

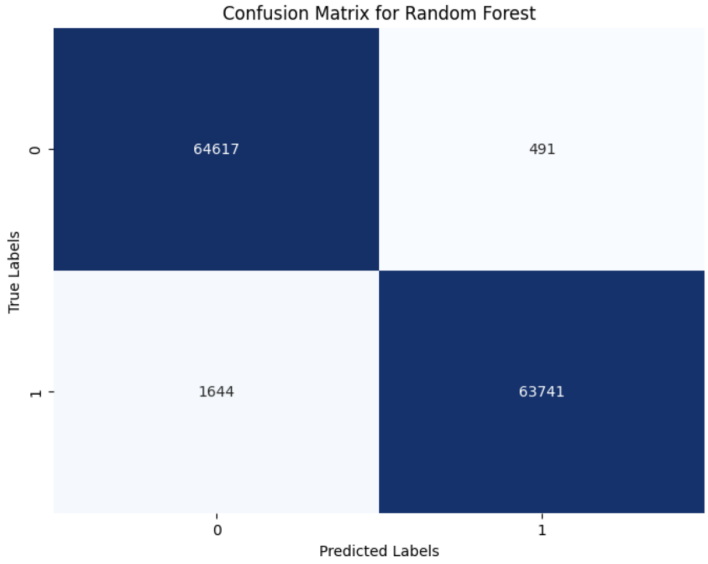

FIGURE 4.2 – Confusion Matrix for Random Forest Classifier

Class	Precision	Recall	F1-Score	Support
0	0.98	0.99	0.98	65108
1	0.99	0.97	0.98	65385
Accuracy		0.98		
Macro Avg	0.98	0.98	0.98	130493
Weighted Avg	0.98	0.98	0.98	130493

TABLE 4.2 – Classification Report for Random Forest Classifier

The Random Forest classifier also achieved high performance metrics, similar to the MLP classifier. Its balanced precision and recall values across both classes highlight its robustness in classification tasks.

4.1.3 XGBoost Classifier

The XGBoost classifier was evaluated next. The confusion matrix and the classification report for the XGBoost classifier are presented in Figure 4.3 and Table 4.3, respectively.

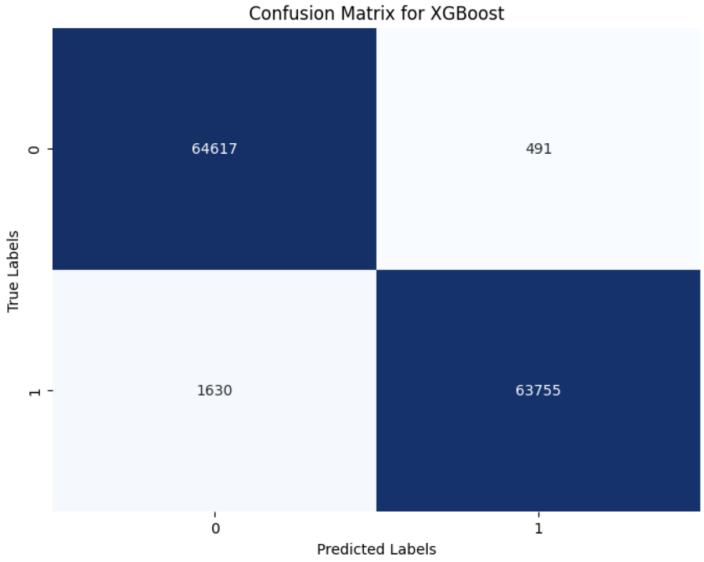

FIGURE 4.3 – Confusion Matrix for XGBoost Classifier

Class	Precision	Recall	F1-Score	Support
0	0.98	0.99	0.98	65108
1	0.99	0.98	0.98	65385
Accuracy		0.98		
Macro Avg	0.98	0.98	0.98	130493
Weighted Avg	0.98	0.98	0.98	130493

TABLE 4.3 – Classification Report for XGBoost Classifier

The XGBoost classifier maintained high accuracy and balanced precision and recall, confirming its effectiveness for this classification task. Its performance was comparable to that of the Random Forest and MLP classifiers.

4.1.4 Logistic Regression

Logistic Regression was also tested for comparison. The confusion matrix and the classification report are shown in Figure 4.4 and Table 4.4, respectively.

Class	Precision	Recall	F1-Score	Support
0	0.96	0.95	0.95	65108
1	0.95	0.96	0.95	65385
Accuracy		0.95		
Macro Avg	0.95	0.95	0.95	130493
Weighted Avg	0.95	0.95	0.95	130493

TABLE 4.4 – Classification Report for Logistic Regression

Although the Logistic Regression classifier achieved slightly lower performance metrics compared to the other classifiers, it still demonstrated robust accuracy, precision, and recall. Its simplicity and interpretability can make it a good baseline model.

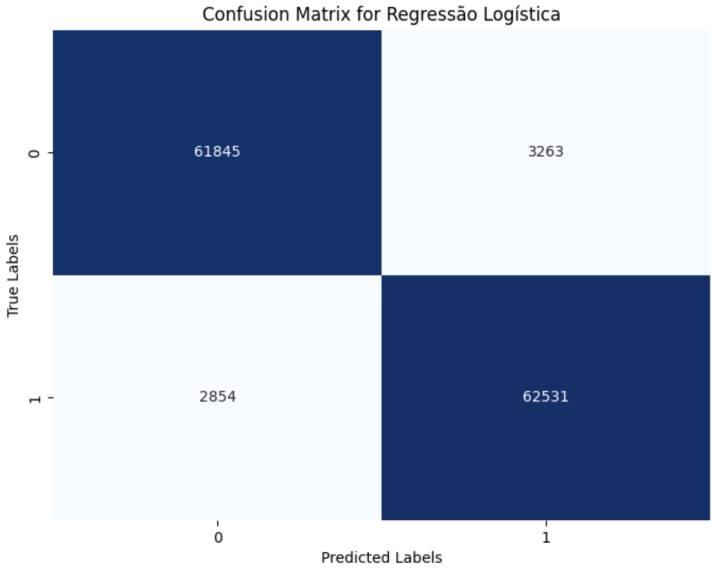

FIGURE 4.4 – Confusion Matrix for Logistic Regression

4.1.5 Voting Classifier

The Voting Classifier combines the predictions from the MLP, Random Forest, and XGBoost classifiers to improve overall performance. The confusion matrix and the classification report for the Voting Classifier are shown in Figure 4.5 and Table 4.5, respectively.

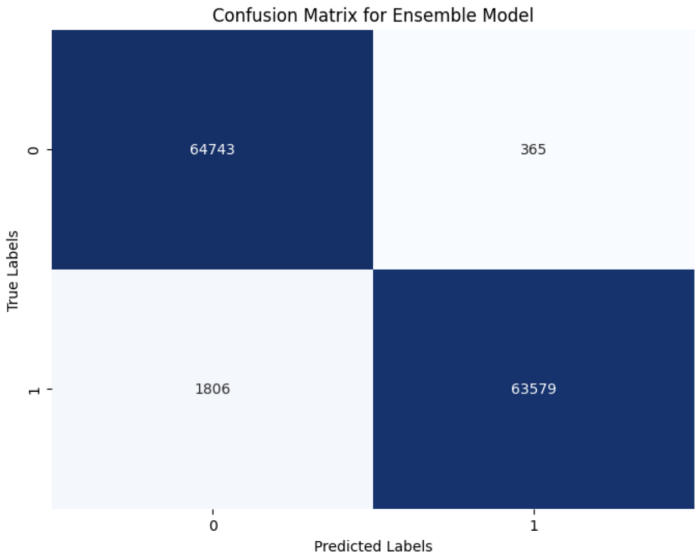

FIGURE 4.5 – Confusion Matrix for Voting Classifier

The Voting Classifier was chosen due to its robust performance and slight improvement in predicting class 0 (flights not yet landed), which is critical for accurately counting in-flight aircraft. The ensemble approach leverages the strengths of multiple models, leading to improved overall accuracy and reducing the likelihood of missing aircraft in the final count. This reduction in missed detections significantly enhances the reliability of the predictions, which is crucial for accurate air traffic management.

Class	Precision	Recall	F1-Score	Support
0	0.97	0.99	0.98	65108
1	0.99	0.97	0.98	65385
Accuracy		0.98		
Macro Avg	0.98	0.98	0.98	130493
Weighted Avg	0.98	0.98	0.98	130493

TABLE 4.5 – Classification Report for Voting Classifier

4.1.6 Discussion

The classification models for predicting whether an aircraft has landed or not within an hour demonstrated robust performance across various metrics. The results showed high precision, recall, and F1-scores for all classifiers, indicating their reliability in distinguishing between landed and in-flight aircraft. This performance can be attributed to several factors, including the richness of the dataset, which includes radar synthesis data providing real-time aircraft positions.

A comparative table summarizing the performance of each classifier is presented below:

Model	Precision	Recall	F1-Score	Accuracy
MLP Classifier	0.98	0.98	0.98	0.98
Random Forest Classifier	0.98	0.98	0.98	0.98
XGBoost Classifier	0.98	0.98	0.98	0.98
Logistic Regression	0.95	0.95	0.95	0.95
Voting Classifier (Ensemble)	0.98	0.98	0.98	0.98

TABLE 4.6 – Performance Comparison of Classification Models

The high performance of the classifiers aligns with findings in existing literature. For example, Cai et al. (CAI YUE LI; ZHU, In press) demonstrated significant accuracy improvements in predicting flight delays using deep learning approaches applied to time-evolving graphs. Their study integrated diverse datasets, including weather reports and real-time aircraft performance data, which parallels our comprehensive methodology and enhances predictive capabilities. Specifically, their one-hour ahead delay predictions achieved an average error of 5.884 ± 0.072 minutes. This result closely aligns with our findings, as these 5 minutes enable accurate predictions for the vast majority of flights, whether they have landed or not. It's noteworthy that their study did not incorporate radar synthesis data, unlike our thesis, which further complicates their forecasting efforts.

Furthermore, Tang (TANG, 2021) investigated flight delay predictions using various machine learning algorithms, including Decision Trees, Random Forests, and Gradient Boosted Trees. The study found that the Decision Tree algorithm achieved the highest accuracy of 0.9777, which is consistent with our findings where tree-based ensemble methods showed robust performance. Tang's study also emphasizes the importance of

detailed and multifaceted data, including meteorological conditions and flight schedules, which significantly enhances predictive accuracy.

In the present thesis, the ensemble method, specifically the Voting Classifier, was chosen for its robustness and slightly better performance in predicting true labels of in-flight aircraft. This choice is significant because accurately identifying aircraft that have not yet landed reduces the risk of undercounting in subsequent analyses. Undercounting penalizes the final predictions, leading to less reliable traffic management outcomes.

The high performance of the classifiers was expected given the one-hour prediction horizon. Predicting whether an aircraft will land within this period is facilitated by the availability of numerous flight data points and the typical punctuality of most flights within a short duration. The incorporation of radar synthesis data, which provides real-time updates, further enhances the models' accuracy by leveraging the latest available aircraft positions.

4.2 Regression Models

In this section, we present the results of various regression models trained to predict the future latitudes and longitudes of flights. The evaluation metrics were calculated over a period of 10 days, with predictions made hourly for the validation period. An arbitrary timestamp (2023-05-01 15:00h) was chosen to illustrate and compare the predictions and errors for all models.

4.2.1 XGBoost

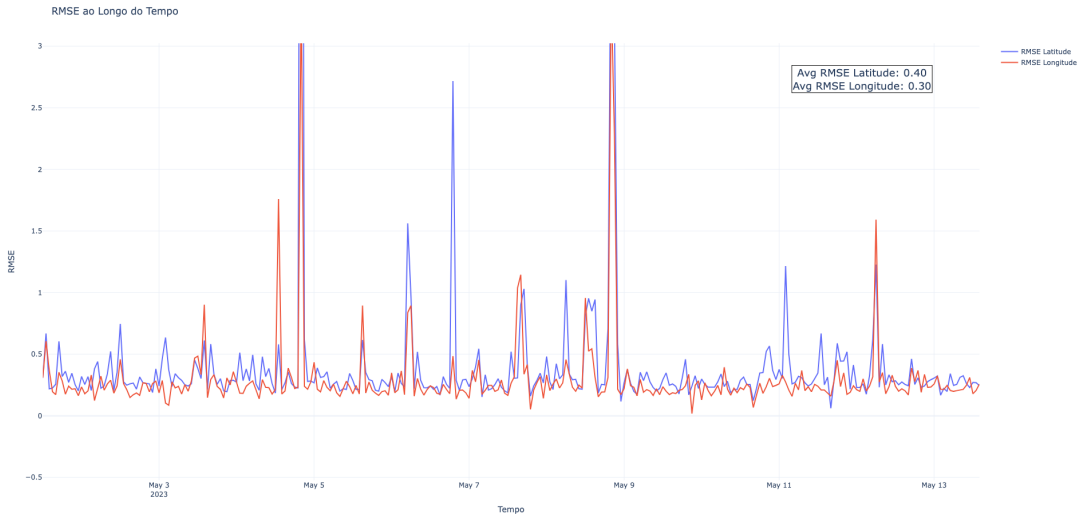

FIGURE 4.6 – Mean RMSE over the last 10 days for XGBoost model.

The plot in Figure 4.6 illustrates the Root Mean Square Error (RMSE) values for latitude and longitude predictions over the validation period. The XGBoost model achieved an average RMSE of 0.40 for latitude and 0.30 for longitude.

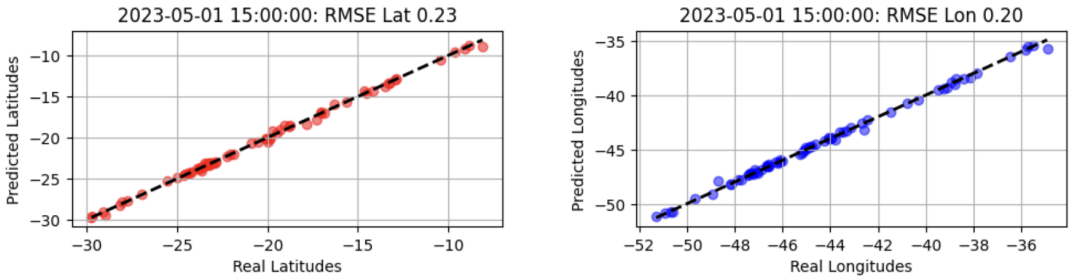

FIGURE 4.7 – Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h with RMSE lat 0.23 and RMSE lon 0.20 for XGBoost model.

Figure 4.7 shows the comparison between the real and predicted latitude and longitude values at a specific timestamp (2023-05-01 15:00h), demonstrating the accuracy of the XGBoost model in predicting flight positions. This model performed exceptionally well and was chosen as the primary model. A stacking approach was not opted for because incorporating the second-best model (NN4) deteriorated the results.

To visually assess the proximity of the results depicted in Figure 4.7, Figure 4.8 illustrates the actual and predicted flight trajectories within the analyzed 1-hour interval.

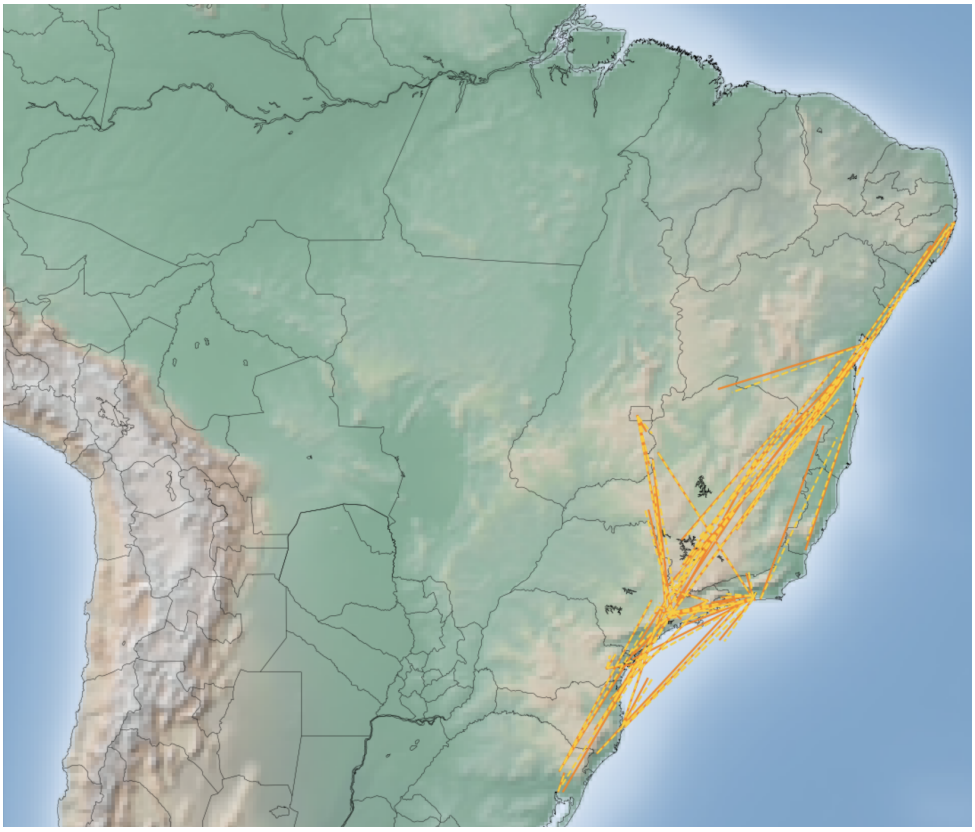

FIGURE 4.8 – Flight trajectories depicting actual (full lines) and predicted (dotted lines) paths.

4.2.2 Lasso Regression

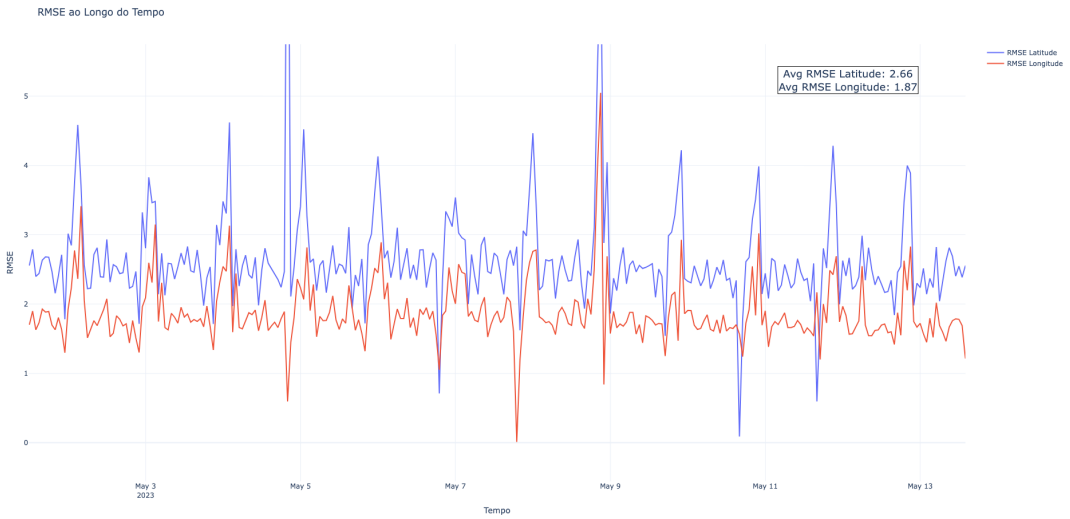

FIGURE 4.9 – Mean RMSE over the last 10 days for Lasso Regression model.

The Lasso Regression model was evaluated similarly, with the mean RMSE values presented in Figure 4.9. The Lasso model achieved an average RMSE of 2.66 for latitude and 1.87 for longitude.

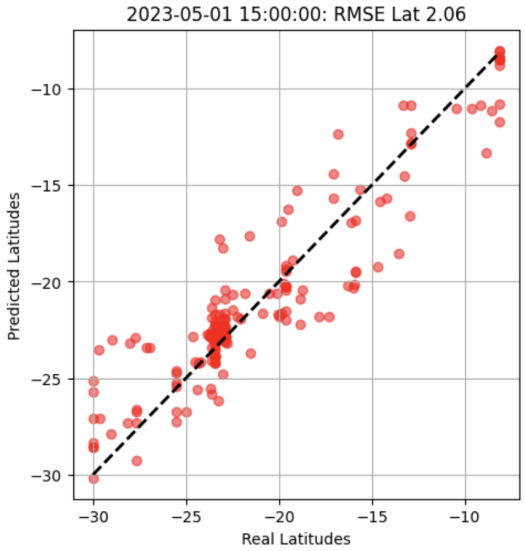

FIGURE 4.10 – Real latitudes versus predicted values for date reference 2023-05-01 15:00h for Lasso Regression model with RMSE lat 2.06.

Figure 4.10 displays the predicted and actual flight positions, indicating the model's performance at the same timestamp. The Lasso model showed poor performance overall.

4.2.3 Gradient Boost

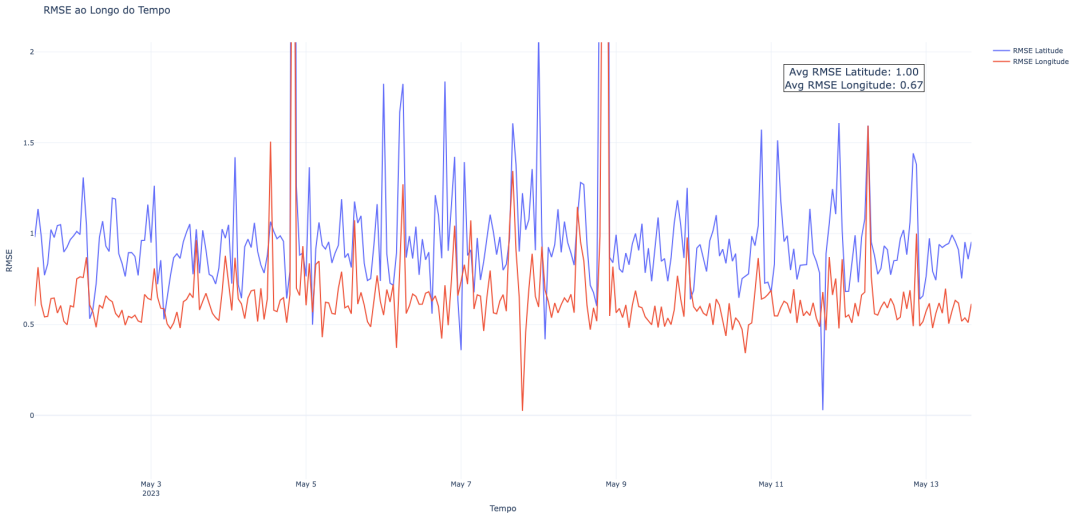

FIGURE 4.11 – Mean RMSE over the last 10 days for Gradient Boost model.

The Gradient Boosting model's performance over 10 days is illustrated in Figure 4.11. The Gradient Boost model achieved an average RMSE of 1.00 for latitude and 0.67 for longitude.

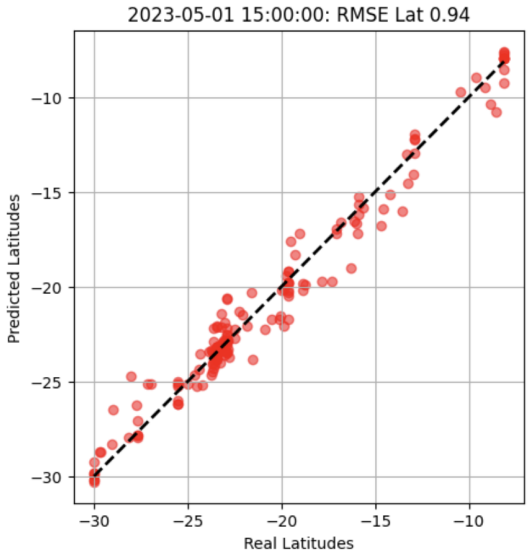

FIGURE 4.12 – Real latitudes versus predicted values for date reference 2023-05-01 15:00h for Gradient Boost model with RMSE lat 0.94.

Figure 4.12 demonstrates the Gradient Boost model's predictions versus the real values at the specified timestamp. While better than the Lasso model, the Gradient Boost model's performance was still not satisfactory.

4.2.4 Elastic Net

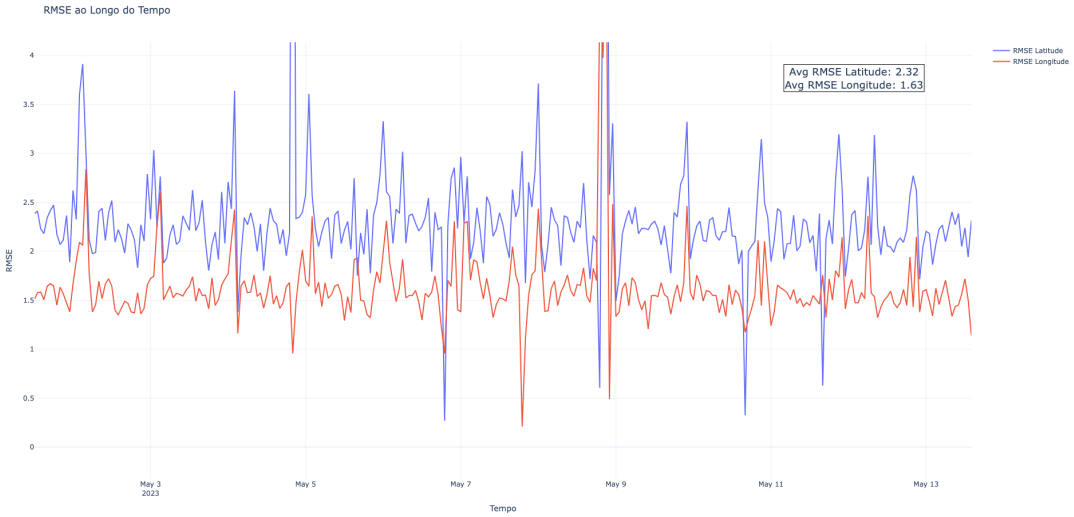

FIGURE 4.13 – Mean RMSE over the last 10 days for Elastic Net model.

Figure 4.13 presents the average RMSE values for the Elastic Net model, which achieved an average RMSE of 2.32 for latitude and 1.63 for longitude.

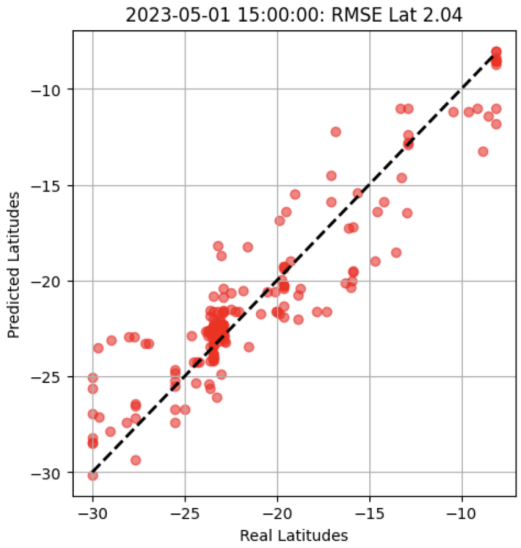

FIGURE 4.14 – Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Elastic Net model with RMSE lat 2.04.

Figure 4.14 demonstrates the Elastic Net model's predictions versus the real values at the specified timestamp. The Elastic Net model also showed poor performance in comparison with XGBoost Regressor.

4.2.5 Neural Network 1

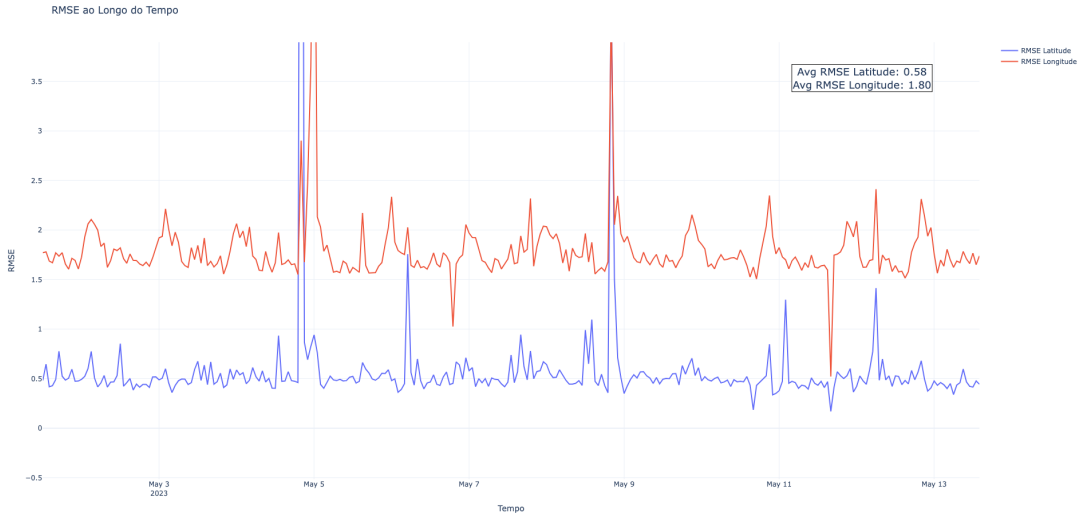

FIGURE 4.15 – Mean RMSE over the last 10 days for Neural Network 1 model.

The first neural network model's performance is shown in Figure 4.15, which achieved an average RMSE of 0.58 for latitude and 1.80 for longitude.

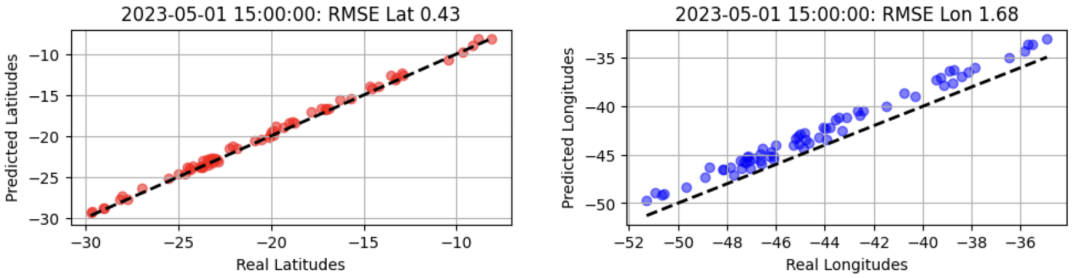

FIGURE 4.16 – Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Neural Network 1 model with RMSE lat 0.43 and RMSE lon 1.68.

Figure 4.16 shows that while latitude predictions were reasonable, the longitude predictions were significantly off, often above the curve, indicating less negative longitudes.

4.2.6 Neural Network 2

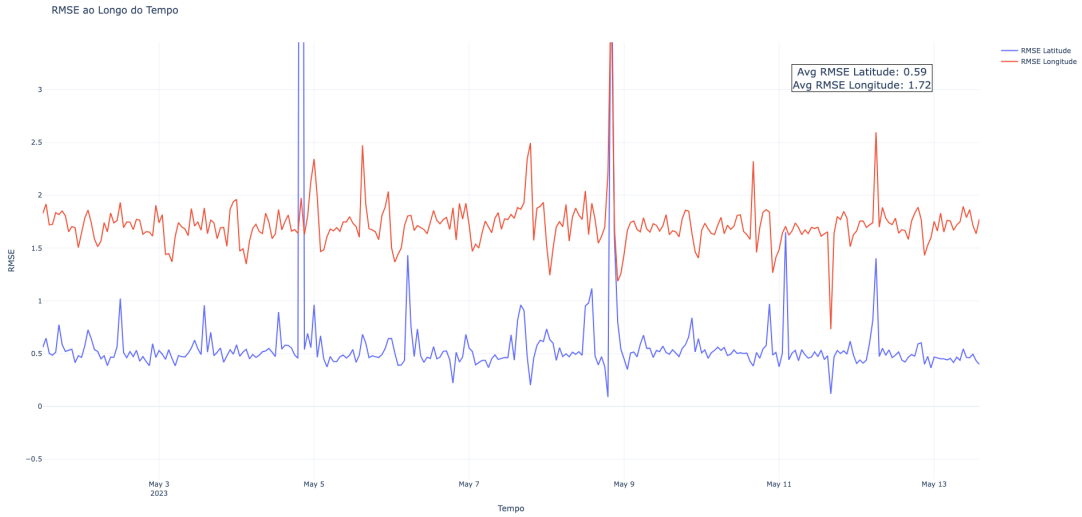

FIGURE 4.17 – Mean RMSE over the last 10 days for Neural Network 2 model.

Figure 4.17 illustrates the RMSE values for the second neural network model, which achieved an average RMSE of 0.48 for latitude and 1.41 for longitude.

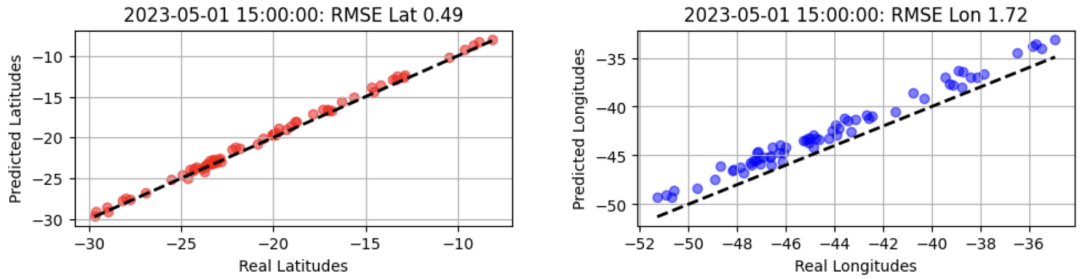

FIGURE 4.18 – Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Neural Network 2 model with RMSE lat 0.49 and RMSE lon 1.72.

Figure 4.18 shows that NN2 slightly reduced the displacement problem observed in NN1 but did not fully resolve it.

4.2.7 Neural Network 3

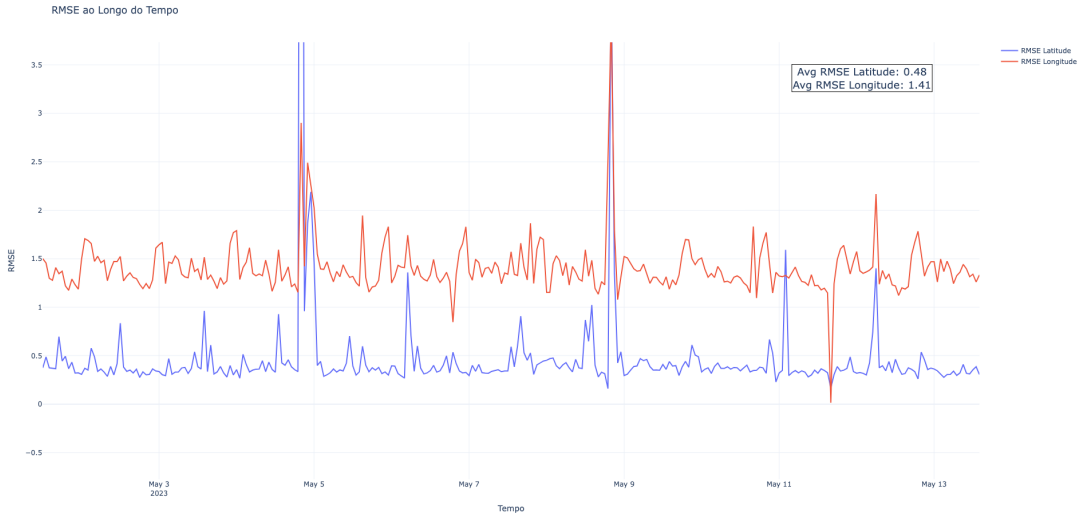

FIGURE 4.19 – Mean RMSE over the last 10 days for Neural Network 3 model.

The third neural network model's RMSE values are presented in Figure 4.19, achieving an average RMSE of 0.48 for latitude and 1.41 for longitude.

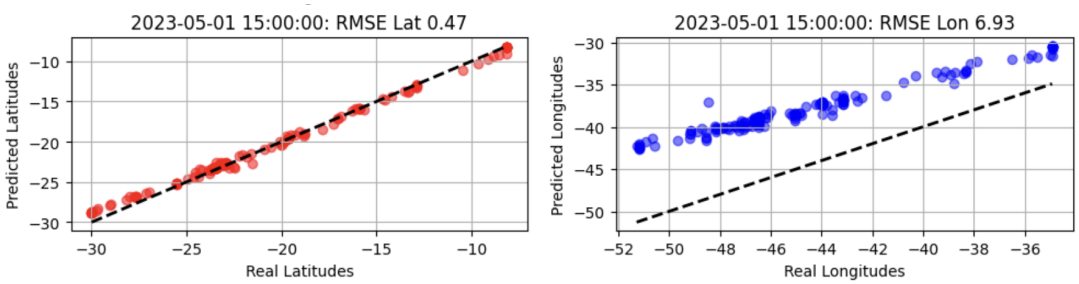

FIGURE 4.20 – Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Neural Network 3 model with RMSE lat 0.47 and RMSE lon 6.93.

Figure 4.20 shows that the third neural network model displaced the longitude predictions even more despite the greater number of neurons, resulting in poorer performance for longitude predictions while not improving latitude predictions.

4.2.8 Neural Network 4

This model used a different approach by combining latitude and longitude predictions into a single model with a dense layer outputting two values (latitude and longitude). This approach has both advantages and disadvantages:

Positive Aspects:

- **Simplicity:** Combining both latitude and longitude predictions into one model simplifies the overall architecture, reducing complexity and potential sources of error.
- **Correlation Capture:** The model can capture correlations between latitude and longitude, potentially leading to more accurate predictions.
- **Efficiency:** Training a single model is computationally more efficient than training separate models for latitude and longitude.

Negative Aspects:

- **Model Complexity:** Predicting two outputs simultaneously may increase the complexity of the model, making it harder to interpret and potentially leading to overfitting.
- **Loss of Flexibility:** A single model may not capture the unique characteristics of latitude and longitude prediction tasks as effectively as specialized models tailored to each coordinate.
- **Difficulty in Debugging:** Debugging and fine-tuning a combined model can be more challenging due to the intertwined nature of the predictions.

These points are discussed in more detail in sources such as Machine Learning Mastery, which explores the advantages and drawbacks of multi-output models (BROWNLEE, 2023).

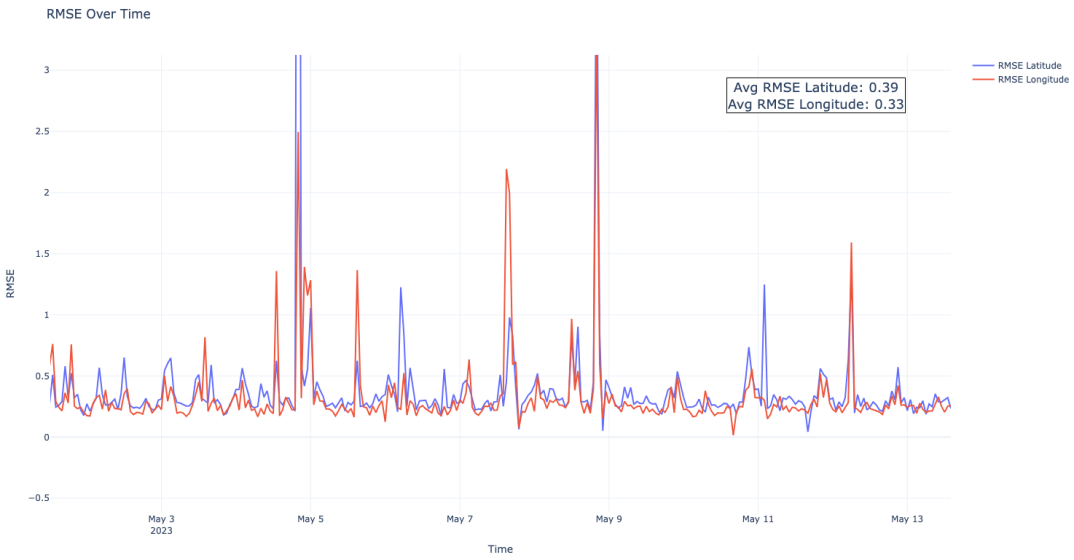

FIGURE 4.21 – Mean RMSE over the last 10 days for Neural Network 4 model.

Finally, Figure 4.21 shows the RMSE performance for the fourth neural network model, which achieved an average RMSE of 0.39 for latitude and 0.33 for longitude.

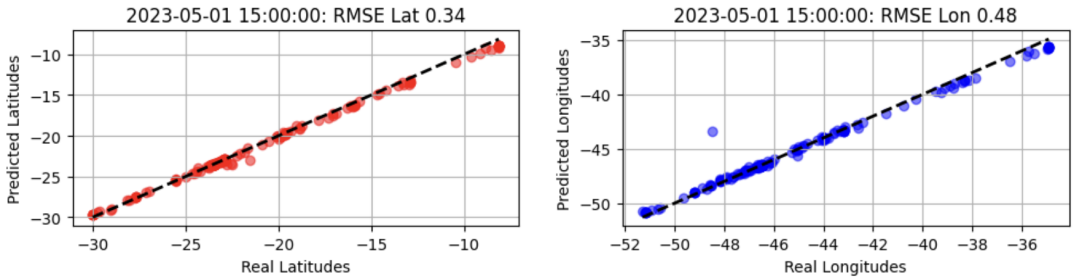

FIGURE 4.22 – Real latitudes and longitudes versus predicted values for date reference 2023-05-01 15:00h for Neural Network 4 model with RMSE lat 0.34 and RMSE lon 0.48.

Figure 4.22 shows the comparison between the real and predicted values, indicating excellent results, closely aligning with the XGBoost model. This model performed nearly as well as XGBoost, making it the second-best model evaluated.

4.3 Discussion

The regression models for predicting the future latitudes and longitudes of aircraft demonstrated varying degrees of performance, reflecting the complexity and challenges associated with precise spatial predictions over time. The evaluation was conducted over a period of 10 days with hourly predictions, providing a comprehensive assessment of each model's capabilities.

A summary of the performance metrics for all regression models is presented in the table below:

Model	**Average RMSE Latitude**	**Average RMSE Longitude**
XGBoost	0.40	0.30
Lasso Regression	2.66	1.87
Gradient Boosting	1.00	0.67
Elastic Net	2.32	1.63
Neural Network 1	0.58	1.80
Neural Network 2	0.59	1.72
Neural Network 3	0.48	1.41
Neural Network 4	0.39	0.33

TABLE 4.7 – Performance Comparison of Regression Models

The XGBoost model showed the best performance among all models tested, with an average RMSE of 0.40 for latitude and 0.30 for longitude. This high level of accuracy can be attributed to XGBoost's ability to handle large datasets and capture complex relationships through its gradient boosting framework. This model's robust performance

aligns with findings in the literature where gradient boosting algorithms have been shown to excel in regression tasks due to their ability to minimize errors iteratively. Studies, such as those by Chen and Guestrin (2016), have highlighted the effectiveness of XGBoost in various predictive modeling tasks (CHEN; GUESTRIN, 2016b).

The Lasso Regression model, in contrast, performed poorly, with an average RMSE of 2.66 for latitude and 1.87 for longitude. Lasso's limitation in capturing complex non-linear relationships between features likely contributed to its lower performance in this spatial prediction task. Elastic Net, another linear model, showed similar deficiencies, with an average RMSE of 2.32 for latitude and 1.63 for longitude.

The Gradient Boosting model achieved better performance than the Lasso and Elastic Net models, with an average RMSE of 1.00 for latitude and 0.67 for longitude. However, the results were still not satisfactory for precise spatial predictions.

Neural Network models exhibited a range of performances. Neural Network 1 achieved an average RMSE of 0.58 for latitude and 1.80 for longitude, with significant bias in longitude predictions. Neural Network 2 showed slight improvement but retained systematic biases. Neural Network 3's increased complexity did not translate into better performance, leading to higher RMSE values, particularly for longitude.

Neural Network 4, which combined latitude and longitude predictions into a single model, performed exceptionally well, achieving an average RMSE of 0.39 for latitude and 0.33 for longitude. This approach of predicting both outputs simultaneously offers several advantages and disadvantages:

In examining the discrepancies between predicted aircraft positions and their actual locations, it's essential to consider various factors influencing these deviations. One such factor is the inherent limitations in our predictive model, particularly regarding the spatial resolution of the data and the dynamic nature of aircraft trajectories.

For instance, the radar data available via the API is provided at intervals of 2 minutes, and sometimes the time difference between subsequent measurements is even larger. However, the actual radar operates every 4 seconds, which would significantly improve the accuracy of our models, especially for classification tasks. When evaluating the models, we often compare predictions at time t with radar data at time $t+2$ minutes. For context, an aircraft can traverse the entire longitudinal span of the Distrito Federal, approximately 75 kilometers, within 5 minutes at high speed. This rapid movement highlights the potential discrepancies in predicted versus actual positions, which can be rationalized by considering the swift movement of aircraft and the limitations of 2-minute radar intervals.

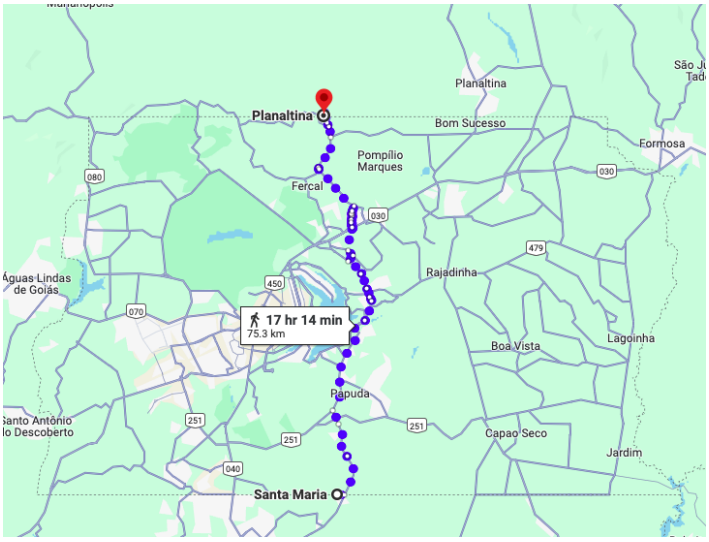

FIGURE 4.23 – Estimate of Longitudinal span of Distrito Federal. Adapted from Google Maps

Given these constraints, the regression models generally performed well, with errors in counting more likely arising from classification errors. The most significant prediction errors occurred in rare cases where the model incorrectly estimated the flight direction, roughly 1 in every 400 flights.

4.4 Sector Grouping

4.4.1 Specific Case Illustration

To illustrate the performance of the models in a specific scenario, the results for the reference date 2023-05-12 11:00h were analyzed. This analysis focuses on comparing predictions with actual aircraft positions within the FIR sectors. Studying this specific time frame is crucial for understanding critical aspects to analyze comprehensively across all cases and ensuring the replicability of tests and results.

Metric	**Value**
MAE for 'predicted'	0.652
MAE for 'mean velocity'	2.957
Correlation (Pearson) between 'predicted' and 'radar'	0.940
Correlation (Pearson) between 'mean velocity' and 'radar'	-0.065

TABLE 4.8 – Evaluation Metrics for the Selected Timestamp

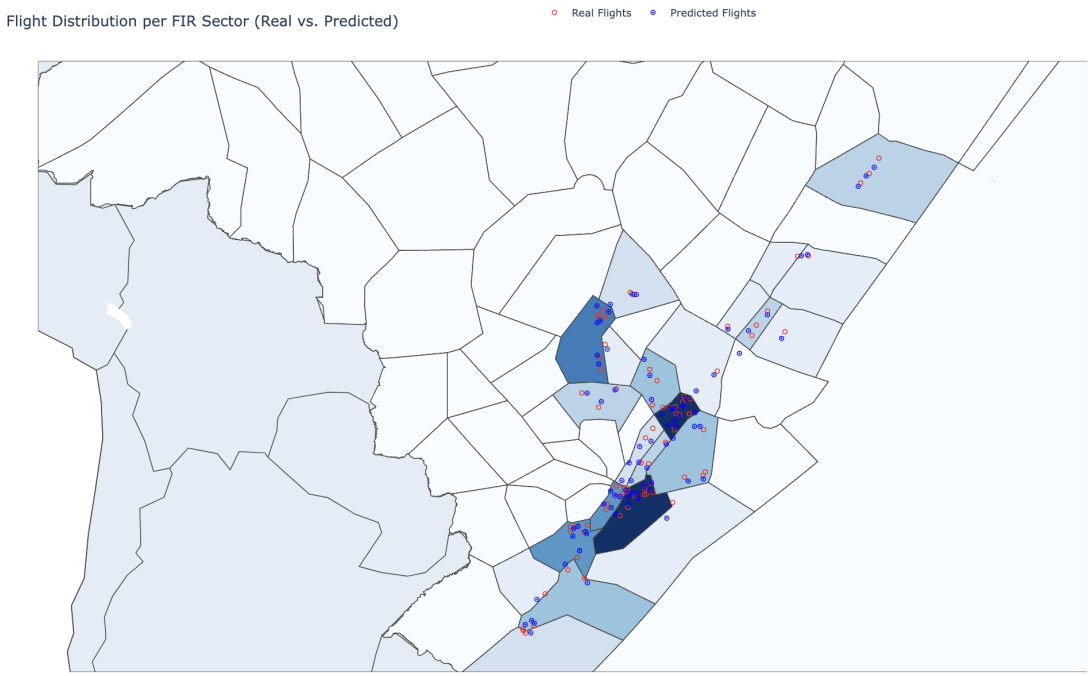

FIGURE 4.24 – FIR sector map with blue points representing predicted positions and red points representing actual positions of aircraft.

The following table summarizes the predictions for this timestamp:

Main Region	Radar	Predicted	Mean Velocity
SBBS_03	10	11.0	1.0
SBCW_05	10	9.0	2.0
SBBS_11	7	6.0	0.0
SBCW_18	6	7.0	4.0
SBCW_09	6	6.0	5.0
SBCW_04	4	4.0	5.0
SBBS_02	4	3.0	3.0
SBCW_08	4	5.0	4.0
SBRE_06	3	3.0	0.0
SBRE_13	3	2.0	1.0
SBBS_10	3	3.0	0.0
SBBS_04	3	3.0	9.0
SBBS_16	2	3.0	0.0
SBBS_05	2	5.0	1.0
SBCW_01	2	1.0	1.0
SBRE_10	1	0.0	1.0
SBRE_12	1	1.0	0.0
SBRE_11	1	2.0	4.0
SBBS_01	1	2.0	14.0
SBCW_06	1	1.0	1.0
SBCW_03	1	1.0	0.0
SBBS_18	1	1.0	4.0
SBRE_14	1	1.0	1.0

TABLE 4.9 – Predictions for FIR sectors on 2023-05-12 11:00h

To further illustrate the model's performance, we compare the predictions with a baseline model. The baseline model's predictions are represented by green points in the figure below.

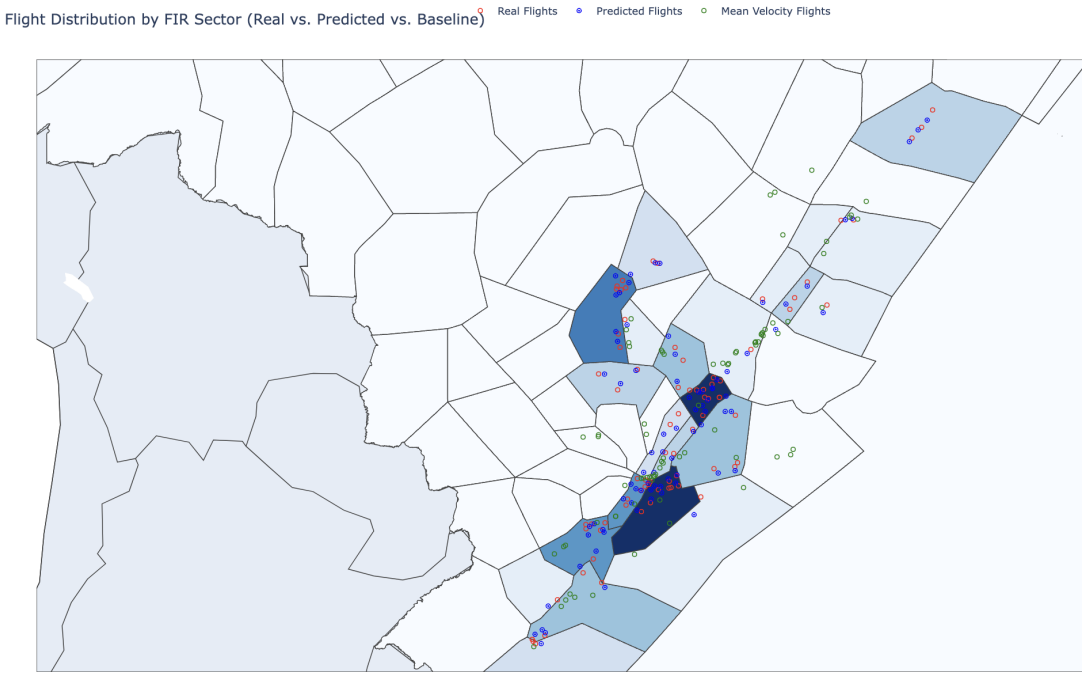

FIGURE 4.25 – Comparison of FIR sector predictions with baseline model (green points), predicted positions (blue points), and actual positions (red points).

The results for this specific case are very promising. Notably, many flights are positioned near FIR sector borders, leading to a double penalty when an aircraft exits one region and erroneously enters another. Despite these challenges, the model performs exceptionally well, particularly considering the radar resolution already discussed.

4.4.2 General Results in Validation Window

The evaluation process described in the specific time case section was repeated hourly throughout the entire validation period. The following graphs provide a comprehensive analysis of the model's performance over this timeframe:

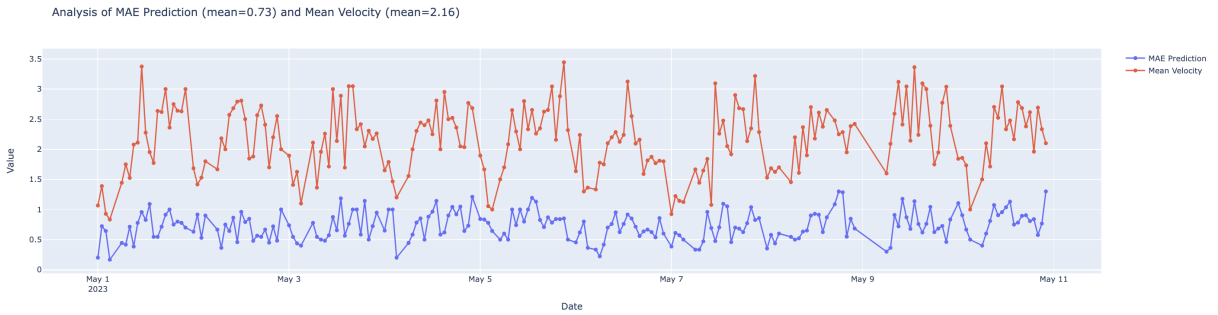

FIGURE 4.26 – MAE over time for the validation period.

The MAE over time graph shows a consistently low error rate, with an average of 0.73. This demonstrates the model's high accuracy in predicting aircraft positions within the sectors. The double penalization effect of MAE for sector misclassifications emphasizes the robustness of these results.

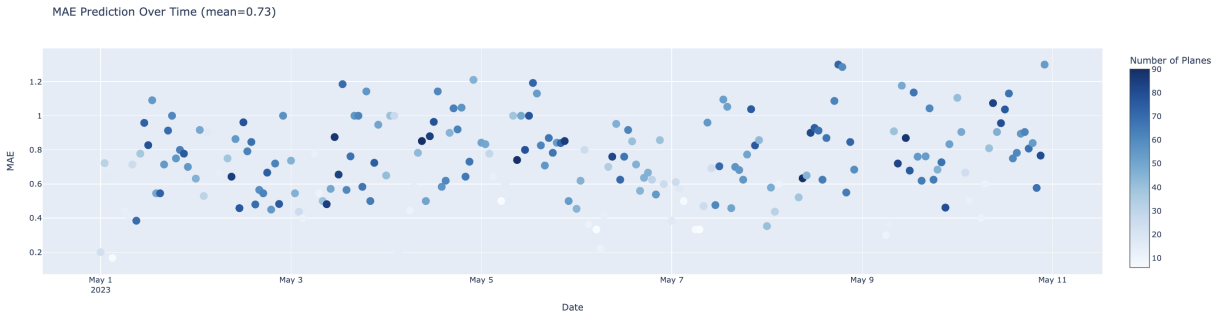

FIGURE 4.27 – MAE over time with the number of planes.

The MAE versus time graph with the number of planes shows that the MAE tends to be lower when fewer planes are in the air, which is expected due to the reduced likelihood of sector misclassifications. This consistency further demonstrates the reliability of the model under varying traffic conditions.

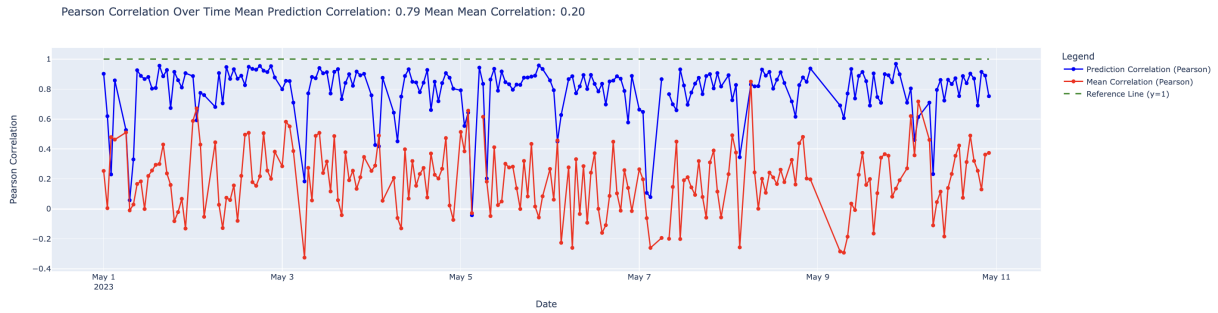

FIGURE 4.28 – Pearson correlation coefficient over time.

The Pearson correlation coefficient over time illustrates the model's ability to maintain a strong relationship between predicted and actual positions. Despite the sensitivity of

this metric to errors, especially during periods with fewer aircraft, the model achieved an average correlation of 0.79, indicating a strong predictive performance.

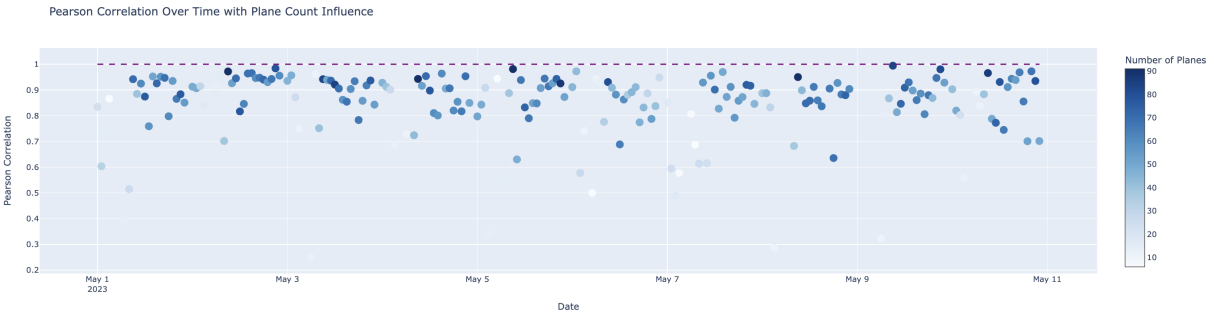

FIGURE 4.29 – Pearson correlation over time with the number of planes.

The Pearson correlation versus the number of planes graph highlights a trend where the correlation tends to decrease when fewer planes are present. This observation is consistent with the inherent behavior of the Pearson correlation coefficient, which amplifies the impact of errors when there are fewer data points available. However, these penalties associated with fewer flights do not pose a significant issue for the objectives of this study. Air traffic management issues typically arise during periods of high traffic volume, where accurate predictions are crucial for operational efficiency and safety. Therefore, while the model's performance may vary with lower flight volumes, its primary goal remains focused on enhancing predictability during high-traffic periods. Despite fluctuations in correlation under different traffic levels, the average correlation remains consistently high, underscoring the model's robustness and its ability to maintain accurate predictions when traffic volumes are substantial.

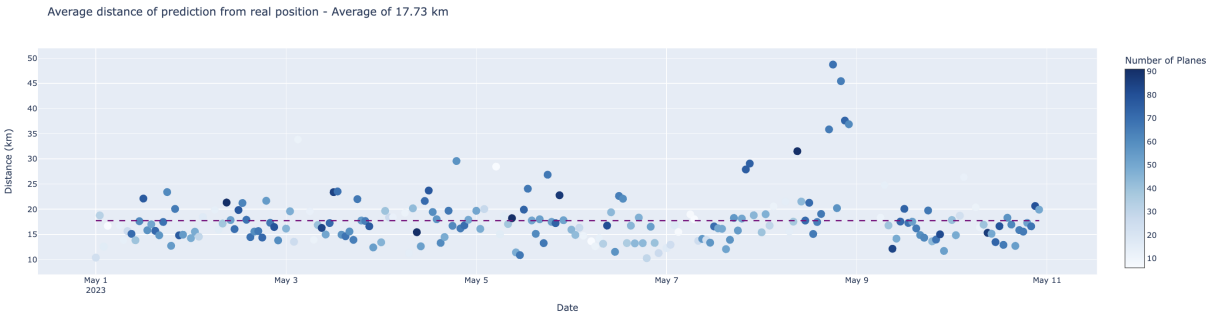

FIGURE 4.30 – Average distance per hour in a scatter plot, showing an average of 17.73 km of distance.

The average distance per hour scatter plot shows an average prediction error of 17.73 km. This metric is naturally more favorable because it excludes flights erroneously classified as landed, focusing only on the correctly predicted flights. Even the highest errors observed, close to 40 km, are relatively small considering the potential range of flight paths.

The algorithm's overall performance during the validation period is evaluated and summarized in Table 4.10.

Metric	**Average Value**	**Max Value**	**Min Value**
MAE	0.73	1.50	0.24
Pearson Correlation	0.79	0.99	0.28
Average Distance (km)	17.73	48.73	10.27

TABLE 4.10 – Summary of Performance Metrics Over the Validation Period

The results indicate a high level of accuracy in predicting aircraft positions and sector occupancy, demonstrating consistent performance across multiple metrics. The average MAE and Pearson correlation values suggest robust and reliable models, capable of handling the dynamic nature and inherent complexities of air traffic.

When it comes to the baseline model, lower performance was expected due to its reliance on average velocity, which is suboptimal in flight contexts. This approach overlooks the non-linear nature of flight trajectories, where even flights along the same route (origin-destination) exhibit considerable variability. This variability is illustrated in Figure 4.31, derived from the exploratory data analysis.

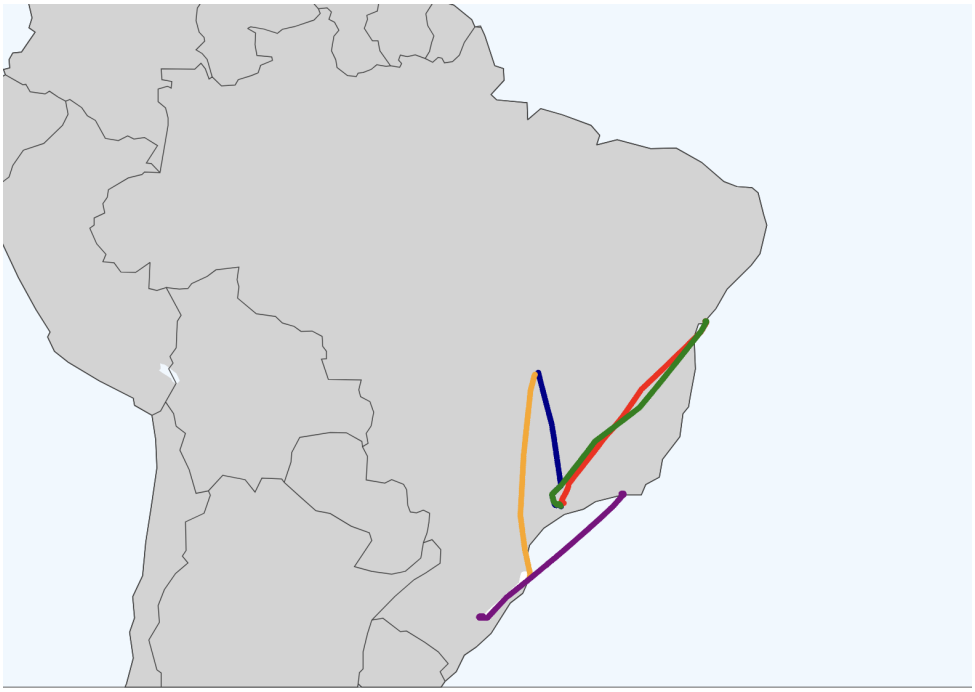

FIGURE 4.31 – Illustration of varied flight trajectories highlighting the non-linear paths observed during exploratory data analysis.

Comparison with the study by Brito et al. (BRITO MAYARA C. ROCHA MURCA; OLIVEIRA, 2021) reveals significant differences in approach and results. Brito's research focused on predicting sector occupancy using various machine learning models, including XGBoost, Random Forests, Support Vector Regression, and Artificial Neural Networks,

akin to the methodology utilized in this study. However, radar synthesis data integration in our approach provides higher temporal resolution and more accurate real-time aircraft positions.

Brito et al. (BRITO MAYARA C. ROCHA MURCA; OLIVEIRA, 2021) aimed to forecast sector crossing times and peak 15-minute occupancy for sectors within the Brasilia Area Control Center. Their findings underscored the effectiveness of machine learning models in enhancing predictability compared to baseline methods. In their study, Brito et al. reported an average RMSE of 5.432 minutes for sector entry times and 5.436 minutes for sector exit times using the XGBoost model for sector SBBS S4, achieving a corresponding sector occupancy accuracy of 77.6%. For sector SBBS S15, the XGBoost model achieved an RMSE of 3.137 minutes for sector entry times and 3.950 minutes for sector exit times, with an occupancy accuracy of 74.2%.

Despite employing different prediction metrics, the results consistently demonstrate the added value of radar data, which provides a deeper layer of information with enhanced temporal resolution. In contrast to Brito et al. (BRITO MAYARA C. ROCHA MURCA; OLIVEIRA, 2021), whose focus on predicting sector crossing times and occupancy would require substantial computational power to cover multiple regions simultaneously, the present thesis aimed to analyze multiple regions concurrently.

4.4.3 Evaluation in Specific Conditions

This section evaluates the model's robustness by analyzing its performance under various challenging scenarios. Six exceptional cases were selected from the general test set to provide a deeper insight into the model's capabilities. These scenarios include high traffic periods, public holidays, adverse weather conditions, and low-traffic times, each presenting unique challenges for air traffic prediction:

1. **05/05/2023 - 19h (Friday Night):** Increased air traffic due to weekend travel results in diverse flight patterns, challenging the model's predictive accuracy.

2. **01/05/2023 - 14h (Labor Day):** Public holidays like Labor Day often lead to altered travel patterns, testing the model's adaptability to fluctuations in air traffic dynamics.

3. **28/04/2023 - 10h (Rainy Day in São Paulo):** Adverse weather conditions significantly impact aviation operations. The day with the highest occurrence of rain in São Paulo in 2023 was selected to test the model under these conditions (Climate-Data.org, 2023).

4. **07/05/2023 - 21h (Sunday Night):** The end of the weekend sees complex travel patterns as people return from trips, providing a varied traffic scenario.

5. **01/05/2023 - 3h (Wednesday Early Morning):** Low-traffic periods allow for evaluation under conditions with reduced air traffic, testing the model's precision in less complex scenarios.

6. **24/12/2022 - 9h (Christmas Eve):** Increased holiday travel and seasonal weather effects create unique traffic patterns, testing the model's ability to manage these variations.

Scenario	**Pearson**	**MAE**	**MAE (Mean Velocity)**
Friday Night	0.874	1.11	2.16
Labor Day	0.862	0.55	1.77
Rainy Day in São Paulo	0.865	0.76	2.24
Sunday Night	0.945	0.57	3.22
Wednesday Early Morning	0.857	0.17	0.83
Christmas Eve	0.905	0.93	2.00

TABLE 4.11 – Performance Metrics for Exceptional Test Cases

The results from these scenarios highlight the model's ability to maintain high predictive accuracy across diverse conditions, though certain scenarios present specific challenges:

1. **Friday Night:** The model exhibits a Pearson correlation of 0.874 and an MAE of 1.11. The higher MAE is due to the increased number of flights, which inherently raises potential errors, yet the robust Pearson correlation underscores the model's reliability.

2. **Labor Day:** Achieving a Pearson correlation of 0.862 and an MAE of 0.55, the model adapts well to the fluctuating air traffic dynamics associated with this public holiday.

3. **Rainy Day in São Paulo:** Under adverse weather conditions, the model maintains a Pearson correlation of 0.865 and an MAE of 0.76.

4. **Sunday Night:** With the highest Pearson correlation of 0.945 and an MAE of 0.57, the model effectively predicts aircraft positions during varied travel patterns at the end of the weekend.

5. **Wednesday Early Morning:** During low-traffic periods, the model achieves a Pearson correlation of 0.857 and a minimal MAE of 0.17.

6. **Christmas Eve:** The model performs well under holiday-specific conditions with a Pearson correlation of 0.905 and an MAE of 0.93.

Overall, the model's consistent performance, particularly in terms of Pearson correlation, emphasizes its robustness. However, varying MAE values indicate areas for further refinement, especially under high-traffic conditions like Friday nights. The integration of high-resolution radar data significantly enhances predictive accuracy in dynamic and challenging air traffic environments, as demonstrated by these results. This advancement underscores the model's potential to improve air traffic management and optimize decision-making related to traffic flow and capacity management.

4.4.4 Limitations and Next Steps

While the current model demonstrates strong predictive capabilities, there are several areas where improvements and further exploration are warranted:

- **Retaining Models with Higher Resolution Radar Data:** To fully leverage the potential of radar data, it will be necessary to retrain the models using the highest possible resolution of radar data. The enhanced computational power available at ICEA will facilitate this retraining. However, it is important to note that incorporating such high-resolution data may yield slightly different results, and these outcomes should be analyzed thoroughly to ensure consistent and reliable performance.

- **Enhancing Classification Model Training:** Although the classification models currently exhibit excellent performance, it would be beneficial to incorporate more data points with landings occurring very close to the prediction time (less than 5 minutes). This can be achieved by extending the training period to include more data, which ICEA's extensive data repository can undoubtedly provide. This enhancement would further strengthen the model's ability to handle edge cases and improve overall accuracy.

- **Exploring Reinforcement Learning:** The entire framework of this study could be adapted for implementation using reinforcement learning algorithms. Such algorithms would enable the model to continuously learn and adapt from incoming data, thereby improving its performance over time. Reinforcement learning could offer a dynamic and responsive approach to air traffic prediction, adjusting to new patterns and anomalies as they arise.

- **Increasing Prediction Frequency:** With greater computational resources, it would be feasible to experiment with more frequent predictions, such as every 30 minutes

instead of hourly. This increased frequency would not only provide more granular insights but also result in a larger dataset, enhancing the model's training and validation processes. Such an approach was previously constrained by the computational limitations during the preparation of this thesis.

- **Incorporating Additional Data Sources:** Future work could benefit from integrating additional data sources such as weather forecasts, aircraft performance metrics, and real-time air traffic control instructions. These data points could provide a more comprehensive understanding of the factors influencing aircraft trajectories and improve prediction accuracy.

In summary, while the current model provides a robust framework for air traffic prediction, further refinements and explorations are necessary to fully harness its potential. Leveraging ICEA's computational power, integrating additional data sources, and exploring advanced machine learning techniques will pave the way for more accurate and reliable air traffic management solutions.

5 Conclusion

This thesis introduces an innovative data-driven approach designed to forecast airspace sector demand and predict aircraft positions within Brazilian airspace, with the goal of optimizing air traffic flow and capacity management. The methodology involved extensive data preparation, integrating computer vision methods and employing a two-step prediction process: firstly, identifying aircraft landing events, and secondly, predicting latitude and longitude coordinates if the aircraft remains airborne. This approach leveraged high-resolution radar synthesis data, alongside flight trajectories, weather conditions, and air traffic flow management constraints, employing advanced machine learning techniques including XGBoost, Random Forests, and Artificial Neural Networks.

The results clearly demonstrated that the data-driven approach significantly surpassed baseline models relying on mean velocity estimates. Particularly, the XGBoost model exhibited superior predictive accuracy both for landing predictions and subsequent position estimations. Evaluating the models under challenging conditions, such as Friday nights, public holidays, and adverse weather scenarios, underscored their robustness and adaptability. Instances with lower air traffic volume (less than 10 aircraft), which typically experience greater penalization, do not significantly impact the core objectives of this study. It is well-established that air traffic management challenges primarily arise during periods of heightened traffic, where accurate predictions are pivotal.

Key findings underscore the significant advantages gained from integrating high-resolution radar data to enhance predictive accuracy. Improving the temporal resolution of radar data and expanding the training dataset holds promise for further optimizing model performance. Moreover, exploring the integration of reinforcement learning algorithms presents an exciting opportunity for ongoing model enhancement. Future efforts should prioritize retraining models with full-resolution radar data, utilizing the computational resources available at ICEA. Augmenting the training dataset, particularly for predictions near landing, could bolster model precision. Additionally, incorporating reinforcement learning mechanisms could enable continuous adaptation of models to new data, thereby refining their predictive capabilities over time.

Bibliography

ABADI, M. *et al.* **TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems**. 2015. `https://www.tensorflow.org/`. Accessed: 2024-06-10.

AGRAWAL, R. **Evaluation Metrics for Your Regression Model**. [*S.l.*], 2024. Available at: https://www.analyticsvidhya.com/blog/2021/06/evaluation-metrics-for-your-regression-model/.

AIRPORT Coordinates and Information. `https://www.latlong.net/`. Accessed: 2024-06-09.

AL., L. et. **imbalanced-learn: A Python Package to Tackle the Curse of Imbalanced Datasets in Machine Learning**. [*S.l.*], 2021. Available at: https://imbalanced-learn.org/.

AL., P. et. **scikit-learn: Machine Learning in Python**. [*S.l.*], 2021. Available at: https://scikit-learn.org/.

Apache Spark. **Apache Spark Documentation**. 2024. `https://spark.apache.org/docs/latest/api/python/`. Accessed: 2024-06-09.

BISHOP, C. M. **Pattern Recognition and Machine Learning**. Berlin, Heidelberg: Springer, 2006. ISBN 9780387310732.

BREIMAN, L. Random forests. **Machine Learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

BRITO MAYARA C. ROCHA MURCA, M. d. O. I. R.; OLIVEIRA, A. V. A machine learning-based predictive model of airspace sector occupancy. **AIAA Scitech 2021 Forum**, 2021. Available at: https://arc.aiaa.org/doi/abs/10.2514/6.2021-2324.

BROWNLEE, J. **How to Develop Multi-Output Regression Models with Python**. 2023. Accessed: 2024-06-19. Available at: https://machinelearningmastery.com/multi-output-regression-models-with-python/.

CAI YUE LI, Y. F. K.; ZHU, Y. A deep learning approach for flight delay prediction through time-evolving graphs. **IEEE Transactions on Intelligent Transportation Systems**, p. 1–11, In press. Available at: https://hal.science/hal-03428046.

CARTER, D. **Speed Up Your API Requests with Thread Pools**. 2020. Accessed: 2024-07-01. Available at: https://daniel-carter.medium.com/speed-up-your-api-requests-with-thread-pools-588ab8e66f19.

CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. Smote: Synthetic minority over-sampling technique. **Journal of Artificial Intelligence Research**, v. 16, p. 321–357, 2002.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. *In*: ACM. **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**. **Proceedings** [...]. [*S.l.: s.n.*], 2016. p. 785–794.

CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. **Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**, ACM, p. 785–794, 2016.

Climate-Data.org. **Highest occurrence of rain in São Paulo in 2023**. 2023. `https://pt.climate-data.org/america-do-sul/brasil/sao-paulo/sao-paulo-655/t/abril-4/`. Accessed: 2024-06-19.

Contentful. **REST API**. 2024. Available at: https://images.ctfassets.net/vwq10xzbe6iz-/5sBH4Agl614xM7exeLsTo7/9e84dce01735f155911e611c42c9793f/rest-api.png.

DEMPSEY, C. **What is GIS?** 2024. Available at: https://www.geographyrealm.com/what-is-gis/.

Departamento de Controle do Espaço Aéreo (DECEA). **DECEA - Departamento de Controle do Espaço Aéreo**. 2024. Available at: https://geoaisweb.decea.gov.br/.

DISTANCES on Earth 2: The Haversine Formula. 2023. Available at: https://www.themathdoctors.org/distances-on-earth-2-the-haversine-formula/.

ENDEAVOURS, P. C. **MLP vs CNN**. 2019. Available at: https://www.peculiar-coding-endeavours.com/2019/mlp_vs_cnn/.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures**. [*S.l.*]: University of California, Irvine, 2000. Doctoral dissertation.

FLAXMAN, D. M. **Feature Engineering in Machine Learning**. 2022. `https://www.heavy.ai/technical-glossary/feature-engineering`. Accessed: 2024-06-08.

FRIEDMAN, J. H. Greedy function approximation: A gradient boosting machine. **Annals of Statistics**, v. 29, n. 5, p. 1189–1232, 2001.

GEEKSFORGEEKS. **XGBoost**. 2023. Available at: https://www.geeksforgeeks.org/xgboost/.

GeoPandas Development Team. **GeoPandas: Python tools for geographic data**. 2013–2024. Accessed: 2024-06-15. Available at: https://geopandas.org.

GONZALEZ, R. C.; WOODS, R. E. **Digital Image Processing**. [*S.l.*]: Prentice Hall, 2002.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, MA: MIT Press, 2016. ISBN 9780262035613.

Google. **Google Colaboratory Documentation**. 2024. `https://colab.research.google.com/notebooks/intro.ipynb`. Accessed: 2024-06-09.

GRISEL, O.; TEAM the joblib development. **joblib: Lightweight Pipelines for Python**. [*S.l.*], 2021. Available at: https://joblib.readthedocs.io/.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning: Data Mining, Inference, and Prediction**. 2. ed. New York: Springer, 2009. ISBN 9780387848570.

HIREGOUDAR, S. **Ways to Evaluate Regression Models**. 2020. Accessed: 2024-06-20. Available at: https://towardsdatascience.com/ways-to-evaluate-regression-models-77a3ff45ba70.

ICEA, B. A. C. I. **API Documentation**. 2023. `http://montreal.icea.decea.mil.br:5002/api/v1/docs`.

International Civil Aviation Organization (ICAO). **Annex 11 - Air Traffic Services**. 15th. ed. Montreal, Canada: ICAO, 2022.

International Civil Aviation Organization (ICAO). **History of ICAO**. 2024. `https://www.icao.int/about-icao/history/pages/default.aspx`. [Online; accessed 8-June-2024].

LITTLE, R. J. A.; RUBIN, D. B. **Statistical Analysis with Missing Data**. 2nd. ed. [*S.l.*]: Wiley, 2014. ISBN 978-0470526798.

MASTER, M. **Introduction: Random Forest Classification by Example**. 2019. Available at: https://medium.com/@mrmaster907/introduction-random-forest-classification-by-example-6983d95c7b91.

MURPHY, K. P. **Machine Learning: A Probabilistic Perspective**. Cambridge, MA: MIT Press, 2012. ISBN 9780262018029.

NumPy Community. **NumPy Documentation**. 2024. `https://numpy.org/doc/`. Accessed: 2024-06-09.

OpenCV. **OpenCV Documentation**. 2024. `https://docs.opencv.org/`. Accessed: 2024-06-09.

ORELLANA, E. **SMOTE**. 2020. `https://emilia-orellana44.medium.com/smote-2acd5dd09948`. Accessed: 2024-06-08.

Pandas Development Team. **Pandas Documentation**. 2024. `https://pandas.pydata.org/pandas-docs/stable/`. Accessed: 2024-06-09.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. **precision_recall_fscore_support — scikit-learn 1.5.0 documentation**. [*S.l.*], 2023. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. **VotingClassifier — scikit-learn 1.5.0 documentation**. [*S.l.*], 2023. Available at: https://scikit-learn.org/stable/modules/ensemble.htmlvoting-classifier.

Plotly. **Plotly Documentation**. 2024. `https://plotly.com/python/`. Accessed: 2024-06-09.

Python Metar Library. **Metar Library Documentation**. 2024. `https://pypi.org/project/Metar/`. Accessed: 2024-06-09.

Python Software Foundation. **concurrent.futures — Launching parallel tasks**. 2023. `https://docs.python.org/3/library/concurrent.futures.html`. Accessed: 2023-06-09.

Python Software Foundation. **Math Module Documentation**. 2024. `https://docs.python.org/3/library/math.html`. Accessed: 2024-06-09.

Python Software Foundation. **re Module Documentation**. 2024. `https://docs.python.org/3/library/re.html`. Accessed: 2024-06-09.

RASCHKA, S. **EnsembleVoteClassifier**. 2019. Available at: https://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/.

REITZ, K. **Requests: HTTP for Humans**. 2023. Available at: https://docs.python-requests.org/en/master/.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, Springer, v. 323, n. 6088, p. 533–536, 1986.

SAMUEL, A. L. Some studies in machine learning using the game of checkers. **IBM Journal of Research and Development**, v. 3, n. 3, p. 210–229, 1959.

SIGLES Programme. **Data Processing, Analysis and Mapping**. 2018. Website. Accessed: 2024-06-15. Available at: https://www.sigles-sante-environnement.fr/en/study-methodology/data-processing-analysis-and-mapping/.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. *In*: **Advances in Neural Information Processing Systems**. **Proceedings** [...]. [*S.l.: s.n.*], 2012. v. 25.

SPICEWORKS. **What is a Neural Network?** 2021. Available at: https://www-.spiceworks.com/tech/artificial-intelligence/articles/what-is-a-neural-network/.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. 2. ed. Cambridge, MA: MIT Press, 2018. ISBN 9780262039246.

TANG, Y. Airline flight delay prediction using machine learning models. *In*: **2021 5th International Conference on E-Business and Internet (ICEBI 2021)**. **Proceedings** [...]. Singapore, Singapore: ACM, 2021. Available at: https://doi.org/10.1145/3497701.3497725.

TIBSHIRANI, R. Regression shrinkage and selection via the lasso. **Journal of the Royal Statistical Society: Series B (Methodological)**, v. 58, n. 1, p. 267–288, 1996.

TUKEY, J. W. **Exploratory Data Analysis**. [*S.l.*]: Addison-Wesley, 1977. ISBN 978-0201076165.

TURING, A. M. Computing machinery and intelligence. **Mind**, Oxford University Press, LIX, p. 433–460, 1950.

VILELA, B. **Latam Challenge Repository**. 2023. `https://github.com/vilelabruno/latamChalenge/blob/main/README.md`. Accessed: 2024-06-09.

WIJAYA, C. Y. **Cross-Validation to Improve Model Performance**. 2023. `https://cornellius.substack.com/p/cross-validation-to-improve-model`. Accessed: 2024-06-08.

WILLMOTT, C. J.; MATSUURA, K. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. **Climate Research**, v. 30, p. 79–82, 2005.

WRAGG, D. W. **A Dictionary of Aviation**. first. [*S.l.*]: Osprey, 1973. 133 p. ISBN 9780850451634.

XGBOOST Documentation. 2016. Available at: https://xgboost.readthedocs.io/.

ZOU, H.; HASTIE, T. Regularization and variable selection via the elastic net. **Journal of the Royal Statistical Society: Series B (Statistical Methodology)**, v. 67, n. 2, p. 301–320, 2005.

Appendix A - Development of Custom Algorithms for Specialized Tasks

A.1 Computer Vision Algorithm

The following code demonstrates the computer vision algorithm used in this study for processing and analyzing images:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
import pandas as pd
import json
import sys

# Define input and output directories
input_dir = '/content/drive/MyDrive/TG/Dados/Images/'
output_dir = '/content/drive/MyDrive/TG/Dados/Contents/'

# Create output directory if it does not exist
os.makedirs(output_dir, exist_ok=True)

# Function to load and process the image
def process_image(img_path, output_path):
    # Load the image
    imageRoot = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    image = cv2.imread(img_path, cv2.IMREAD_COLOR)[740:len(imageRoot) - 100, 940:len(
    imageRoot[0]) - 100]

    kernel = np.ones((5, 5), np.uint8)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    closing = cv2.morphologyEx(gray, cv2.MORPH_CLOSE, kernel)
    closing = (closing * -1) + np.max(closing)
    closing[closing < 100] = 0
    _, thresh = cv2.threshold(closing, 50, 255, cv2.THRESH_BINARY)
```

```
28
29      # Sort contours by their area in descending order
30      thresh = np.array(thresh, np.uint8)
31      contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
32      contours = sorted(contours, key=cv2.contourArea, reverse=True)
33
34      # Copy the original image for drawing rectangles
35      image_with_rectangles = image.copy()
36
37      # Create a blank mask to fill polygons
38      mask = np.zeros_like(image)
39
40      # Define the distance threshold to determine proximity for union
41      distance_threshold = 20
42
43      # Utility function to check proximity between two points
44      def is_within_distance(point1, point2):
45          return np.linalg.norm(point1 - point2) <= distance_threshold
46
47      # Function to merge contours representing polygons
48      def merge_polygons(contour1, contour2):
49          merged_contour = np.vstack((contour1, contour2))
50          hull = cv2.convexHull(merged_contour)
51          return hull
52
53      poly_vertices = []
54
55      # Iterate through each contour
56      for i in range(len(contours)):
57          epsilon = 0.01 * cv2.arcLength(contours[i], True)  # Aumentado para reduzir a
    sensibilidade
58          approx = cv2.approxPolyDP(contours[i], epsilon, True)
59
60          # Union adjacent polygons based on proximity
61          for j in range(i + 1, len(contours)):
62              epsilon = 0.01 * cv2.arcLength(contours[j], True)  # Aumentado para
    reduzir a sensibilidade
63              approx_j = cv2.approxPolyDP(contours[j], epsilon, True)
64
65              # Check proximity and merge polygons
66              for point in approx:
67                  if any(is_within_distance(point[0], p[0]) for p in approx_j):
68                      approx = merge_polygons(approx, approx_j)
69                      break
70          poly_vertices.append(approx.tolist())
71          # Draw a red rectangle around the merged polygon
72          x, y, w, h = cv2.boundingRect(approx)
```

```
        cv2.rectangle(image_with_rectangles, (x, y), (x + w, y + h), (0, 0, 255), 2)

    # Save the images
    original_image_path = os.path.join(output_dir, f'original_{os.path.basename(
    img_path)}')
    processed_image_path = os.path.join(output_dir, f'processed_{os.path.basename(
    img_path)}')
    cv2.imwrite(original_image_path, image)
    cv2.imwrite(processed_image_path, image_with_rectangles)

    # Plot the original image and the result with rectangles
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 2, 1)
    plt.title('Original Image')
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.subplot(1, 2, 2)
    plt.title('Contours with Rectangles')
    plt.imshow(cv2.cvtColor(image_with_rectangles, cv2.COLOR_BGR2RGB))
    plt.show()

    return poly_vertices

# Create the Pandas dataset
data_list = []

# Iterate through each file in the 'input_dir' folder
for file in os.listdir(input_dir):
    # Get the full path of the file
    file_path = os.path.join(input_dir, file)

    # Define the output path for the processed image
    output_path = os.path.join(output_dir, file)

    # Call the process_image function
    try:
        poly_vertices = process_image(file_path, output_path)

        # Process the output to extract the desired information
        polyCount = 0
        for poly in poly_vertices:
            verticeCount = 0
            for vertice in poly:
                x = vertice[0][0]
                y = vertice[0][1]

                # Add the data to the Pandas dataset
                data_list.append({'data': file,
```

```
                                    'X': x,
                                    'Y': y,
                                    'poly': polyCount,
                                    'vertice': verticeCount})
                verticeCount += 1
            polyCount += 1

    except Exception as err:
        print(err)

dataset = pd.DataFrame(data_list)

# Display the Pandas dataset
print(dataset)
dataset.to_csv('data/imageData.csv', index=False)
```

Listing A.1 – Computer Vision Algorithm

A.2 Algorithm to Correct Duplicate or Intersecting Regions in FIR Shapefiles

This section details the algorithm developed to correct duplicate or intersecting regions in the FIR shapefile. The algorithm ensures that each FIR sector is uniquely identified and that any nested or intersecting sectors are correctly accounted for in the overall analysis.

```
import geopandas as gpd
from shapely.geometry import Point

# Assume ‘final‘ is the original DataFrame with flight data
# Convert DataFrame to GeoDataFrame
aviation_gdf = gpd.GeoDataFrame(
    final,
    geometry=[Point(xy) for xy in zip(final.lon_pred, final.lat_pred)],
    crs="EPSG:4326"
)

# Load the .shp file of FIR sectors
path_to_fir_shp = '/content/drive/MyDrive/TG/Dados/SetoresFIR/SETOR_FIR.shp'
sectors_fir_gdf = gpd.read_file(path_to_fir_shp)

# Preparation: Ensure all sectors are in the same coordinate system
sectors_fir_gdf = sectors_fir_gdf.to_crs(aviation_gdf.crs)

# Initialize a column to store the subregions of each FIR sector
sectors_fir_gdf['contained_firs'] = None
```

```
21
22  # Check for contained in relation to every other FIR sector
23  for index, fir in sectors_fir_gdf.iterrows():
24      # Create a temporary GeoDataFrame for other sectors, excluding the current sector
25      other_firs = sectors_fir_gdf.drop(index)
26
27      # Identify sectors that are contained within the current sector
28      contained_firs = other_firs[other_firs['geometry'].within(fir['geometry'])]
29
30      # If there are contained sectors, save their identifiers
31      if not contained_firs.empty:
32          sectors_fir_gdf.at[index, 'contained_firs'] = contained_firs['ident'].tolist()
33
34  # Filter sectors that contain other FIR sectors
35  sectors_with_subregions = sectors_fir_gdf.dropna(subset=['contained_firs'])
36
37  # Visualize the results
38  tabela = sectors_with_subregions[['ident', 'contained_firs']]  # 'ident' should be
        replaced with the column name that identifies each FIR sector
39  print(tabela)
40
41  # Build a dictionary mapping child sectors to parents
42  parent_child_map = {}
43  for index, row in tabela.iterrows():
44      for child in row['contained_firs']:
45          parent_child_map[child] = row['ident']
46
47  # Adjust the count to accumulate in the parent sector
48  for child, parent in parent_child_map.items():
49      if parent in count_per_fir.index and child in count_per_fir.index:
50          count_per_fir[parent] += count_per_fir[child]
51          count_per_fir[child] = 0  # Reset child count to zero after adding to parent
52
53  # Filter to remove sectors with zero count
54  count_per_fir_filtered_pred = count_per_fir[count_per_fir > 0]
55  print(count_per_fir_filtered_pred)
```

Listing A.2 – Algorithm to Correct Duplicate or Intersecting Regions in FIR Shapefiles

A.3 Association of Flights and Storm Areas

This section describes the data processing and analysis pipeline used in the study. The methodology was based on the work by Vilela (VILELA, 2023).

```
1  from pyspark.sql import SparkSession
2  import pandas as pd
```

```
import math
from shapely.geometry import Polygon
import matplotlib.pyplot as plt
import os
import numpy as np
import geopandas as gpd
from scipy import interpolate
from pyspark.sql.functions import to_timestamp

print("Setting up Spark session...")
# Initialize Spark session
spark = SparkSession.builder \
    .master('local[*]') \
    .config("spark.driver.memory", "12g") \
    .appName('air-traffic-analysis') \
    .getOrCreate()
spark.conf.set('spark.sql.crossJoin.enabled', 'true')

base = '/content/drive/MyDrive/TG/Dados/'

print("Reading image data...")
# Load image data
image_data_files = [base + 'imageData_1.csv', base + 'imageData_2.csv']
dataset = pd.concat([pd.read_csv(f) for f in image_data_files], ignore_index=True)

print("Reading CAT-62 data...")
# Load CAT-62 data
cat62_files = [base + 'cat-62_part_1.csv', base + 'cat-62_part_2.csv', base + 'cat-62
    _part_3.csv', base + 'cat-62_part_4.csv']
cat62 = pd.concat([pd.read_csv(f) for f in cat62_files], ignore_index=True)

print("Converting and sorting by 'dt_radar'...")
cat62["dt_radar"] = pd.to_datetime(cat62["dt_radar"], unit='ms')
cat62.sort_values(by=['dt_radar'], inplace=True)

print("Applying transformations on lat/lon...")
cat62["lat"] = cat62["lat"].apply(math.degrees)
cat62["lon"] = cat62["lon"].apply(math.degrees)

print("Calculating polygons and centroids...")
dataset['X'] = dataset["X"].apply(lambda x: x + 940)
dataset['Y'] = dataset["Y"].apply(lambda x: x + 740)
polygons = dataset.groupby(['data', 'poly'])[['X', 'Y']].apply(lambda x: Polygon(x.
    values))
centroids = polygons.apply(lambda polygon: polygon.centroid)
areas = polygons.apply(lambda polygon: polygon.area)

```

```
print("Creating results DataFrame...")
results = pd.DataFrame({
    'polygon': polygons.index,
    'centroid_x': [p.x for p in centroids],
    'centroid_y': [p.y for p in centroids],
    'area_estimate': areas.values
})

print("Transforming coordinates...")
def create_transform(pixel_coords, latlon_coords):
    pixel_x, pixel_y = zip(*pixel_coords)
    lat, lon = zip(*latlon_coords)
    lat_interp = interpolate.interp1d(pixel_y, lat, fill_value="extrapolate")
    lon_interp = interpolate.interp1d(pixel_x, lon, fill_value="extrapolate")
    return lat_interp, lon_interp

def convert_pixel_to_latlong(x, y):
    lat_lon_points = [(7.455477, -77.829943), (-51.848798, -76.538633), (1.355664,
    -28.966528), (8.000000, -29.500000)]
    x_y_points = [(940, 740), (940, 2192), (2092, 2192), (2092, 740)]
    lat_interp, lon_interp = create_transform(x_y_points, lat_lon_points)
    return lat_interp(y), lon_interp(x)

print("Converting and saving data...")
results["lat"] = results.apply(lambda x: convert_pixel_to_latlong(x["centroid_x"], x["
    centroid_y"])[1], axis=1)
results["lon"] = results.apply(lambda x: convert_pixel_to_latlong(x["centroid_x"], x["
    centroid_y"])[0], axis=1)
results.to_csv(base + "storm_areas.csv", index=False)

print("Reading data with Spark and preparing SQL queries...")
df_areas = spark.read.csv(base + 'storm_areas.csv', inferSchema=True, header=True)
df_areas.createOrReplaceTempView('df_areas')

cat62_files = [base + 'cat-62_part_1.csv', base + 'cat-62_part_2.csv', base + 'cat-62
    _part_3.csv', base + 'cat-62_part_4.csv']
df_flight = spark.read.csv(cat62_files, inferSchema=True, header=True)

df_flight = df_flight.withColumn("dt_radar", to_timestamp(df_flight["dt_radar"] /
    1000))
df_flight.createOrReplaceTempView('df_flight')

print("Executing SQL query...")
query = """
    SELECT
        f.flightid, f.dt_radar,
    MAX(CASE
```

```
            WHEN (2 * 6371 * ASIN(SQRT(POW(SIN((f.lat - a.lat) * .0174532925 / 2), 2)
                                    + COS(f.lat * .0174532925) * COS(a.lat *
    .0174532925) *
                                    POW(SIN((f.lon - a.lon) * .0174532925 / 2), 2))))
                <= (a.area_estimate / 2) THEN 1
            ELSE 0
        END) as flight_through_area
    FROM (
        SELECT *,
        SUBSTRING(CAST(dt_radar AS STRING), 1, 13) AS rounded_event_time
        FROM df_flight
    ) f
    LEFT JOIN df_areas a
    ON f.rounded_event_time = SUBSTRING(CAST(a.polygon AS STRING), 3, 13)
    GROUP BY f.flightid, f.dt_radar
"""
result_df = spark.sql(query)
result_df.show()

print("Saving results...")
result_df.write.csv(base + "flightThourghtArea2.csv", header=True, mode="overwrite")
print("Processing completed.")
```

Listing A.3 – Data Processing and Analysis Pipeline

A.4 Additional Charts from Exploratory Analysis

This section presents additional charts from the exploratory analysis conducted during the study. These charts provide further insights into the data and support the findings discussed in the main body of the thesis.

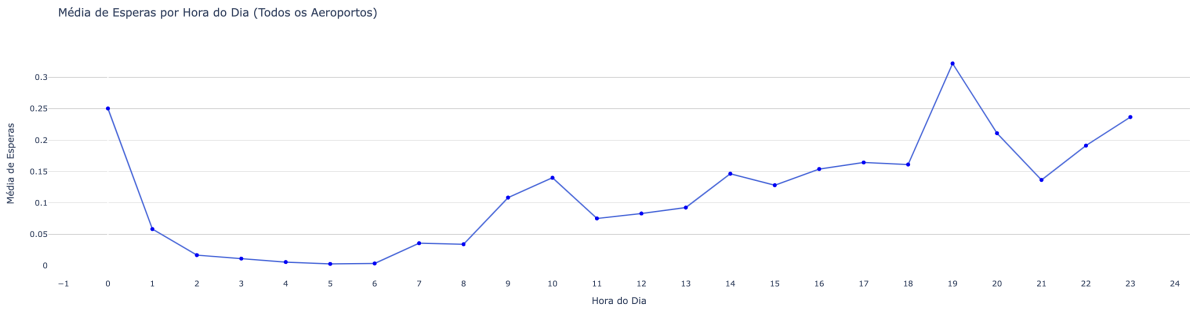

FIGURE A.1 – Wait times by hour of the day

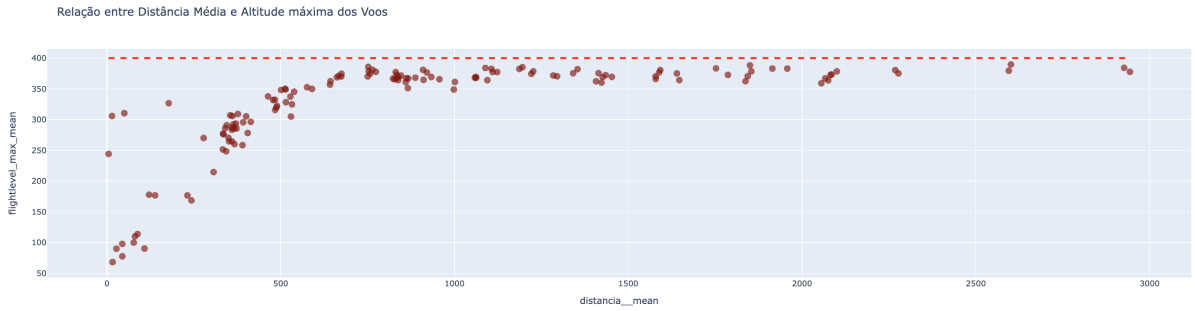

FIGURE A.2 – Mean distance for highest flight traffic times

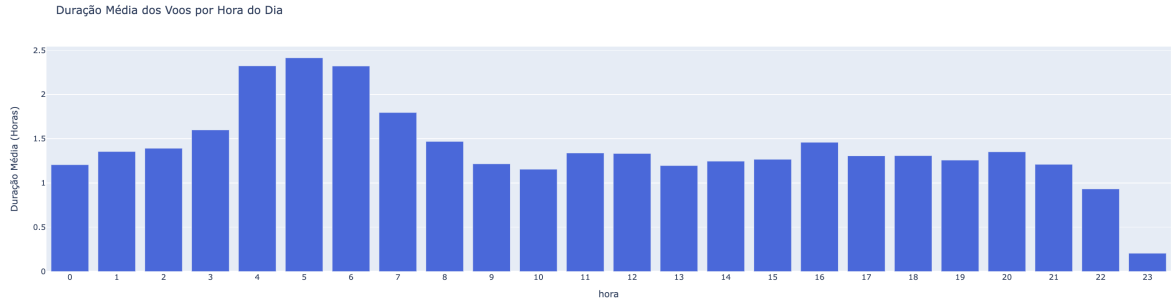

FIGURE A.3 – Mean duration of flights by hour of the day

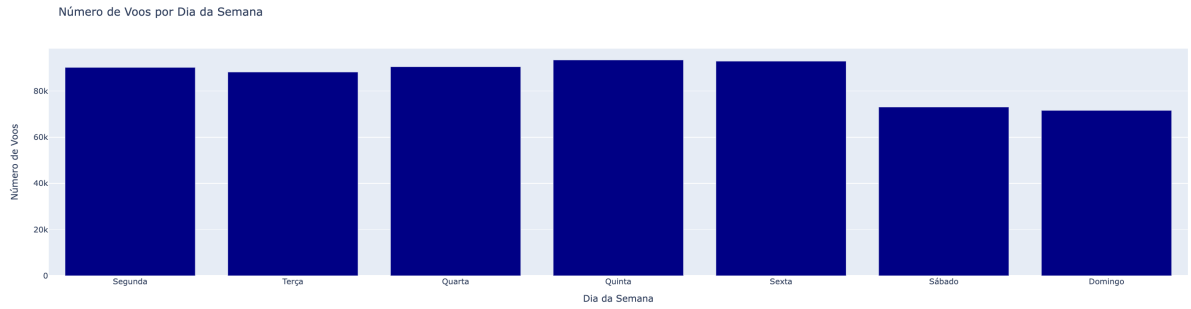

FIGURE A.4 – Number of flights by day of the week

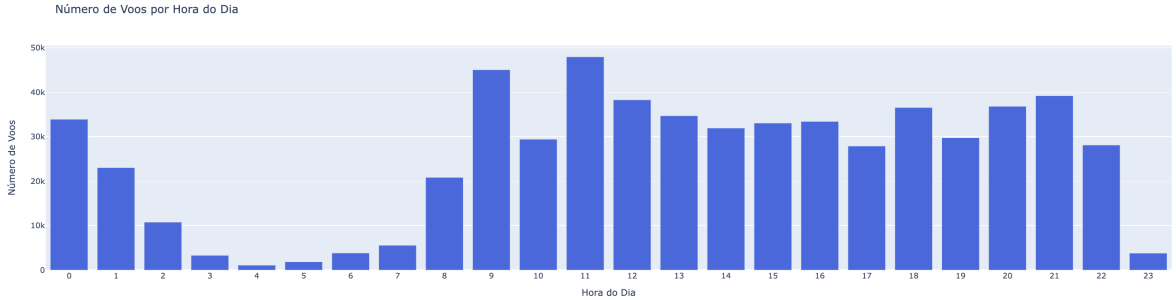

FIGURE A.5 – Number of flights by hour of the day

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
TC	25 de novembro de 2024	DCTA/ITA/DM-018/2015	140

5. TÍTULO E SUBTÍTULO:

Prediction of Air Traffic in Control Sectors using Machine Learning for ATFM Analysis

6. AUTOR(ES):

Victor Hugo de Oliveira Bastos

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Airspace; Air Traffic Flow and Capacity Management; Machine Learning; Flight Trajectory; Computer Vision

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Airspace; Air Traffic Flow and Capacity Management; Machine Learning; Flight Trajectory; Computer Vision

10. APRESENTAÇÃO: (**X**) **Nacional** ( ) **Internacional**

ITA, São José dos Campos. Curso de Graduação em Engenharia Aeroespacial. Orientador: Prof. Dr. José Maria Parente de Oliveira. Defesa em 24/11/2024. Publicada em 25/11/2024.

11. RESUMO:

The effective prediction of air traffic within Flight Information Regions (FIR) is crucial for managing and optimizing air traffic operations. This research leverages computational and machine learning techniques, along with cloud-based resources, to develop a sophisticated pipeline for predicting the number of aircraft flying over these FIR sections, intended for use in ICEA's ATFM Analysis. By integrating multiple datasets, including flight information, weather data, and radar observations, and refining them to perform well in machine learning models, the study aims to enhance the accuracy and reliability of air traffic predictions. The final algorithm developed predicts aircraft positions with an average error of less than 18 km one hour ahead, which leads to the reliability of the regional aircraft counts. This is evidenced by the comparison with a baseline model based on the average speed of the aircraft, which has an average error 2.9 times greater in regional aircraft counts. Potential improvements and limitations have been identified and discussed so that future implementation can be optimized by the ICEA team.

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **SECRETO**