RAFAEL FALECK REJTMAN
RAUL LOSKER

PREDICTING AIRPLANE PILOT PERFORMANCE ERROR FROM GAZE BEHAVIOR DATA

Thesis presented to the Polytechnic School of the University of São Paulo towards obtaining the title of Master in Mechatronics Engineering.

Subject Area:
PMR - Biomecatronichs Laboratory

Superviser:
Arturo Forner Cordero

Co-superviser:
Oron Levin

São Paulo
2020

Acknowledgements

Here we would like to dedicate a few words to thank so many important that, directly or indirectly, made this work possible. Here is our profound thank you to:

- *Professor Arturo Forner and Professor Oron Levin, which made this project possible, supporting us not only with knowledge and excitement but also in starting this whole trajectory a few years ago, in Belgium;*
- *Erich Natsubori Sato, for giving us a fundamental support in Machine Learning and problem-solving;*
- *To Stephanie Brams, and her amazing team: Gal Ziv, Ignace TC Hooge, and Professor Werner F. Helsen, for having started and nurtured this collaboration ever since;*
- *Finally, to thank all friends that helped us with such dedication in our data collection event: Ariana Armijo, Beatriz Biscegli, Bianca Bonugli, Gabriela Rudge, João Ximenes, Marina Tiellet, Otávio Policarpo, Rafaela Nagoshi, Renato dos Santos, Valentina Barrios and Vitor Vieira;*

To my relentless truth-pursuing mother, Jane Ticiana Oliveira Souza, who always inspired a curious and inquisitor spirit in me. To my careful father, Gilberto Losker, who taught me how to honor my responsibilities and work hard to achieve my goals. To my family and friends, who supported me throughout the severe engineering graduation.

Raul Losker

To my loving mother, Paula Kertzmann Faleck Rejtman, which has to lead me to this day with her light, thank you. To my loving father, Sergio Rejtman, whose lessons and passion for science, which shaped me since a child, thank you. To my loving family, which supported me every day through University and beyond, thank you.

Rafael Faleck Rejtman

CONTENTS

PART I

INTRODUCTION

1 INTRODUCTION

1.1 General Overview

Aviation accidents occur every year worldwide, having Pilot Performance Error as their major initiating cause [42]. In the context of flight safety and control, the importance of properly and effectively extracting information from the various cockpit instruments cannot be understated. The field lends itself to exploring gaze behavior and visual perception in the modern aircraft cockpit's complicated environment. This important field of human-machine interaction has been the subject of various remarkable dissertations (which many are in Ziv's review [63]), whose work is a solid foundation for applications that improve flight performance and safety, in this and other works to come. This very thesis's scope is the study of the topic under the light of Machine Learning and the novelties provided by its algorithms.

Figure 1: Modern Airplane Cockpit - Photo by Caleb Woods [56]

This work results from a collaboration between the Biomecatronichs Laboratories of the University of São Paulo (USP) and the Katholieke Universiteit Leuven (KU Leuven) coordinators are both supervisors of the present work. The KU Leuven provided data from experiments done at the CAE Brussels® Flight Academy [5] which were fundamental to this thesis's development. During the experiment, a set of cadet pilots had flight sessions recorded in which a high fidelity device tracked their eye movements. This dataset's details are further presented 6how it was obtained and how it, i.e., used in the section 4.1.

Preliminary analysis of these data suggests the importance of the cockpit instruments' visual scanning patterns. Moreover, previous work is done on this very dataset to evaluate these patterns' execution's quality and success, as described in [15]. Using eye-tracking data and a Machine Learning approach, we intend to develop a predictive model capable of identifying pilot performance errors, to the extent that this might provide help to train and improve the performance pilots.

1.1.1 Goals

This project aims to develop a performance error model based on gaze information. This is to say; the main desired output is a model that could predict Pilot Performance Error as mentioned in section 3, based on gaze behavior and eye-tracking data.

We intend to use the results obtained with this model to predict performance error during flight tasks, which is a major cause for plane accidents, as mentioned in section 2.1 and possibly avoid plane accidents.

Approaching this previously collected data as a time-series variable, gaze-related features can be generated to implement and run a Machine Learning algorithm and obtain a predictive model. The main objectives are as follows:

- Extract and build important features concerning pilot gaze behavior.
- Build a pilot Performance Error model based on time series analysis of multiple in-flight, gaze related variables
- Run and evaluate the model quality and reliability.

1.2 Bibliographic Review

"This review demonstrates that eye-tracking applications have the potential to enhance both training and operational performance [...]"

-- Mackenzie G. Glaholt, DRDC Toronto Research Centre

The eye movements of airplane pilots have been a topic of interest for research for decades [30]. By reviewing some of the most relevant works in this research body, the present dissertation aims to provide a solid basis for this eye-tracking application to predict pilot performance error.

An airplane cockpit instrument display poses a complex visual environment. A cluster of instruments and screens provides a dense visual input to the aviator, containing relevant information about the aircraft's state and functioning of aircraft systems (e.g., movement vector, the status of aircraft systems, and communication systems) [26]. Simple airplanes may contain only six basic flight instruments - i.e., airspeed indicator, altimeter, compass gyroscopic pitch-bank (artificial horizon indicator), direction (directional gyro indicator), and rate of turn indicator - while more modern cockpits typically rely on electronic flight instrument systems (EFIS), which present information electronically, in a screen, rather than electromechanically.

As a pilot continuously monitors these instruments, information must flow smoothly from the cockpit instruments to their brains to enable high awareness of the current flight conditions. In situations of an imminent accident, having a clear perception of the plane's status can be the defining factor in allowing for a fast and decisive response.

In this human-machine interaction, the eyes represent the primary source of input of information. The FAA's Instrument Flying Handbook [21], dedicates a whole section to the instruction of the cockpit instruments' visual scanning patterns, which allows precise control based only on instruments. When explaining the cross-check method, it mentions, "cross-checking is the continuous and logical observation of instruments for attitude and performance information. Due to human error, instrument error, and airplane performance differences in various atmospheric and loading conditions, it is impossible to establish an attitude and have performance remain constant for a long period. These variables make it necessary for the pilot to constantly check the instruments and make appropriate changes in airplane attitude."

In the cockpit, the sequence of eye movements, or so-called gaze behavior, arises then, as a most relevant manner of assessing a pilot's state of perception and, possibly, propensity to commit errors. To this effect, a review of several important eye movements and metrics is presented below:

- Saccades are rapid eye movements that direct the point of gaze to another area of the visual field. The processing of information from the visual field is suspended during saccades (known as saccadic suppression).
- Fixations occur at the end of a saccade when the eyes fixate a point in the visual field, and a relatively stable image is projected onto the retina. It is during these fixation events that visual information is extracted from the fixated area.
- Smooth pursuit eye movements continuously align the point of gaze with a target moving across the visual field (e.g., a passing aircraft).
- Vergence eye movements serve to align the eyes when focusing on different depth planes (e.g., when moving from the instrument panel to an object outside the windscreen).
- Vestibular-ocular movements occur automatically to compensate for movements of the head.
- Dwell time is a typical measure used by researchers to get the total viewing time of a particular area of interest (e.g., altimeter). During tasks such as reading, scene perception, and visual search, fixation durations range from 40 ms to over 800 ms, with mean fixation durations typically falling between 200 ms and 400 ms depending on the viewing task (Rayner [48]). For this dissertation, each area of interest (AOI) will be a cockpit instrument.

1.2.1 Historical Review

By extracting information from eye movement in simulators and real aircraft throughout the decades, researchers have attempted to grasp a pilot's expertise, workload, and situational awareness [26]. In this light, we present a brief review of historical progress in this field.

The first studies of gaze behavior on pilots recall to the post-WWII period [30]. Initial groups of researchers were led by the U.S. Air Force and did not focus directly on gaze but optimal instrument panel arrangements. Results, though, pointed out that the arrangement of the instruments could indeed interfere with gaze behavior. This early work stimulated future research into scanning patterns of instrument panels. In the following years, eye-tracking models started to emerge. The first occurrence was by Senders [59] [58] and attempted to evaluate workload based on instrument scanning. Carbonell [57], proposed a model that assumed a queue of instruments that the pilot would look at in sequence. His model could successfully predict the percentage of fixations of different cockpit instruments. Simultaneously, it could not distinguish different pilots from each other based on their experience(cadet or full pilot).

The 1970s were marked by the development of novel eye-tracking devices and microcomputers. One important debate on the field was whether visual scanning patterns were random or deterministic processes. Dick [40], suggested that the scanning strategy had more correlation with the aircraft state than the position or arrangement of the instruments. He went further to propose a model in which pilots would follow "mini scanning patterns," which combined form a custom "global scanning pattern". Using factor analysis, this model was capable of distinguishing patterns from different pilots. In the same period, Tole et al. [47] found that, by ignoring dwell times within fixations, some scanning patterns could be identified on experienced pilots. This study also suggested an entropy (or disorderliness) metric for scanning patterns and was able to identify an inverse correlation between entropy and mental workload.

In the next decade, Kramer et al. [18] utilized discriminant analysis and both aircraft state and gaze data to create a model capable of distinguishing pilots at different levels of expertise. Hayashi [35] progressed to use a hidden-Marvok Chain model as a pattern recognition method, with good results.

An increase in gaze research distinguished the 2000s focused on applications. Most of these in one of two categories: validating and evaluating the impact of new avionic systems and comparing different proposed avionic systems. Williams [60], Schaudt et al.

[44], and Oseguera-Lohr Nadler (2004) [19] compare attention distribution in flights with an automation systems against flights with standard equipment. In this case, gaze behavior was used as an indicator of attention. A similar approach was used by Schnell et al. [52], Alexander Wickens [17] and Johnson et al. [16] to evaluate the impact of 'Synthetic Vision Systems' (SVS) on pilot attention. Schnell demonstrated positive results regarding overall performance using SVS, while Alexander Wickens and Johnson emphasized visual tunneling as an undesirable side-effect of automation.

Flemisch Onken [41] uses gaze data to compare attention distribution throughout several support devices in a military cockpit. Hayashi et al. [33] compared two different spacecraft interfaces, using gaze behavior to measure attention. An exciting application was made by Wickens et al. [50], which used gaze to determine the impact of visual clutter in pilots' workload. This work showed that a medium level of clutter results in higher performance than low- or high-clutter flight displays.

1.2.2 State of the Art

In the last section, we highlighted gaze behavior applications as a trend for aircraft pilots. In the recent past, rising accessibility and affordability of eye-tracking technology made researchers more interested in using such devices, enabling these new applications (King et al. [31]). This lead to new fields of research using gaze behavior applications.

One example of a new kind of application is railway safety. Madleňák et al. [34] analyzed dwell time, number of fixations, average fixation time, and number of revisits (in a certain region of the cabin) to produce a heatmap of a train driver's attention. The insights generated should help developing future research regarding railway safety, such as upgrading training methods and identifying dangerous pieces of the railway.

Since using eye-tracking in the airport environment already existed, use gaze behavior for air traffic controllers (ATC) is not an absurd idea. McClung Kang [62] were able to identify and map ATCs' visual scan path patterns. Causse et al. [37] propose a very sophisticated real-time decision support system that considers the operator's gaze behavior to suggest decisions for flight decisions.

Li et al. [28] present a gaze behavior method to identify and classify fatigue in construction workers that use heavy equipment. This research, besides acting in favor of safety, correlates fatigue level with productivity and performance.

The advancements of gaze behavior technology are not just limited to new fields of applications. Several other cognitive features could be detected or evaluated using gaze behavior data from the aforementioned workload and attention distribution.

For instance, Hasse et al. [54] conclude that humans supervising automation in teams present i. better performance in detecting failures and ii. “deeper information processing whilst anticipating and detecting relevant events”. In this sense, gaze data was used to understand the process of information acquisition.

Another cognitive feature studied in the recent past is cognitive engagement, defined by Zennifa et al. [32] as “a cognitive process involving decision making, information gathering, visual scanning, and selectively sustaining attention on a specific event while ignoring other external influences”. The authors proceed to use eye-tracking combined with EEG data to recognize cognitive engagement levels.

Nonetheless, Maurois et al. [29] put effort into detecting drowsiness on car drivers. Thought gaze data establishes a model capable of predicting impairment up to 5 minutes before the failure occurs.

Beyond all of that, new methodologies arose in the last ten years. Machine Learning (ML) became a prominent method of analyzing gaze behavior. For instance, Li [28] workers’ fatigue, Naurois drivers’ drowsiness, and Zennifa cognitive engagement detection models use ML to present results. Specifically, the latter uses more than one modeling technique (correlation-based feature selection and nearest neighbor), resulting in a more accurate model.

Finally, Liu et al. [23] used ML to analyze gaze data in a general cognitive task. The researchers also propose Supervised Vector Machine (SVM) as a valid method for classifying workload, as SVM is a very relevant method in the human-computer interaction field.

2 RELEVANCE AND MOTIVATION

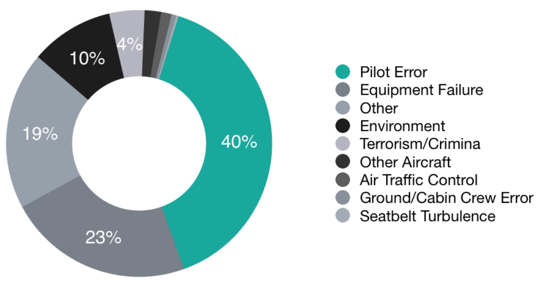
2.1 Relevance

The current commercial aviation scenario is one in which pilot activities revolve more and more around monitoring than active control of the plane [38]. This requires constant and reliable situational awareness of the cockpit's various elements. The pilot is largely responsible for identifying potential failures in instruments and threats that cannot be autonomously detected by the automated flight systems.

In this context, visual attention is of remarkable importance and represents a major contributing factor to aviation accidents. When examining the causes of major international flight accidents over a 15 year period (from 1990 to 2006), Oster and colleagues [42] identified "Pilot Performance Error" as the initiating cause - i.e., the cause that initiated the sequence of events that culminated in the accident - in around 40% of the flight accidents analyzed, representing the most frequent cause of fatal accidents.

Figure 2: Percentage of aviation accidents by initiating cause. Pilot Performance Error is highlighted, representing the largest part. (Source: Author)

Cause of Accident	Fatal Accidents	
	Number	Share
Pilot Error	278	40%
Equipment Failure	158	23%
Other	134	19%
Environment	71	10%
Terrorism/Crimina	30	4%
Other Aircraft	14	2%
Air Traffic Control	9	1%
Ground/Cabin Crew Error	4	1%
Seatbelt Turbulence	2	0%



Furthermore, the authors attempted to break down Pilot Performance Errors into specific types, coming up with the following categories: flying skills, unstabilized approach, controlled flight into terrain, in-flight judgment, on-ground judgment, fuel management, and cause ambiguous.

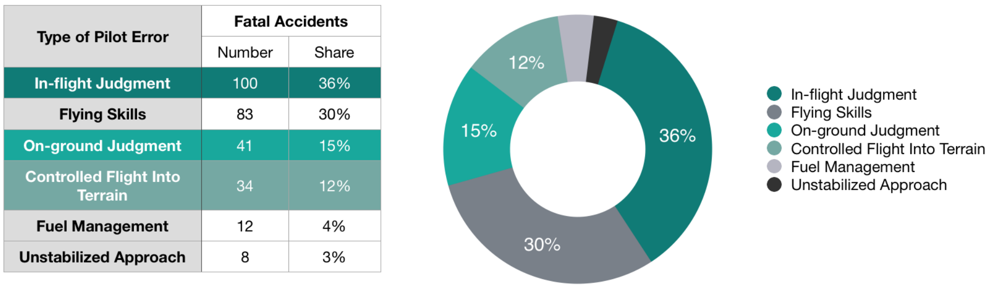
For this dissertation, it is relevant to identify which of these categories encompass visual attention causes. To that extent, the paper leads to considering the *pilot judgment* and *controlled flight into terrain* as the most relevant ones. As the authors say:

> "It is striking that over half of Pilot Performance Error accidents (involving well over half of Pilot Performance Error caused fatalities) are due to errors in pilot judgment, either in-flight (36 percent) or on-ground (15 percent), rather than the pilot losing control of the aircraft. Such errors include things like continuing visual flight rules (VFR) flight into instrument meteorological conditions (IMC) [...]" [42]

And, also:

> "Accidents involving controlled flight into terrain (12%) can also be difficult to assess. The notion behind such an accident is that the pilot may become distracted or lose situational awareness and simply fly the plane into the ground, often in level flight into rising terrain". [42]

Figure 3: Percentage of Pilot Performance Error accidents by type of Pilot Performance Error. Visual attention is considered to be a contributing effect within the highlighted categories in the table. (Source: Author).



Type of Pilot Error	Fatal Accidents	
	Number	Share
In-flight Judgment	100	36%
Flying Skills	83	30%
On-ground Judgment	41	15%
Controlled Flight Into Terrain	34	12%
Fuel Management	12	4%
Unstabilized Approach	8	3%

2.2 Motivation

As technology in aircraft control progressed, over-reliance on automation has been a growing concern in the industry. Pilots can present a poor performance while supervising an automated system. That is known as the Out-Of-The-Loop (OOTL) Performance Problem, described by Endsley and Kiris [39]. Some remarkable recent aviation history accidents - i.e., the Boeing 737 MAX crashes in Indonesia and Ethiopia in 2018 and 2019 - were largely associated with the pilots' lack of skills in understanding and controlling the planes systems. As mentioned in [38].

> "In October, a Lion Air jet crashed in Indonesia, killing 189 people. Investigators now think the pilots struggled to control the Boeing aircraft after its automated systems malfunctioned, in part because they did not fully understand how the automation worked. The authorities are investigating what caused Sunday's crash of the same model jet in Ethiopia, in which 157 people died".

While automation has played a major role in reducing aviation accidents over the years, it has certainly contributed negatively to the confidence in a pilot's ability to fly a plane properly. Many have become overdependent on technology. As former Delta Airlines pilot Kevin Hiatt put in an interview with the New York Times article, *"[...] automation in the aircraft, whether it is a Boeing or an Airbus, has lulled us into a sense of security and safety [38]"*.

Specifically, plane accidents can be directly traced to gaze-related issues, such as the 2009 Continental Connection Flight 3407 (Colgan Air) crash. Here, one of the crash's contributing factors was "the flight crew's failure to monitor airspeed about the rising position of the low-speed cue" [14] as concluded in research by The National Transportation Safety Board of the USA.

In many cases, the problem is further aggravated by the ever-increasing complexity of avionics systems and modern airplane cockpits. "[Pilots] *may not exactly know or recognize quickly enough what is happening to the aircraft, and by the time they figure it out, it may be too late,* "said Hiatt later in the same interview. This is exactly what happened to the AF 447 flight from Rio de Janeiro to Paris. The pilots were not aware of the aircraft getting into a "stall" situation before it was, indeed, too late. Although not all "recognition" problems are directly correlated to gaze and visual attention, they are certainly increased by inadequate scanning of the cockpit instruments during flight.

It is notable that even in modern times when advances in automatic control have greatly changed aviation, it is still imperative that certified pilots can fully control an aircraft. Bendheim [12] mentions, in his study, *"degradation of the operator's skills, the difficulty of understanding automation because of increasing complexity, and 'automation complacency,' or the over-reliance and delegation of authority to automatic systems [...] are problematic because, in many cases, automation is not yet fully autonomous, and, ironically, require increased human engagement"*.

In this light, this dissertation aims to develop a machine learning tool that, based on monitoring of gaze behavior, could assist the training of pilots or alarm pilots that their performance may be jeopardized due to poor scanning of cockpit instruments. The authors hope that in developing such a tool, better training and easier evaluation of pilot's before and during flight may lead to fewer aviation accidents by making sure that, regardless of automation, pilots are well capable of coping with the complicated environment of a modern plane and flying it safely.

PART II

METHODOLOGY

As reviewed in previous sections, several methods were proposed and successfully used by researchers to create gaze behavior models for airplane pilots. From Markov-chains to statistical factor analysis, these methods differed in their approach to the problem, and, perhaps most importantly, in their scope - or variables of interest. A gaze behavior model can be created to predict different target variables, such as eye movement characteristics - e.g., dwell time - or related to pilot cognitive state, like mental workload. The present dissertation attempts to create a model to predict what we name pilot performance error (as defined in section 3).

To this objective, by tracking pilot gaze behavior, we intend to create a model that detects, and potentially predicts, pilot performance error (view definition at 3). The gaze behavior data is captured during the execution of predefined flight tasks in a simulator. We propose to gather enough data to apply a Machine Learning toolset to create the model. The proposed method is described in detail in the following sections.

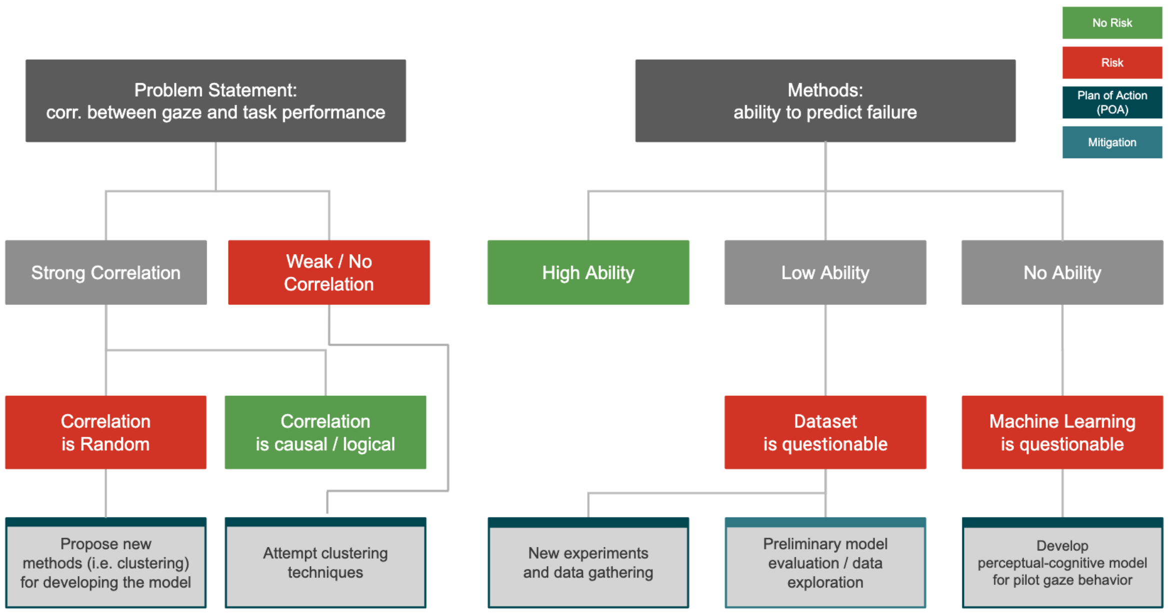
Machine Learning Approach

As presented in the State of the Art section 1.2.2, advanced analytics methods have shown their first appearance in the Gaze Behaviour area of research in the last ten years. Some have shown impressive success. This work was inspired by the fruitful works and their potential to demonstrate in Machine Learning (ML) methods to create gaze behavior models.

Looking to acquire some more confidence in ML as a method to answer our problem question and uncover signals relating gaze behavior to flight performance error, a suggestion was to, early in the project, develop a very rudimentary ML model as validation. This model was created and evaluated with results that enforced the current method proposition, at least, in some ways. The results of this model are shown in the first part of the Results section.

As a third tool to support the methodology choice, risk analysis was carried out considering the use of Machine Learning as a method:

Figure 4: Risk Analysis for Machine Learning Methodology (Source: Author).


No Risk
Risk
Plan of Action (POA)
Mitigation
Problem Statement: corr. between gaze and task performance
Methods: ability to predict failure
Strong Correlation
Weak / No Correlation
High Ability
Low Ability
No Ability
Correlation is Random
Correlation is causal / logical
Dataset is questionable
Machine Learning is questionable
Propose new methods (i.e. clustering) for developing the model
Attempt clustering techniques
New experiments and data gathering
Preliminary model evaluation / data exploration
Develop perceptual-cognitive model for pilot gaze behavior

3 DEFINITIONS

Pilot Performance Error: a deviation of plane metric (e.g., altitude, heading) from the target, in measurable flight parameters. A quantified "deviation" value will be presented in results, as various options will be tested to tune the model. As for the flight parameters considered, these were assumed as the same parameters originally measured in the simulator (a list is presented in Table 2).

Flight Instruments: any of the instruments contained in the flight instruments panel used in the described experiment. These are:

- **Attitude Direction Indicator (ADI):** the attitude indicator, with its miniature aircraft and horizon bar, displays a Figure of the aircraft's attitude. The miniature aircraft's relationship to the horizon bar is the same as the real aircraft's relationship to the actual horizon. The instrument gives an instantaneous indication of even the smallest changes in attitude.

- **Altimeter(ALT):** the altimeter is an instrument that measures the height of an aircraft above a given pressure level. Since the altimeter is the only instrument capable of indicating altitude, this is one of the most vital instruments installed in the aircraft.

- **Vertical Speed Indicator (VSI):** The VSI, also called a vertical velocity indicator (VVI), indicates whether the aircraft is ascending, descending, or in level flight. The rate of ascension or descent is indicated in feet per minute (fpm). If properly calibrated, the VSI indicates zero in level flight.

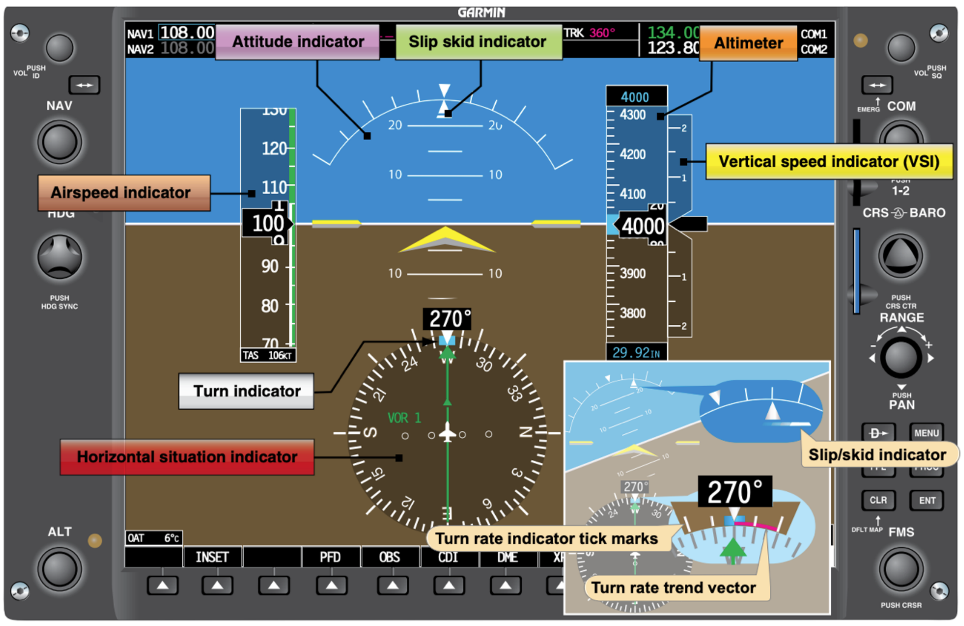
- **Airspeed Indicator (ASI):** the ASI is a sensitive, differential pressure gauge that measures and promptly indicates the difference between pitot (impact/dynamic pressure) and static pressure. These two pressures are equal when the aircraft is parked on the ground in calm air. When the aircraft moves through the air, the pressure on the Pitot line becomes greater than the pressure in the static lines.

This pressure difference is registered by the airspeed pointer on the instrument's face, which is calibrated in miles per hour, knots (nautical miles per hour), or both.

- **Heading Indicator (HDG):** the heading indicator is fundamentally a mechanical instrument designed to facilitate the use of the magnetic compass. In the magnetic compass, errors are numerous, making straight flight and precision turns to headings difficult to accomplish, particularly in turbulent air. However, a heading indicator is not affected by the forces that make the magnetic compass difficult to interpret.
- **Turn and Slip Indicator (BA):** the turn-and-slip indicator uses a pointer, called the turn needle, to show the direction and rate of turn, in degrees per second.
- **Tachometer (PWR):** the tachometer is a device for counting. It is used to show the number of revolutions per minute (RPM) of the aircraft engine. An airplane needs one tachometer for each of its engines.

**Instrument descriptions taken from [21] and [22].*

Figure 5: Primary flight display (PFD). The actual location of indications vary depending on manufacturers. Taken from [21] page 8-12



.

AOI: Area of Interest. In the case of this dissertation is used about any Flight Instrument.

Abbreviations	
Gaze Data Table	GDT
Raw Data Table	RDT
Performance Error Instant	PEI
Master Table	MT
Reduced Master Table	RMT

4 METHOD STEPS

The proposed method can be broken into three major steps, as follows:

1. **Data Gathering:** extracting data from an experiment.
2. **Data Processing:** preparing data for a learning framework (master table formatting).
3. **Machine Learning Modelling:** implementing Machine Learning methods and testing.

This division considers the steps needed to obtain a functional model within the defined requirements. Further development steps on this work could be implemented in case a successful model is obtained:

4. Testing in Simulator:
   (a) Post-flight
   (b) Real-time

4.1 Data Gathering

As a raw data source, a collection of gaze related data from an experiment detailed in [15] is suggested. This data was provided in good faith to the authors as a contribution from the Faculty of Kinesiology and Rehabilitation Science of the Katholieke Universiteit Leuven, whose member, Dr. Oron Levin, is a supervisor in the present work.

A brief description of the experiment realized to obtain the dataset is as follows: The experiment involved 32 cadet pilots from the CAE Brussels® [5] Flight Academy, subject to a pair of instructed flights in a basic simulator. Each flight lasted approximately 30 min. and consisted of a similar sequence of flying tasks (i.e., turning, climbing and descent, straight and level flight) that were instructed via automated voice command to each cadet. During this experiment, the cadets were equipped with a Tobii [6] Pro Glasses 2® wearable eye tracker (details at [1]) that recorded their pupil movement over a Garmin G1000® [51] electronic instrument panel. Their flight performance was also being evaluated throughout the task via the simulator's internal computer. 22 pilots had their performance recorded. Participants provided written informed consent, and ethical approval was provided by the local university (KU Leuven) ethics committee (G-201504218).

The experiment was designed to evaluate the impact - in gaze behavior - of a particular flight course given to the pilots. So, it followed a pre-post test format with control and trial groups. This meant the cadets followed the above procedure twice, once in a pre-course flight and the second one in a post-course flight. This does not pose any particular risk or disadvantage to the model to the present paper's extent. The approach to be described, in theory, agnostic to expertise level. In that sense, having two experiments per pilot would only contribute with more data points exemplifying "success" and "failure" with the associated gaze data. More details regarding the set-up of the experiment can be obtained in Brams'work [15]. Therefore we initially had gaze data regarding 64 flight sessions. Considering the performance recordings, we have the performance reports of 44 flight sessions. Although, only 42 of these flight sessions had a match between gaze data and performance reports.

Two main data outputs from this experiment are useful to this paper:

4.1.1 Gaze Data

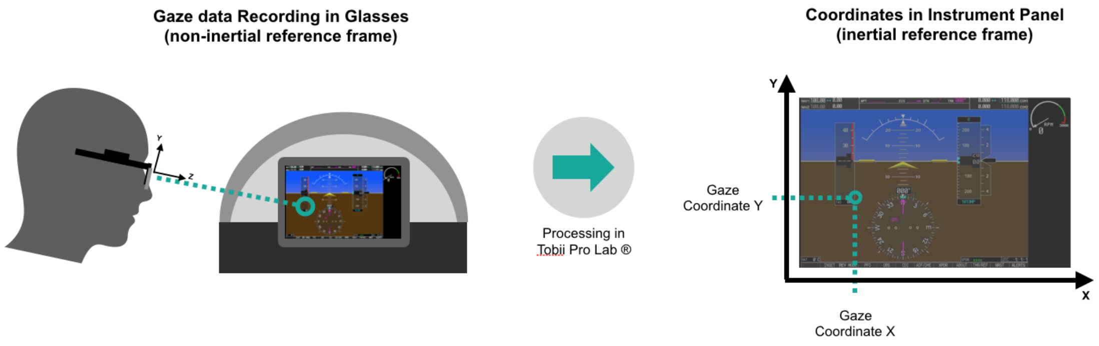
Data was gathered from the eye-tracking device at a 50Hz rate and stored in a memory unit plugged into the glasses. The gaze data was transferred to a computer and processed via a proprietary Tobii [6]® software. This was done with a few objectives:

- To convert gaze coordinates from a non-inertial reference frame - the cadet's head - to pixel coordinates in a static snapshot of the instrument panel. This complex process was done by an automatic routine in the Tobii [6]® software. The routine associates a confidence level - named "AutoMap Score" - to each pair of mapped coordinates, exported together with the gaze coordinates, to allow filtering down the line.

- To isolate fixation coordinates and remove saccadic recorded data (that is, remove "eyes not found" and "unclassified" data, which has ambiguous interpretations concerning data, as explained in section 6.1.3)

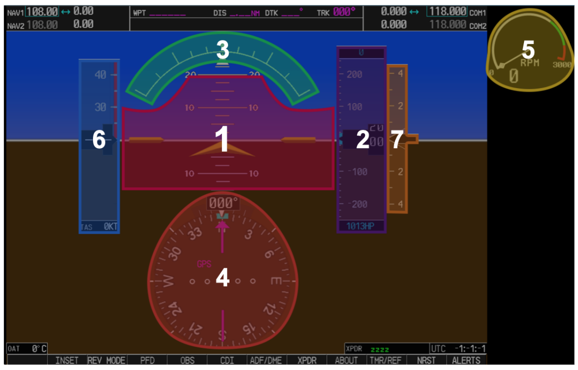
A schematic arrangement of the experiment is demonstrated below:

Figure 6: Expteriment arrangement from data colletion to pre-processing in Tobii [6] Pro Lab® (Source: Author)


Gaze data Recording in Glasses
(non-inertial reference frame)
Y
Z
Processing in
Tobii Pro Lab ®
Coordinates in Instrument Panel
(inertial reference frame)
Y
Gaze
Coordinate Y
X
Gaze
Coordinate X

After processing in Tobii [6] Pro Lab®, the data were exported as a table containing a time series of gaze coordinates and their mapping confidence. A post-processing algorithm was used in Matlab® to convert each pair of gaze coordinates into Area of Interest (AOI) hits - i.e., which instrument in the panel was being observed at any point. In this process, the following AOIs were considered, representing each instrument in the panel:

Figure 7: Instrument Panel AOIs (based on Garmin G1000® [51]



. Source: Author)

A simple numeric encoding was used for each instrument, as follows:

Table 1: Encoding of panel instruments (Source: Author).

ADI	1
ALT	2
BA	3
HDG	4
PWR	5
SPEED	6
VS	7

This process was repeated for each flight recording.

Finally, a Gaze Data Table could be exported, containing a unique identifier for the flight recording, a time-series of pairs of gaze coordinates, their respective AOI hit, and mapping confidence score. A snapshot of a few rows of this table is presented below:

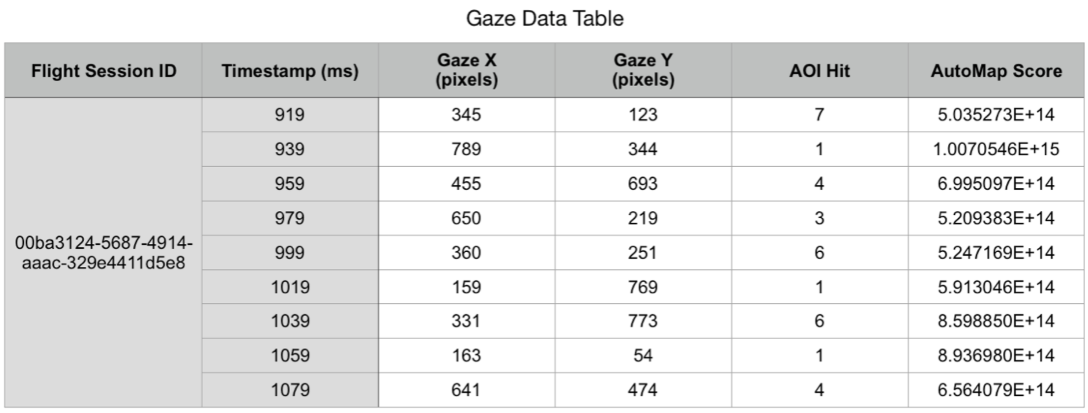
4.1.2 Simulator Performance Reports

As the cadets flew in the simulator, flight parameters (e.g., airspeed, altitude) were recorded and stored. A post-processing software was used to plot these variables (as well as the desired target) over time and generate PDF reports. These reports were made available as well. The tracked performance metrics are as follows:

A snapshot of two tasks and their performance plots is presented below:

As these reports were only available in PDF file format, a process of "translating" this data into numeric, usable data is necessary. To extract tables from the plotted PDF graphs, some computational approaches are attempted as described in the section

Figure 8: Snapshot of Gaze Data Table (GDT).(Source: Author).



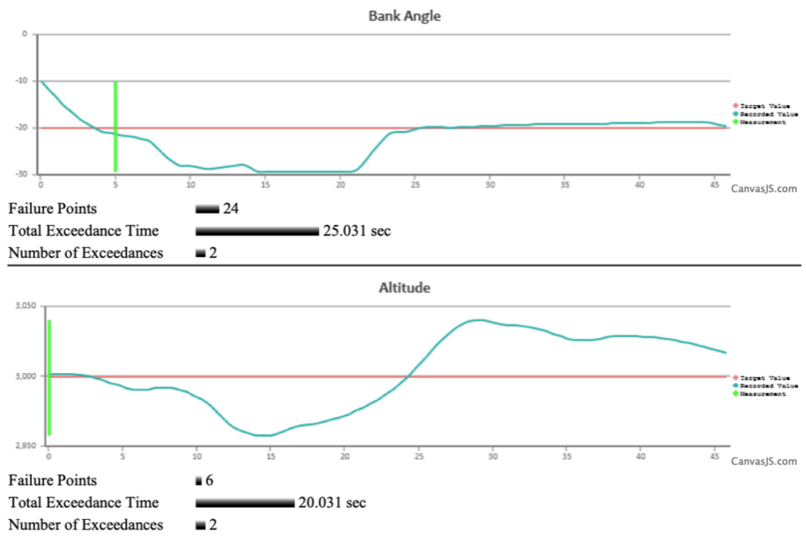
Gaze Data Table

Flight Session ID	Timestamp (ms)	Gaze X (pixels)	Gaze Y (pixels)	AOI Hit	AutoMap Score
00ba3124-5687-4914-aaac-329e4411d5e8	919	345	123	7	5.035273E+14
	939	789	344	1	1.0070546E+15
	959	455	693	4	6.995097E+14
	979	650	219	3	5.209383E+14
	999	360	251	6	5.247169E+14
	1019	159	769	1	5.913046E+14
	1039	331	773	6	8.598850E+14
	1059	163	54	1	8.936980E+14
	1079	641	474	4	6.564079E+14

Table 2: Flight Performance Metrics tracked in the Simulator. (Source: Author).

Metric	Unit
Bank Angle	degrees
Altitude	feet
Airspeed	knots per hour
Heading	degrees
Vertical speed	feet per minute

Figure 9: Simulator Performance Report Snapshot. (Source: Author).



Failure Points	24
Total Exceedance Time	25.031 sec
Number of Exceedances	2

Failure Points	6
Total Exceedance Time	20.031 sec
Number of Exceedances	2

5, weighing the potential losses and uncertainties presented by each. A description of the chosen method is presented in the section.

4.2 Data Processing

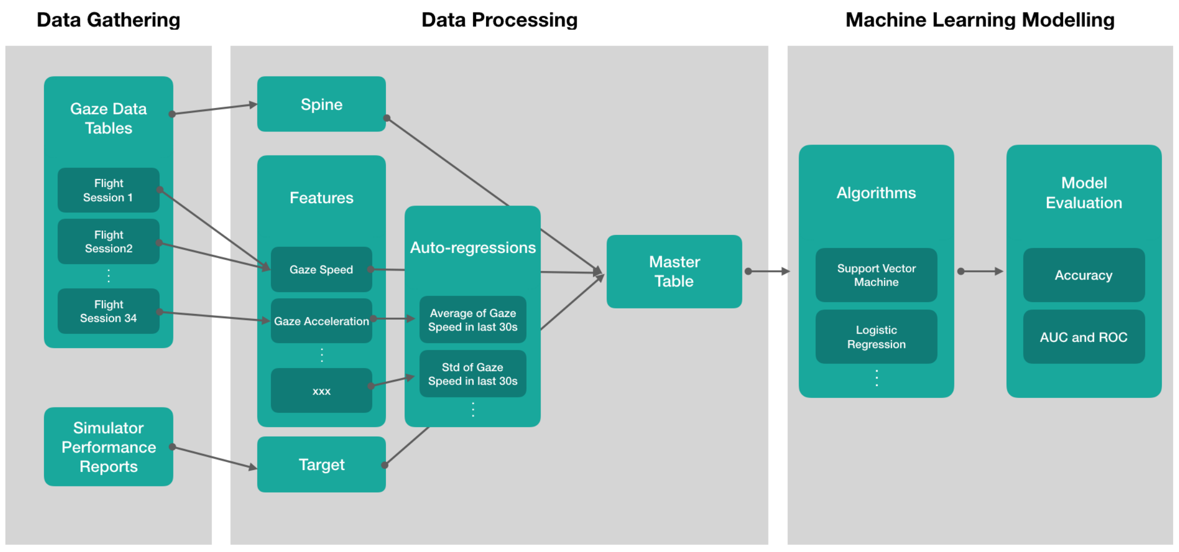
Although this step is presented before the Machine Learning section, the data processing is done based on a few key notions (some related to ML) about the problem trying to be solved. Essentially, these are:

- That this is understood as a classification problem focused on classifying a certain state of gaze as one which will cause performance error or not
- That the recorded data is labeled in terms of flight performance, which suggest the use of supervised Machine Learning [36]

4.2.1 General Framework

This step is designed to structure the data adequately to apply Machine Learning techniques. In general, the intent is to arrive at a so-called master table that contains three structural items: a spine (unique identifiers of the data), a set of features (bulk data to be used for learning), and a target feature (the information trying to be modeled). An overview of the framework to get to this stage is presented below:

Figure 10: Graph of General ML Framework. (Source: Author).



Spine

The spine is a set of columns that uniquely identifies a row in the master table. The spine reflects the granularity in which the data is available and that the model will be trained in. An important aspect of the spine is that, once it is defined, all of the features and the target must have each row's values. In the next section's modeling approach, the spine used for this problem will be detailed.

Features

The features are the core information onto which the model will "learn." They constitute a set of columns containing data deemed relevant to signal detection when trying to model the target. Features can be separated into two main types: Auto-regressive: these are features created by making aggregate calculations on past values of regular features. They are mostly calculated using a so-called moving window method, further detailed in the results section 7. Regular: these features are mostly simple calculations associated with the field of research at hand. In this work, these would include gaze related metrics (e.g., gaze speed, acceleration, dwell times, number of fixations, etc.). More sophisticated features could include normalization (based on population averages or other) and grouping (e.g., per instrument in the panel).

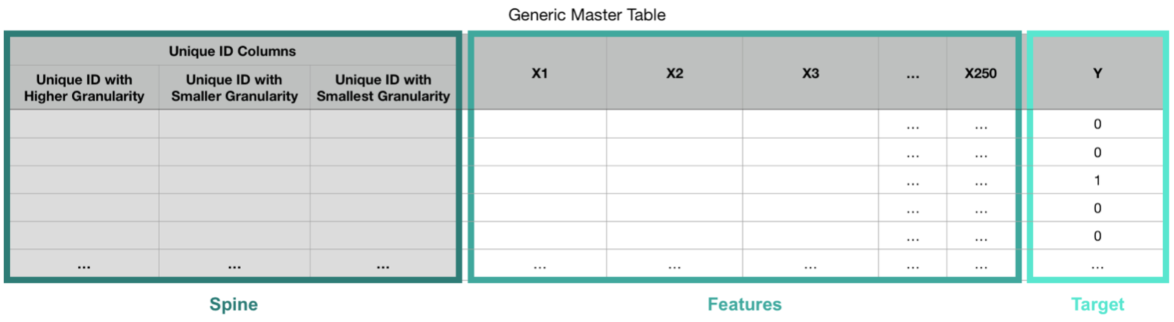
Target

In a supervised classification problem, the target is a single column containing the information or phenomena intended to be modeled. In the case of this problem, this could be an indication of Pilot Performance Error (view definition at 4.2.4) over the course of the flight. Binary targets are ones that can assume only two values (True or False). Simultaneously, multi-classification problems work with non-binary targets, assuming any number of values, one per class.

Master Table

The master table is the concatenation of the Spine, the Features, and the Target columns. A generic master table would have the following format:

Figure 11: Generic Master Table Structure with Spine, Features and Target. (Source: Author).


Generic Master Table
Spine
Features
Target

Unique ID Columns								
Unique ID with Higher Granularity	Unique ID with Smaller Granularity	Unique ID with Smallest Granularity	X1	X2	X3	...	X250	Y
						...	...	0
						...	...	0
						...	...	1
						...	...	0
						...	...	0
...	...	...	...	...	...	...	...	...

4.2.2 Experimental Data

As mentioned previously, the available Gaze Data Table (GDT) is essentially a time-series of gaze coordinates for the duration of the recorded flight sessions. It is also known that each of these flight sessions included a defined set of flight tasks (i.e., turning, climbing and descending, etc.) instructed to the cadets. A list of possible flight tasks instructed to a cadet is as follows:

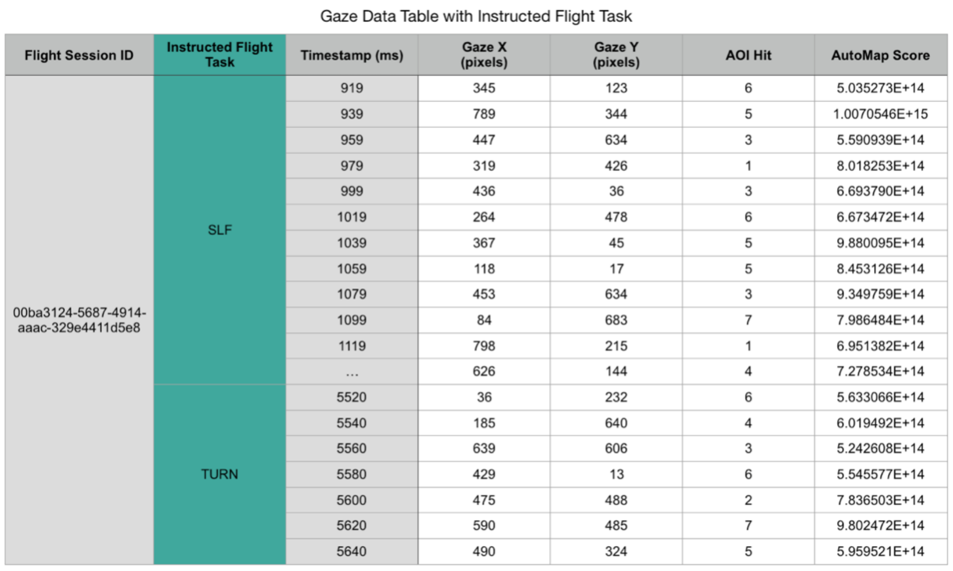
Table 3: All possible Instructed Flight Tasks in the Simulator. (Source: Author).

Instructed Flight Task	**Description**
SLF	Straight and level flight
TURN	Coordinated turning maneuver
ROLL-OUT	Getting out of turn to return to straight flight
VERT	Ascent or descent maneuver
COMBO	Vertical maneuver coupled with turn maneuver
RO+LO	Getting out of combined maneuver to straight and level flight

The GDT can be broken into time slices, where each flight task was supposed to be followed. Creating a column with the current instructed Flight Task would enrich the GDT, as follows:

This table is the raw basis for creating a master table in the next section's method. For ease of notation, this will be named Raw Data Table, or RDT for short.

Figure 12: Raw Data Table. (Source: Author).



Gaze Data Table with Instructed Flight Task

Flight Session ID	Instructed Flight Task	Timestamp (ms)	Gaze X (pixels)	Gaze Y (pixels)	AOI Hit	AutoMap Score
00ba3124-5687-4914-aaac-329e4411d5e8	SLF	919	345	123	6	5.035273E+14
		939	789	344	5	1.0070546E+15
		959	447	634	3	5.590939E+14
		979	319	426	1	8.018253E+14
		999	436	36	3	6.693790E+14
		1019	264	478	6	6.673472E+14
		1039	367	45	5	9.880095E+14
		1059	118	17	5	8.453126E+14
		1079	453	634	3	9.349759E+14
		1099	84	683	7	7.986484E+14
		1119	798	215	1	6.951382E+14
		...	626	144	4	7.278534E+14
	TURN	5520	36	232	6	5.633066E+14
		5540	185	640	4	6.019492E+14
		5560	639	606	3	5.242608E+14
		5580	429	13	6	5.545577E+14
		5600	475	488	2	7.836503E+14
		5620	590	485	7	9.802472E+14
		5640	490	324	5	5.959521E+14

4.2.3 Preliminary Model Framework

At the initial phases of the project, an important task was demonstrating the potential of Machine Learning as a solution for the proposed problem. In addition to a review of similar attempts by other researchers, presented in the section 1.2.2, a secondary form of validation was proposed: a rudimentary Preliminary ML Model.

The main objective of this model is to demonstrate the correlation between gaze-related data and flight performance error. If the model shows some correlation, we will build the complete GAPE model, detailed in the section 4.2.4. Its results are presented in the annex B.

The requirements for this model are as follows:

1. "Concede" the minimum to the algorithms whenever possible. That is, offer as few resources as possible to the ML algorithms to achieve its objective.
2. Be developed in less than 30 work-hours (to reflect a proposed week of work with 6h per day of dedicated work)
3. Yield results in the same format of the GAPE model, a time series prediction of pilot performance error.

For this purpose, a quicker method - than the GAPE model described in the next section 4.2.4 - was used. This method is briefly described here.

In general, this method follows the General Framework presented in the Data Processing section of Methodology. This means it aims to create a Master Table with a Spine, Features, and a Target. A brief description of each is presented as follows.

Spine

For this model, the idea was to create a Master Table with less granularity, as in having a Spine composed of Flight Session ID and Pilot Instructed Task only - and not the Timestamp column. This means any row of data in this Master Table is an aggregate of data from a whole Instructed Flight Task instead of data from each timestamp within that Instructed Flight Task.

Feature (Gaze Duration of Most Important Instrument)

Here, in an attempt to follow item 2 of the requirements, a single feature is proposed. This contrasts with the hundreds of features typically used in such modeling, as explained in the section 4.2.4.

The proposed single feature is the total gaze time (in milliseconds) of what is considered the "most relevant" instrument in each Instructed Flight Task. The definition of a most relevant instrument per task is complex in itself. In this approach, general knowledge of aviation and inputs from consulted flight experts [15], was used to define each Flight Task's correlation to an instrument that is the "most important information source" that task. This correlation is presented as follows:

Table 4: Most Important Instrument for Information Gathering by Flight Task. (Source: Author).

Flight Task	Most Important Instrument
SLF	ADI
TURN	BA
ROLL-OUT	BA
VERT	ALT
COMBO	BA
RO+LO	ALT

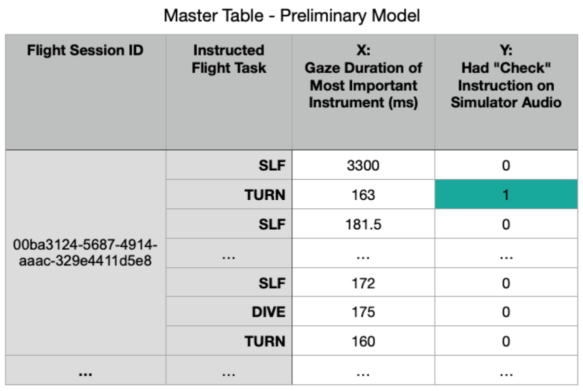
Target

This target is a simple classification vector that flags each Flight Task with a zero (no error) or one (performance error) during that task. For this preliminary methodology, the presence of a performance error was determined using the audio recordings of the flight sessions. Basically, these audio files contain recorded commands from the simulator computer that presents to be the next task for the pilots. Whenever a Pilot strays from the intended performance target, the simulator speaks a "Check" command that indicates an error was present. These commands were listened for, and this was used to create the target variable.

Master Table

With the aforementioned Spine, Feature and Target, a three-column master table is created. A visual representation of this table is shown in Figure 13:

Figure 13: Master Table for Preliminary Model. (Source: Author).



Master Table - Preliminary Model

Flight Session ID	Instructed Flight Task	X: Gaze Duration of Most Important Instrument (ms)	Y: Had "Check" Instruction on Simulator Audio
00ba3124-5687-4914-aaac-329e4411d5e8	SLF	3300	0
	TURN	163	1
	SLF	181.5	0
	...	...	...
	SLF	172	0
	DIVE	175	0
	TURN	160	0
...	...	...	...

4.2.4 GAPE Framework

Abbreviation

GAPE: Gaze based Performance Error model

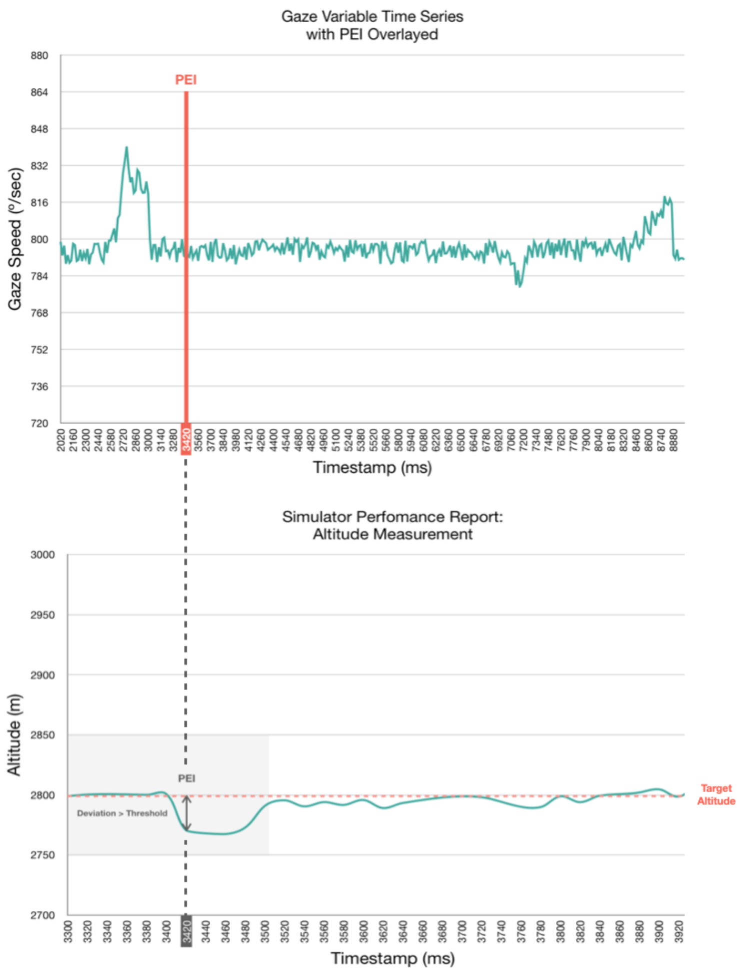
At the core of this methodology is examining gaze data as a time-series of individual fixation events that simultaneously execute a flying task with measurable performance. Gaze and performance events occur in the same timeframe. Thus, it is reasonable to propose an approach to detect variations of gaze data at a certain moment that could indicate a possible performance error at the following instant.

The Target: Performance Error Events

The first step to this method is observing Performance Error (view definition at section 3) as an event that occurs at a given instant in time. This instant will be named Performance Error Instant (PEI). In this light, any variable created from gaze data (i.e., a Feature) could be carefully observed before the PEI in an attempt to identify the PEI. This would be visually represented as follows (Gaze Speed is an arbitrary example of a feature, any other, regular or auto-regressive could be used equally):

An important question then arises: how to precisely define the PEI in time for the

Figure 14: Gaze Variable Time Series with highlighted PEI (above). Matching Performance Report and PEI detection during Straight and Level Flight (below) (Source: Author).



GAPE model. The approach, in this case, is to define a threshold of deviation in the performance metric. The PEI is flagged when the performance metric exceeds this threshold. For this, the data provided by the Simulator Performance Reports is the source. An important caveat at this point is that due to the nature of these performance reports - being PDF files - the translating process introduces errors and greatly limits the time resolution to define these PEIs adequately (ideally, the accuracy needed would be of milliseconds). The value of the used time resolution will be presented in the Results section. The aforementioned method is illustrated as well in Figure 14.

Note

As we went further into the work development, developing a model Target defined by PEIs was not feasible due to matching the Gaze Data's performance reports. This is thoroughly explained in the section 5. Also, the approach to make a new, simpler Target is presented in that section. Although we had to follow a backup plan in our case, we still consider the PEI version as a more robust solution, if it can be used.

These Performance Error events and their associated PEIs are the basis of what will become the target feature for this method, as presented further on. A further observation is that Performance Errors can be detected for each of the measured performance metrics presented in Table 2. At this point, each of these errors can be considered a different class in our target feature in the following manner:

Table 5: Target in a Multi-classification Problem Framing. (Source: Author).

Flight Performance Error	**Target (Y)**
Bank Angle Performance Error	1
Altitude Performance Error	2
etc.	

This would frame the scenario as a multi-classification problem, but we suggest a different approach to create target features in the following section.

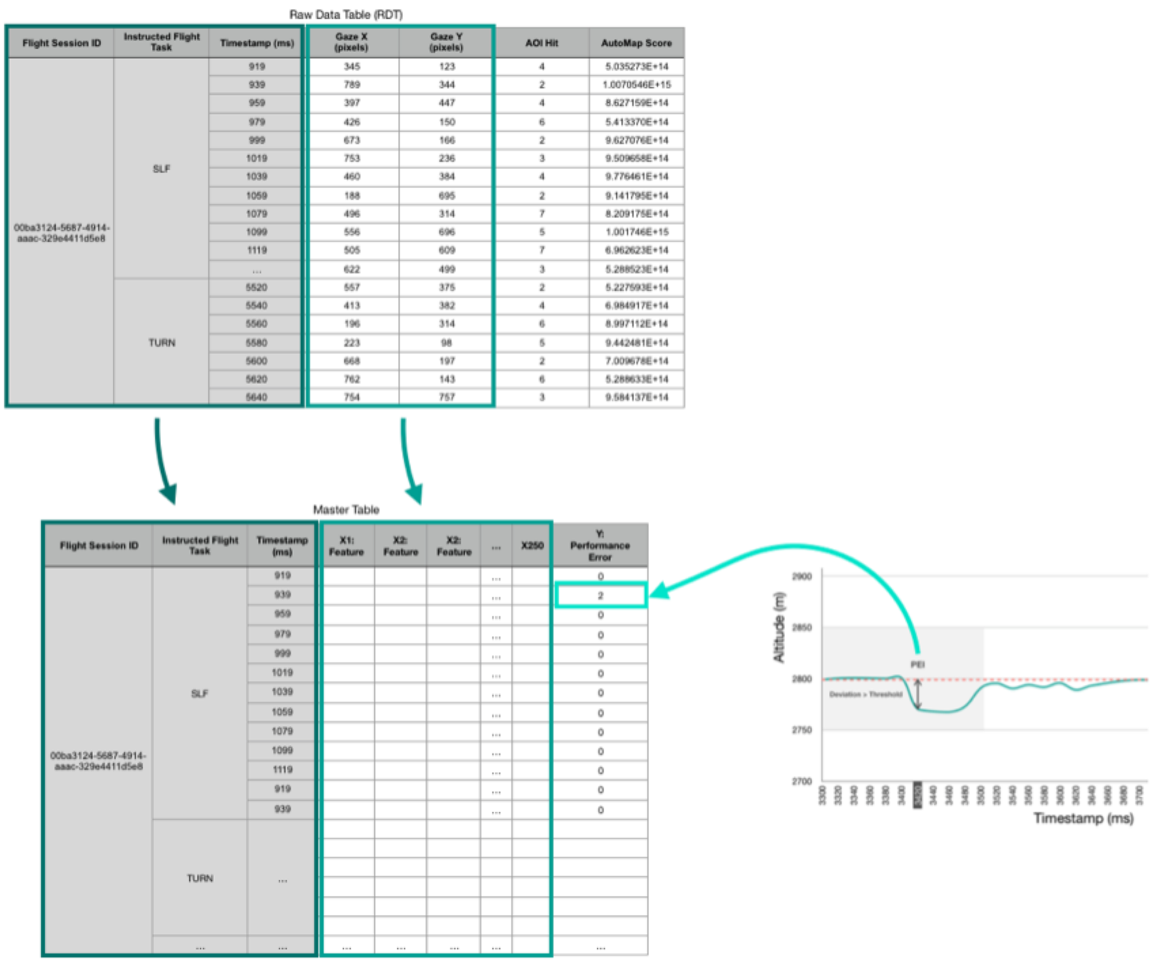
From Raw Data to Master Table

At this point, equipped with the RDT (Raw Data Table) and the PEIs (Performance Error Instants), it would be possible to generate a Master Table, as follows:

1. **From the RDT:** isolate the Flight Session ID, Instructed Flight Task and Timestamp columns to create the Spine
2. **From the RDT:** utilize the Gaze X and Gaze Y columns for calculations, aggregation functions to create the Features
3. **From the PEIs:** set all PEI events in the sequence in a column to from the Target
4. Concatenate the Spine, Features, and Target to obtain a Master Table

This slicing and grouping are shown below, for illustration:

Figure 15: Bulding Master Table from RDT and PEIs. (Source: Author).



Raw Data Table (RDT)

Flight Session ID	Instructed Flight Task	Timestamp (ms)	Gaze X (pixels)	Gaze Y (pixels)	AOI Hit	AutoMap Score
00ba3124-5687-4914-aaac-329e4411d5e8	SLF	919	345	123	4	5.035273E+14
		939	789	344	2	1.0070546E+15
		959	397	447	4	8.627159E+14
		979	426	150	6	5.413370E+14
		999	673	166	2	9.627076E+14
		1019	753	236	3	9.509658E+14
		1039	460	384	4	9.776461E+14
		1059	188	695	2	9.141795E+14
		1079	496	314	7	8.209175E+14
		1099	556	696	5	1.001746E+15
		1119	505	609	7	6.962623E+14
		...	622	499	3	5.288523E+14
	TURN	5520	557	375	2	5.227593E+14
		5540	413	382	4	6.984917E+14
		5560	196	314	6	8.997112E+14
		5580	223	98	5	9.442481E+14
		5600	668	197	2	7.009678E+14
		5620	762	143	6	5.288633E+14
		5640	754	757	3	9.584137E+14

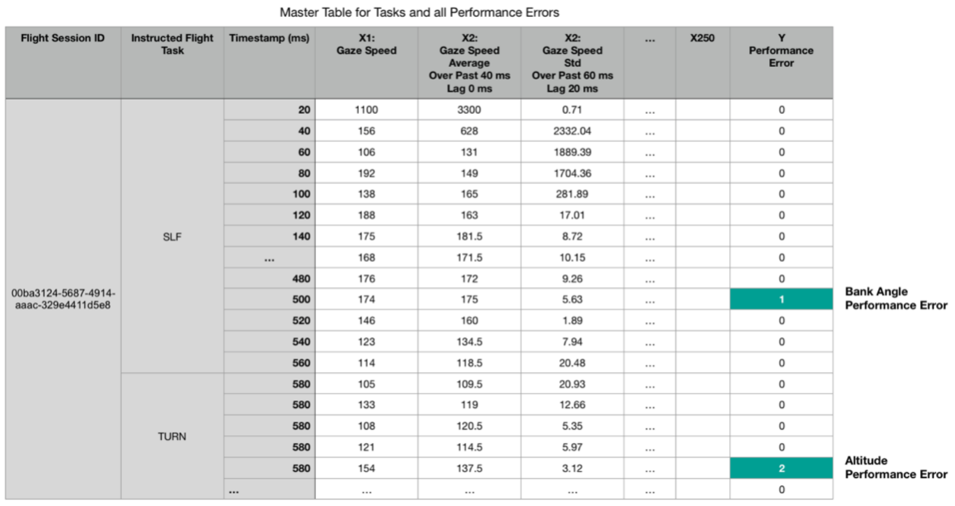
Master Table

Flight Session ID	Instructed Flight Task	Timestamp (ms)	X1: Feature	X2: Feature	X2: Feature	...	X250	Y: Performance Error
00ba3124-5687-4914-aaac-329e4411d5e8	SLF	919				...		0
		939				...		2
		959				...		0
		979				...		0
		999				...		0
		1019				...		0
		1039				...		0
		1059				...		0
		1079				...		0
		1099				...		0
		1119				...		0
		919				...		0
		939				...		0
	TURN	...						
	...	...	...	...	...	...		...

Reduced Master Tables

A straight-forward approach to the Machine Learning section's steps would be to construct a Master Table, including all of the available gaze data in the RDT (Raw Data Table) with their associated PEIs. Such a design would yield a table like this at the end of the data processing step:

Figure 16: Master Table with Multi-class Target. (Source: Author).



Master Table for Tasks and all Performance Errors

Flight Session ID	Instructed Flight Task	Timestamp (ms)	X1: Gaze Speed	X2: Gaze Speed Average Over Past 40 ms Lag 0 ms	X2: Gaze Speed Std Over Past 60 ms Lag 20 ms	...	X250	Y Performance Error	
00ba3124-5687-4914-aaac-329e4411d5e8	SLF	20	1100	3300	0.71	...		0	
		40	156	628	2332.04	...		0	
		60	106	131	1889.39	...		0	
		80	192	149	1704.36	...		0	
		100	138	165	281.89	...		0	
		120	188	163	17.01	...		0	
		140	175	181.5	8.72	...		0	
		...	168	171.5	10.15	...		0	
		480	176	172	9.26	...		0	
		500	174	175	5.63	...		1	Bank Angle Performance Error
		520	146	160	1.89	...		0	
		540	123	134.5	7.94	...		0	
		560	114	118.5	20.48	...		0	
	TURN	580	105	109.5	20.93	...		0	
		580	133	119	12.66	...		0	
		580	108	120.5	5.35	...		0	
		580	121	114.5	5.97	...		0	
		580	154	137.5	3.12	...		2	Altitude Performance Error
		...	...	...	...	...		0	

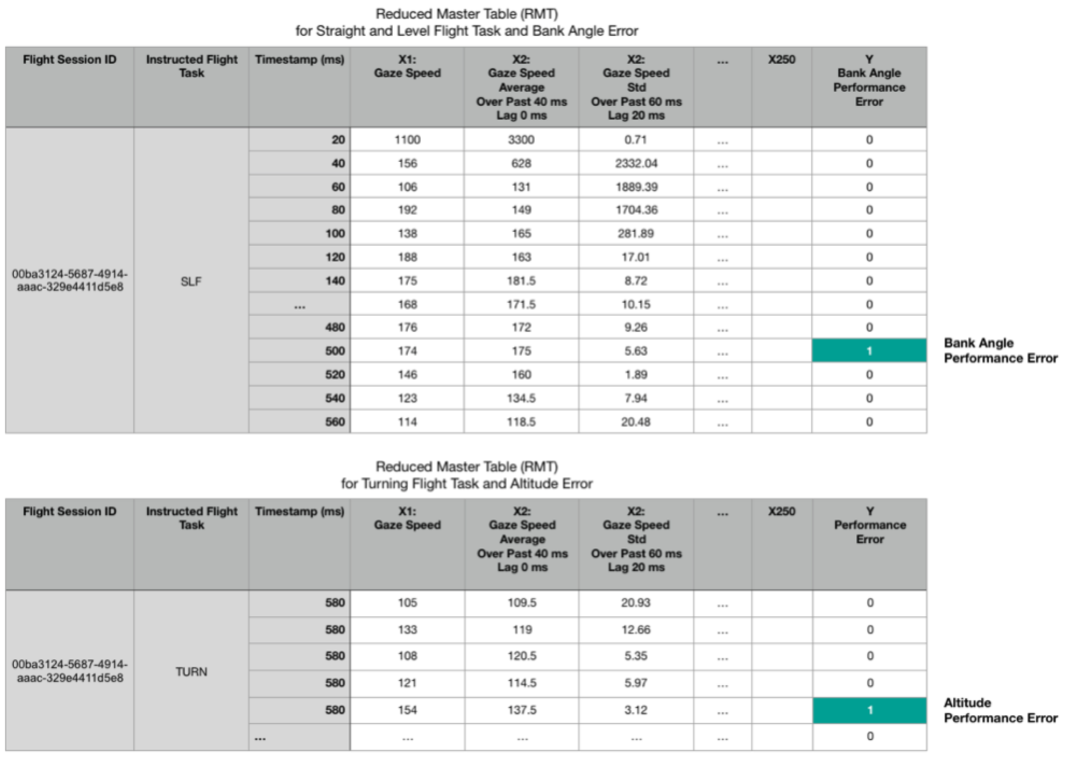
A different approach will be proposed, in which several task-specific, performance-error-specific Master Tables are created instead. These reduced master tables would have the same elements (spine, features, and target) but reduced scope. To obtain such a Reduced Master Table (RMT) from the complete Master Table, mere filtering would be needed, as such:

1. The Master Table's Spine is filtered to contain data from only a single Instructed Flight Task (e.g., turn, descent, straight and level flight)

2. The Master Table's Target is filtered to contain data from only a single Performance Error

By this point, Reduced Master Tables would look as follows, based on the data shown in 16 :

Figure 17: Reduced Master Tables. (Source: Author).



Reduced Master Table (RMT)
for Straight and Level Flight Task and Bank Angle Error

Flight Session ID	Instructed Flight Task	Timestamp (ms)	X1: Gaze Speed	X2: Gaze Speed Average Over Past 40 ms Lag 0 ms	X2: Gaze Speed Std Over Past 60 ms Lag 20 ms	...	X250	Y Bank Angle Performance Error
00ba3124-5687-4914-aaac-329e4411d5e8	SLF	20	1100	3300	0.71	...		0
		40	156	628	2332.04	...		0
		60	106	131	1889.39	...		0
		80	192	149	1704.36	...		0
		100	138	165	281.89	...		0
		120	188	163	17.01	...		0
		140	175	181.5	8.72	...		0
		...	168	171.5	10.15	...		0
		480	176	172	9.26	...		0
		500	174	175	5.63	...		1
		520	146	160	1.89	...		0
		540	123	134.5	7.94	...		0
		560	114	118.5	20.48	...		0

Bank Angle Performance Error

Reduced Master Table (RMT)
for Turning Flight Task and Altitude Error

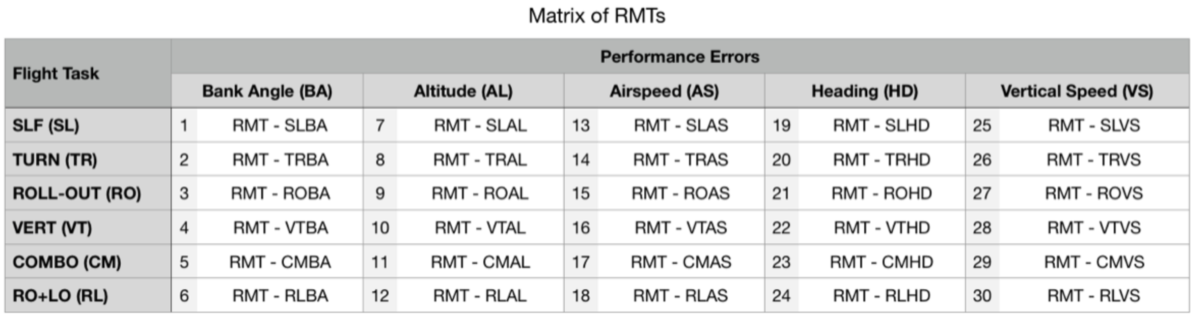
Flight Session ID	Instructed Flight Task	Timestamp (ms)	X1: Gaze Speed	X2: Gaze Speed Average Over Past 40 ms Lag 0 ms	X2: Gaze Speed Std Over Past 60 ms Lag 20 ms	...	X250	Y Performance Error
00ba3124-5687-4914-aaac-329e4411d5e8	TURN	580	105	109.5	20.93	...		0
		580	133	119	12.66	...		0
		580	108	120.5	5.35	...		0
		580	121	114.5	5.97	...		0
		580	154	137.5	3.12	...		1
		...	...	...	...	...		0

Altitude Performance Error

An important insight, at this point, is that the target, for each RMT, is a column that can assume either zero or values from 1 to 6 according to the kind of Performance Error (as mentioned in Table 5). An elementary step would be to transform all 1-5 values to 1, or True. This seemingly simple transformation hides an important change of approach from solving a large multi-class classification problem to solving several small binary-class classification problems.

The next logical step is to examine how the problem would be framed in this Reduced Master Tables format. As these tables contain each one, a subset of the complete master table, reduced in scope by two dimensions (or columns), a single RMT would be generated for each pair of Instructed Flight Task and Performance Error in the problem. This can be represented in a matrix, as such:

Figure 18: Matrix of all RMTs in the problem. (Source: Author).

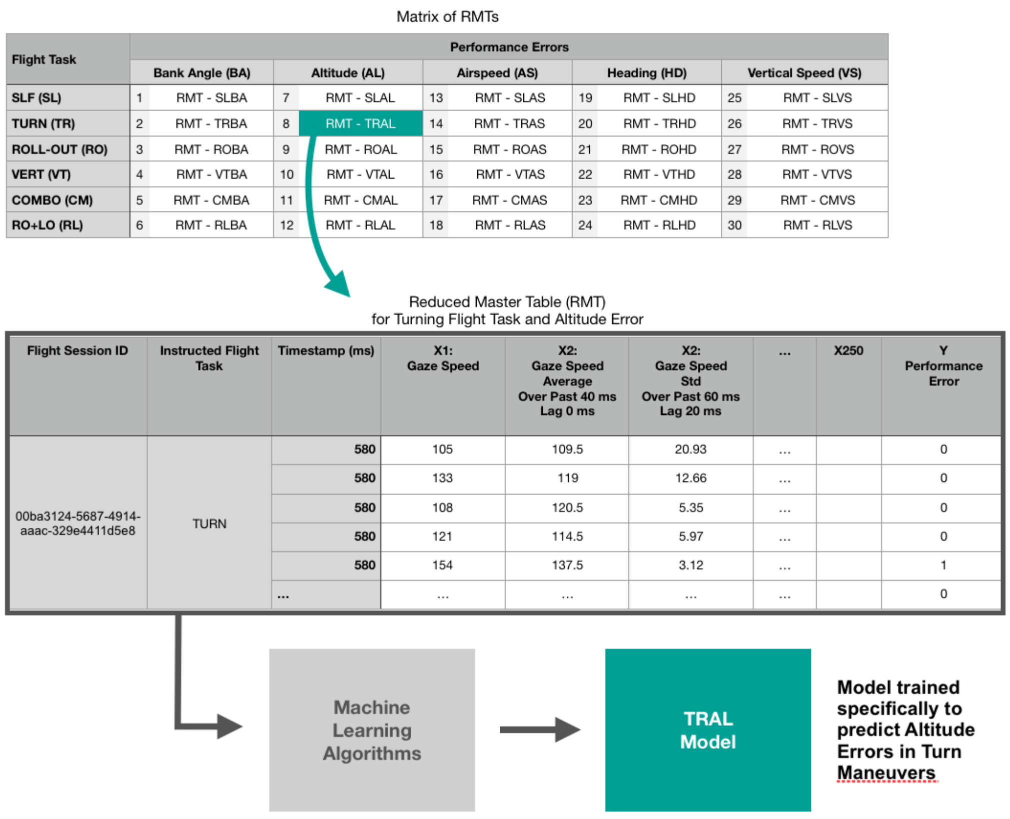


Matrix of RMTs

Flight Task	Performance Errors									
	Bank Angle (BA)		Altitude (AL)		Airspeed (AS)		Heading (HD)		Vertical Speed (VS)	
SLF (SL)	1	RMT - SLBA	7	RMT - SLAL	13	RMT - SLAS	19	RMT - SLHD	25	RMT - SLVS
TURN (TR)	2	RMT - TRBA	8	RMT - TRAL	14	RMT - TRAS	20	RMT - TRHD	26	RMT - TRVS
ROLL-OUT (RO)	3	RMT - ROBA	9	RMT - ROAL	15	RMT - ROAS	21	RMT - ROHD	27	RMT - ROVS
VERT (VT)	4	RMT - VTBA	10	RMT - VTAL	16	RMT - VTAS	22	RMT - VTHD	28	RMT - VTVS
COMBO (CM)	5	RMT - CMBA	11	RMT - CMAL	17	RMT - CMAS	23	RMT - CMHD	29	RMT - CMVS
RO+LO (RL)	6	RMT - RLBA	12	RMT - RLAL	18	RMT - RLAS	24	RMT - RLHD	30	RMT - RLVS

The Machine Learning pipeline presented in the next section will be applied separately to each RMT, yielding a unique task-specific, performance-error-specific ML Model for each cell in the Matrix, as illustrated below:

Figure 19: Process Diagram from each RMT to its Model. (Source: Author).



Matrix of RMTs

Flight Task	Performance Errors									
	Bank Angle (BA)		Altitude (AL)		Airspeed (AS)		Heading (HD)		Vertical Speed (VS)	
SLF (SL)	1	RMT - SLBA	7	RMT - SLAL	13	RMT - SLAS	19	RMT - SLHD	25	RMT - SLVS
TURN (TR)	2	RMT - TRBA	8	RMT - TRAL	14	RMT - TRAS	20	RMT - TRHD	26	RMT - TRVS
ROLL-OUT (RO)	3	RMT - ROBA	9	RMT - ROAL	15	RMT - ROAS	21	RMT - ROHD	27	RMT - ROVS
VERT (VT)	4	RMT - VTBA	10	RMT - VTAL	16	RMT - VTAS	22	RMT - VTHD	28	RMT - VTVS
COMBO (CM)	5	RMT - CMBA	11	RMT - CMAL	17	RMT - CMAS	23	RMT - CMHD	29	RMT - CMVS
RO+LO (RL)	6	RMT - RLBA	12	RMT - RLAL	18	RMT - RLAS	24	RMT - RLHD	30	RMT - RLVS

Reduced Master Table (RMT)
for Turning Flight Task and Altitude Error

Flight Session ID	Instructed Flight Task	Timestamp (ms)	X1: Gaze Speed	X2: Gaze Speed Average Over Past 40 ms Lag 0 ms	X2: Gaze Speed Std Over Past 60 ms Lag 20 ms	...	X250	Y Performance Error
00ba3124-5687-4914-aaac-329e4411d5e8	TURN	580	105	109.5	20.93	...		0
		580	133	119	12.66	...		0
		580	108	120.5	5.35	...		0
		580	121	114.5	5.97	...		0
		580	154	137.5	3.12	...		1
		...	...	...	...	...		0

Machine Learning Algorithms
TRAL Model
Model trained specifically to predict Altitude Errors in Turn Maneuvers

As the method was explained, some important questions arise around the efficiency and advantages of such an approach. An important conceptual question is whether it is efficient or clever to solve 30 small problems instead of a large problem 30 times the size. Although this is a complex discussion in itself, some clarity is presented by GENTLE et al., "*powerful recursive algorithms, such as the Fast Fourier Transform (FFT) and sorting algorithms, follow a divide-and-conquer paradigm: to solve a big problem, break it into little problems and use the solutions to the little problems to solve the big problem*" [24].

Moving on to the advantages posed by this methodology, some are highlighted below:

1. It allows for - still useful - partial solutions to the problem to be offered. If a specific performance error during a certain flight task proves to be impossible to model with the available data, the other models can still function and offer predictions that reduce pilot risk nevertheless

2. It allows for a more iterative process in getting to a final usable product. Each reduced model can go through the whole pipeline (from raw data to tested model) quicker, which enables delivery of intermediate results, rather than only a single, huge, deliverable model

3. It creates a specific model, with a smaller dimensionality of data for training, which increases the potential for good results in a Machine Learning perspective [11]

4. It mitigates chances of a "make-or-break" situation. Reduced models are also quicker to fail, which makes the process of investigating problems quicker, avoiding a situation of a problem that risks the whole project at once

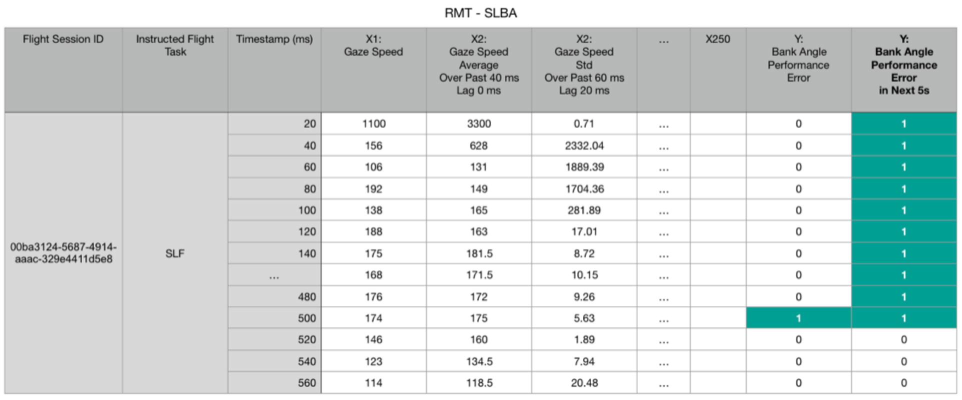
That said, some important caveats of this proposition are the increased complexity of organizing the data processing, modeling, and evaluation pipelines. The tradeoff, nonetheless, seems advantageous at the present state of development of this paper.

Target Refining

At this point, the target is a single column in each RMT containing a binary flag marking Performance Error Events (PEIs). This seems reasonable to the posed problem question: "can gaze data be utilized to predict flight performance errors?". However, a closer look at the problem reveals that detecting the exact instant in time where the error may happen is not as important as finding a time window in which it could then alert the pilot.

Defining such a window size is a problem in itself, but it is possible to generate several window versions and then optimize for model quality its size. Implementing this change on the target column would yield a column with a slight modification, as shown below:

Figure 20: Target with time window. (Source: Author).



RMT - SLBA

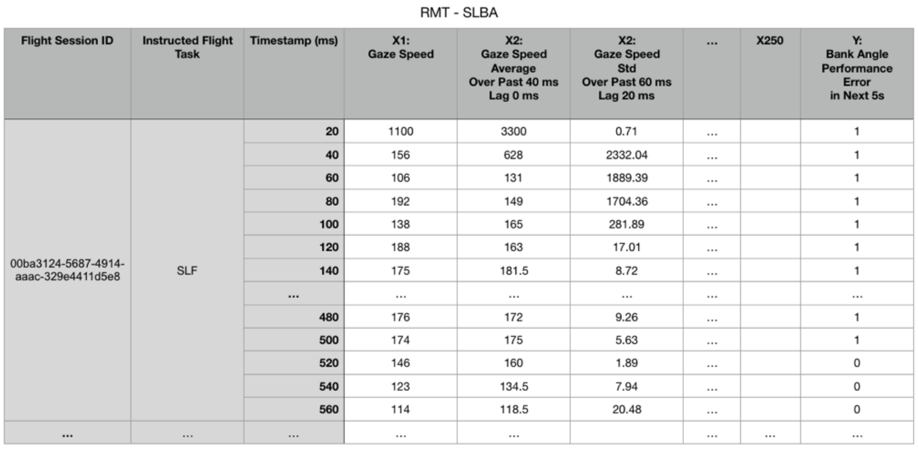
Flight Session ID	Instructed Flight Task	Timestamp (ms)	X1: Gaze Speed	X2: Gaze Speed Average Over Past 40 ms Lag 0 ms	X2: Gaze Speed Std Over Past 60 ms Lag 20 ms	...	X250	Y: Bank Angle Performance Error	**Y: Bank Angle Performance Error in Next 5s**
00ba3124-5687-4914-aaac-329e4411d5e8	SLF	20	1100	3300	0.71	...		0	1
		40	156	628	2332.04	...		0	1
		60	106	131	1889.39	...		0	1
		80	192	149	1704.36	...		0	1
		100	138	165	281.89	...		0	1
		120	188	163	17.01	...		0	1
		140	175	181.5	8.72	...		0	1
		...	168	171.5	10.15	...		0	1
		480	176	172	9.26	...		0	1
		500	174	175	5.63	...		1	1
		520	146	160	1.89	...		0	0
		540	123	134.5	7.94	...		0	0
		560	114	118.5	20.48	...		0	0

An added benefit to this procedure is that it also mitigates the problem of imbalance of the dataset (larger number of successful examples than performance error examples) by adding more "ones" to the target column. This problem is further discussed and analyzed in the next section.

Final Preprocessed Tables

Following the steps described, one individual Reduced Master Table can be created for each Performance Error, Instructed Flight Task pair, and structured in the Spine, Feature, Target format, including the suggested refinements. An example of such a table is illustrated below:

Figure 21: Final Reduced Master Table for Straight and Level Maneuver and Bank angle Performance Error (RMT - SLBA). (Source: Author).



RMT - SLBA

Flight Session ID	Instructed Flight Task	Timestamp (ms)	X1: Gaze Speed	X2: Gaze Speed Average Over Past 40 ms Lag 0 ms	X2: Gaze Speed Std Over Past 60 ms Lag 20 ms	...	X250	Y: Bank Angle Performance Error in Next 5s
00ba3124-5687-4914-aaac-329e4411d5e8	SLF	20	1100	3300	0.71	...		1
		40	156	628	2332.04	...		1
		60	106	131	1889.39	...		1
		80	192	149	1704.36	...		1
		100	138	165	281.89	...		1
		120	188	163	17.01	...		1
		140	175	181.5	8.72	...		1
		...	...	...	...	...		...
		480	176	172	9.26	...		1
		500	174	175	5.63	...		1
		520	146	160	1.89	...		0
		540	123	134.5	7.94	...		0
		560	114	118.5	20.48	...		0
...	...	...	...	...		...	...	...

4.3 Modeling

4.3.1 General Machine Learning Structure

This section addresses the proper machine learning modeling after the master table was obtained by following the aforementioned methods (as exemplified in Figure 10).

Through machine learning methods we intend to obtain a desired value Y, using $(X_1, X_2, ..., X_N) = X$ variables. Each X_j is a *model feature* and Y is the *target*.

X_j is a vector of *observations* $(x_{0,j}, x_{1,j}, ..., x_{j,n})$ a a given feature.Y is a vector of *labels* y_i of the target. N is the number of features and n is the number of observations made (another name for observation is *sample*, and both are used as synonyms in this work).

However, Y cannot be determined using X because:

items Y is not deterministic regarding X. In general, more variables than contained in X are needed to determine Y perfectly.Even if X variables can determine Y, it is impracticable to measure the response Y for all possible states of X.

Although, it is possible to estimate Y through X:

$$\hat{Y} = f(X),$$

where $\hat{Y}$ is an approximation to Y and f is our model.

In our specific case, $y_i = 1$ when a performance error occur and $y_i = 0$ otherwise in a given time instant. X is the collection of all features generated in the master table, as described in the section 4.2.4. A summary of this is shown in the table 6.

The process of obtaining f is called *training* the model; to do so, we use known data for features X and its label Y. Different modeling techniques obtain different f, which leads to different models.

An important point of any classification problem is called the Classification Boundary (or Decision Threshold, or even so, Classification Threshold) D_t. When producing the $\hat{Y}$ prediction, a binary classifier produces a probability of a $y_i = 1$ for the observation, ranging from zero to one. The decision threshold is the smallest value of probability for a positive label classification, as presented in the equation 4.1. If, for example, an observation i has a positive class probability of 0.7, and the threshold is set as $D_t = 0.6$, this point is classified as positive.

Table 6: Theoretical features and target table

Observation $\hat{Y}$	X_1	X_2	$\cdots$	X_N	Y
1 $\hat{y}_1$	$x_{1,1}$	$x_{1,2}$	$\cdots$	$x_{1,N}$	y_1
2 $\hat{y}_2$	$x_{2,1}$	$x_{2,2}$	$\cdots$	$x_{2,N}$	y_2
$\vdots$ $\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\cdots$
n $\hat{y}_n$	$x_{n,N}$	$x_{n,N}$	$\cdots$	$x_{n,N}$	y_n

$$\hat{y} = \begin{cases} 1, & \text{if } \mathbb{P}(y=1) \geq D_t \\ 0, & \text{else} \end{cases} \tag{4.1}$$

4.3.2 Model Selection

Abbreviations

ML: Machine learning

DL: deep learning

GBDT: Gradient Boosted Decision Trees

4.3.2.1 Bias Variance Dilemma

The choice of an algorithm for a Machine Learning problem is mostly related to the bias-variance tradeoff. The goal is to obtain a model that minimizes variance while also keeping a minimal bias. This can be better achieved by a certain algorithm, depending on some characteristics of the data itself and the type of problem being solved.

This section aims to clarify some concepts of problem types, data size, algorithm characteristics, and then suggest an initial guess of the best algorithm for our problem of performance error prediction based on gaze.

Definitions

Bias: the inability of a model to capture the behavior of a certain set of data correctly; high bias is associated with under-fitting

Variance: the difference in for the same model when applied to different datasets; high bias is associated with overfitting

Overfitting: The production of an analysis which corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably [43]

Underfitting: a model that cannot properly model the training data and not generalize well to new data. An analysis essentially fits too loosely the data and does not provide good measures for future predictions.

4.3.2.2 Problem Types

Machine learning problems can be broken down into supervised or unsupervised (defined below), depending on the data being labeled or not, and classification or regression (defined below) depending on whether the target is a continuous numerical value a discrete set of classes. The problem presented in this dissertation is supervised, as the target labels are created before the ML pipeline (as shown in the section 4.2.4). It is also a classification problem, as the target consists of a boolean indicator of performance error (as presented in the section 4.2.4).

This is the first important separation to narrow the selection of an adequate algorithm, basically due to the way algorithms operate (e.g., a perceptron algorithm takes values of the target variable is continuous intervals, which would not work properly in a classification problem). A supervised classification problem leads to algorithms that can yield good results: SVM, Logistic Regression, Random Forrest, and Gradient Boosted Trees. These algorithms will be explained in the following sections.

Comment on Neural Networks

The developments in Deep Learning (neural networks stacked in hundreds or thousands of layers) in the last decade have been substantial and demonstrated impressive results in various research fields. However, this option was not explored in this work because of the early stage of Deep Learning development for applications with tabular data.

In general, DL is most effective when it can learn deep hierarchical representations of data, such as in language processing and image analysis. When it comes to structured data (such as tables), nevertheless, some challenges still exist for deep learning to be used. According to Tune [55], a few can be highlighted:

- Tabular data is often heterogeneous, with columns containing different types of information (e.g., time in milliseconds, adimensional entropy);
- Features themselves are many times sparse: unlike data from images, audio, and language, there can be a little variation in a column of a table;
- In tables, there are typically more categorical features, in which the order (and value) of the features themselves are not important, and unlike numerical features, are discrete by nature;

Tune continues, *"these differences lead to features in a high-dimensional space that is generally not dense and continuous, making it difficult to exploit for a typical deep neural network"* [55].

As the problem here presented is represented mostly by tabular information, these challenges have to lead our analysis towards the use of non-deep learning algorithms, which have knowingly shown better performance in such scenarios.

4.3.2.3 Description of Selected Algorithms

Support Vector Machines (SVMs)

Support vector machines (SVMs) are linear classifiers based on the margin maximization principle. They perform structural risk minimization, which improves the classifier's complexity to achieve excellent generalization performance. The SVM accomplishes the classification task by constructing, in a higher-dimensional space, the hyperplane that optimally separates the data into two categories [25].

Mathematical definition

The optimization problem and its solution can be represented in a special way that only involves the input features via inner products. Applying this directly for the transformed feature vectors h(xi) shows that for particular choices of h, these inner products can be computed easily. The Lagrange dual function has the form [27]:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \left\langle h\left(x_i\right), h\left(x_{i'}\right)\right\rangle$$

From we see that the solution function f(x) can be written:

$$f(x) = h(x)^T \beta + \beta_0 \quad = \sum_{i=1}^{N} \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0$$

Some advantages of this algorithm are that it is effective in high dimensional spaces. It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient. It can also be considered versatile as different Kernel functions can be specified for the decision function. These prove the classification problem posed here [45]. When the number of features is much greater than the number of samples, there is a higher risk of over-fitting in choosing Kernel functions. This, though, is not a problem that will likely occur in our case.

The last caveat is that SVMs do not directly provide probability estimates. To obtain these, an expensive five-fold cross-validation needs to be utilized, which may increase computing times.

Logistic Regression

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt), or the log-linear classifier. In this model, the probabilities describing a single trial's possible outcomes are modeled using a logistic function [45].

Mathematical Definition

As an optimization problem, binary class $l2$ penalized logistic regression minimizes the following cost function:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log \left(\exp \left(-y_i \left(X_i^T w + c \right) \right) + 1 \right)$$

Some positive notes of this method are that it is easy to implement, interpret, and efficient to train. It also makes no assumptions about distributions of classes in feature space and can easily extend to multiple classes(multinomial regression). Generally, it offers good accuracy when the dataset is linearly separable. Lastly, its coefficients can be interpreted as indicators of feature importance.

On negative notes, logistic regression can overfit in high dimensional datasets. It also constructs linear boundaries, and the major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables. It is generally tough to obtain complex relationships using logistic regression.

Decision Trees and Gradient Boosting

Boosting is one of the most profitable learning ideas introduced in the last decades. It was originally designed for classification problems, with the motivation to develop a procedure that combines the outputs of many "weak" classifiers to produce a stronger "group" [46]. Specifically, when used with Decision Trees, it is called Gradient Tree Boosting or Gradient Boosted Decision Trees (GBDT). It is, essentially, a generalization of boosting to arbitrary differentiable loss functions. [45]

Mathematical definition

A tree can be formally expressed as:

$$T(x;\Theta) = \sum_{j=1}^{J} \gamma_j I\left(x \in R_j\right)$$

, with parameters $\Theta = \{R_j \gamma_j\}_1^J$

The parameters are found by minimizing the empirical risk:

$$\hat{\Theta} = \arg\min *\Theta \sum *j = 1^J \sum_{x_i \in R_j} L\left(y_i, \gamma_j\right)$$

This is in itself a complex problem, and it is acceptable to utilize approximate sub-optimal solutions.

The boosted tree model, then, can be written as a sum of such trees:

$$f_M(x) = \sum_{m=1}^{M} T\left(x; \Theta_m\right)$$

4.3.3 Model Validation

Before taking action to improve the model performance, we must define accuracy. For a continuous result model, the mean square error can be defined for a given dataset. Despite that, this approach is not very adequated for binary yes-or-no output. James, G. et. al. [46] defines the *error rate* as:

$$e_{ratio} = \frac{1}{n} . \sum_{i=1}^{n} I(y_i \neq \hat{y}_i)^2$$

, where $I = 1$ if $y_i - \hat{y}_i$, otherwise is zero.

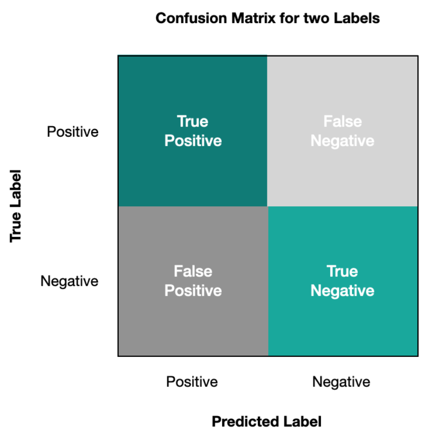
Finally, the model accuracy can be defined as

$$acc = 1 - e_{ratio}$$

Confusion Matrix and Error Types

To better understand the model performance, we can divide the model results into true and false positives and negatives. A true positive or negative situation is when the model predicted $y_i = \hat{y}_i$ correctly. $y_i = \hat{y}_i = 1$ stand for the true positives and $y_i = \hat{y}_i = 0$ for the true negatives. A false negative is when $\hat{y}_i = 0 \neq y_i = 1$ and a false positive is $\hat{y}_i = 1 \neq y_i = 0$. The amount of these results can be summarized in what is called a confusion matrix:

Figure 22: Confusion Matrix. (Source: Author).



The present model is developed to integrate real-time in-simulation and in-flight applications, eventually. The GAPE model indicates positive for performance failure; hence, in case of a false positive, the pilot is flying correctly and a possible warning low effect on performance (if this kind of error is too great, the pilot can start to ignore the warnings, but we will not consider this limitation now). In the case of a false negative, the pilot is not flying correctly and not aware of it, which can lead to an accident. Therefore, a false negative error is much more critical for flight safety than a false positive.

Recall

An equivalent way of calculating the model accuracy is:

$$\frac{true\ positive + true\ negatives}{true\ positive + true\ negative + false\ positive + false\ negative}$$

This way of calculating the accuracy does not account for error (false positive or false negative). Model accuracy will be the recall accuracy, which measures the success of the model regarding true positives, and it is defined as:

$$\frac{true\ positive}{true\ positive + false\ negative}$$

Note

The combination of $true\ positives$ and $false\ negatives$ accounts for all situations where $y = 1$.

Receiver Operating Characteristics (ROC) and Area Under the Curve

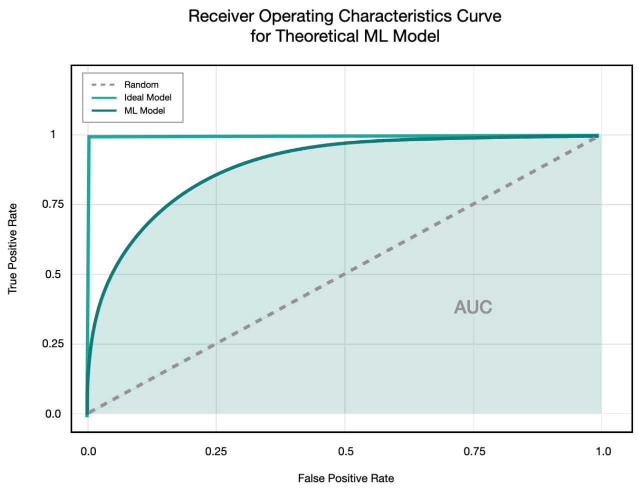
The Receiver Operating Characteristics plots the (recall) accuracy versus the False positive rate, defined as

$$\frac{false\ positive}{false\ positive + true\ negative}$$

for different threshold values. It is another way of evaluating the performance of a binary classifier model. The ideal classifier curve is straight upward until value 1, which represents absolute prediction. The $y = x$ line resents a fully random model, a coin toss, hence any model performance worst than worthless.

The AUC is the *area under the curve* and can summarize the results of any curve. when using the ROC curve, the ideal model has $AUC = 1$ and the random model has $AUC = 0.5$.

Figure 23: Receiver Operating Characteristics Curve for Theoretical ML Model. (Source: Author).



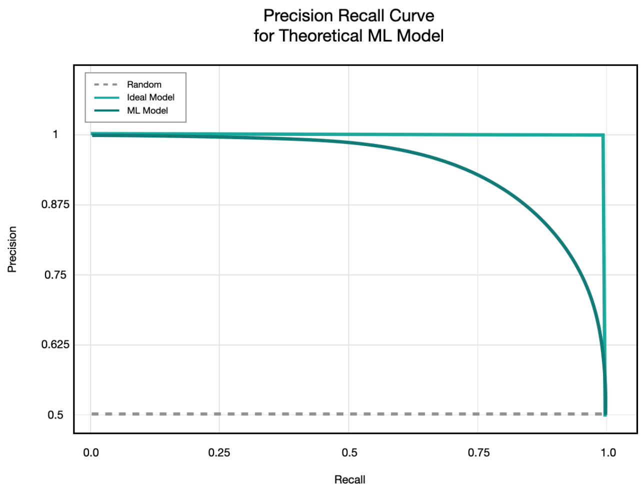
Precision-Recall Curve

Considering the *precision* of the model, defined as

$$\frac{true\ positive}{true\ positive + false\ positive}$$

we can plot the Precision versus the Recall for different classificatory thresholds (as done for the ROC Curve). The Precision-Recall Curve also displays a tradeoff of accuracy and overfitting, but it is more suitable for imbalanced data, as is the case.

Figure 24: Precision Recall Curve for Theoretical ML Model. (Source: Author).



Lift Curve

The Lift curve shows how often the model is better than a random selection for a certain percentage of samples classified as the positive class.

As mentioned in the section 8.4, when the classification threshold th (or classification boundary) changes, the prediction $\hat{Y}$ also changes, and therefore many model metrics change. The *Sample Ratio* is the rate of samples classified as 1, and it is defined as:

$$Y_{rate,th} = \frac{true\ positive_{th}}{true\ positive_{th} + false\ negative_{th}}$$

It is easy to recognize that as the threshold tends to zero, the $Y_{rate,th}$ tends to 1: as the model gets less demanding, more samples are classified as True, making the $false\ negative_{th}$ goes to zero. To the opposite side, when the model gets more demanding, the model is less, and less capable of detecting true positives: when $th = 1$, $true\ positive_{th} = 0$, which makes $Y_{rate,th} = 0$.

The *Gain* is how many times our model is better than a random selection, and it is defined as

$$Gain = \frac{recall_{th}}{random\ recall},$$

The $recall_{th}$ is the recall of the selected model (in that threshold), and the $random\ recall$ is the recall of purely random decision.

The Lift curve is the plot of $Gain$ as a function of Y_{rate} and helps evaluate the model once the classification threshold is set.

What is a good performance?

Feng et al. [61] used gaze behavior data and achieved 84.85% model accuracy when predicting aircraft pilot workload. Liu's model [23] delivered 90.25% accuracy when recognizing workload in lab conditions through gaze data. Li's model [28] obtained AUC of 0.96 for a particular set of features when predicting construction workers fatigue, once again, using gaze data. All of those authors considered all their models' performance satisfying.

It is completely expected from pilots not to commit performance errors, and most of the time, they perform well. This causes an imbalance in the data. Approximately 91% of the pilots' tasks do not contain error; consequently, if a model guessed every time that an error did not occur, it would be correct 91% of the time. Henceforth, this number constitutes a baseline performance to the model.

Considering this, and our system is safety-related, we could consider 90% accuracy and an AUC of 0.95 as optimal performance for the model.

4.3.4 Data Leakage

Data leakage is using future data, or data altered by future events, to predict present events. Of course, this is absurd during model applications, and it is possible only during the training phase.

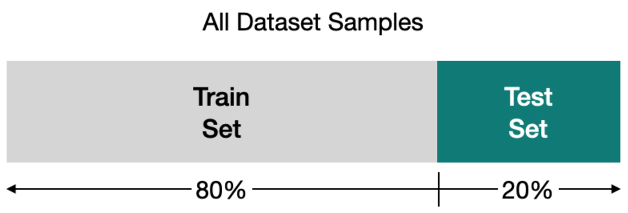
When defining whether an error has occurred or not, if our deviance criteria are too large, the pilot may recognize this performance error before the system and then change its gaze behavior to correct its performance. Using data of this already altered gaze behavior is an example of data leakage because it is contaminated by an error in the future (for model instance).

4.3.5 Train Test Split and Cross-Validation

4.3.5.1 Train-test Split

Training a Machine Learning model requires a way to properly evaluate it, ensuring that it can predict results for data it was never exposed to. This is usually done by what is called a train-test split. The process is quite simple: we split data into two datasets, one for training and one, generally smaller, for testing; then, the model is fitted on the training set while the test set is held out; lastly, the test data is used as input for the model, and the outputs are stored and used to calculate performance metrics. [46]

Figure 25: Train Test Split. (Source: Author).



Various measures can be used to evaluate the predictor, as is defined in the section 4.3.3. A fairly simple one calculates the so-called *test error*, which is the error between the prediction and the test target's actual values.

A relevant aspect of this method is deciding the proportion of data used for training and testing. This is called the train-test split percentage, and typical values are 70-85% of data for training and 15-30% for testing. A more sophisticated approach is to create several splits of different proportions and evaluate each one's model. This is called cross-validation and is explained in more detail hereafter [46].

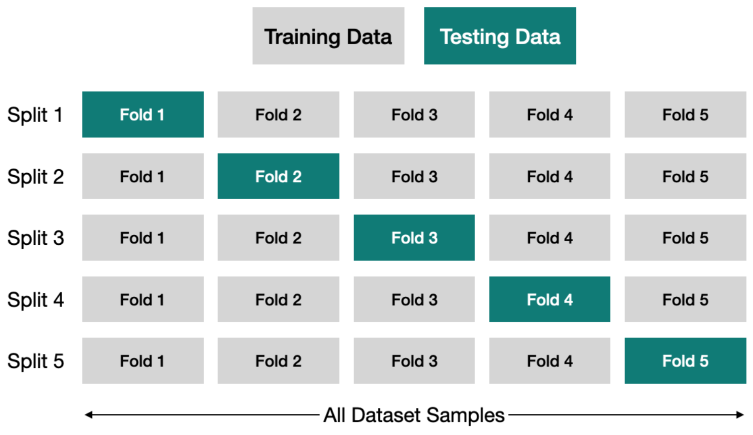
4.3.5.2 Cross Validation

As previously explained, Cross-Validation is the technique of train-test splitting the data with n groups with different split proportions, then fitting and evaluating the performance of the model at each one. This method offers an assessment of how the results of a statistical analysis will generalize to an independent data set. Therefore, it can be used to extract a more robust result metric for a Machine Learning Model.

4.3.5.3 K-Fold Cross Validation

When done simply by splitting the data into groups, this is commonly called K-fold cross-validation, where K indicates the number of groups used for splitting, and fold is the name used for each group. A common variation of this process is to shuffle the train and test data to provide more variability. The figreffig:k$_{f}oldillustratestheprocess$[46].

Figure 26: K-folding Validation Strategy

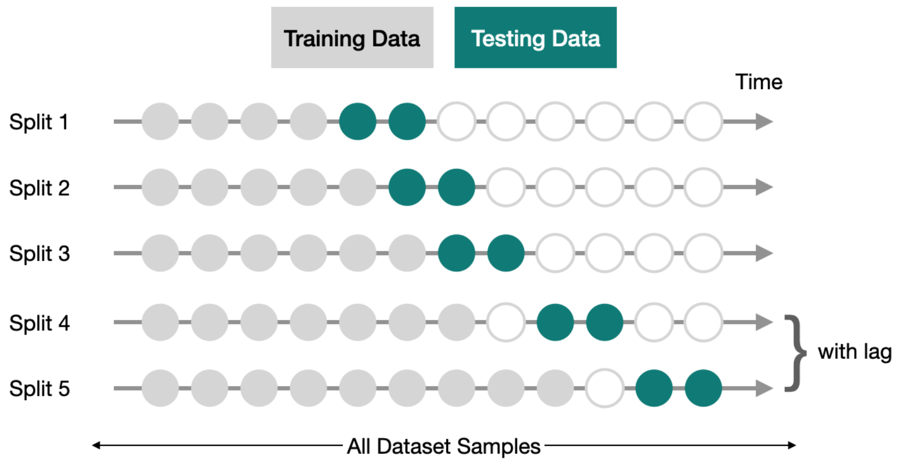


4.3.5.4 Time Series Cross Validation

In time series problems, Cross-Validation can be more complicated. As mentioned in the section, 4.3.4an essential aspect of modeling is avoiding **Leakage**, which is to offer the model some estimator of the target variable as a feature for training, and in time-series problems, commonly occurs when data from a future instant in time is inputted to the model at a time t. Bittencourt and colleagues [13] when doing cross-validation, it is clear that a K-fold approach, especially with shuffling, would cause a great *Leakage* problem.

This is why time series problems require a more sophisticated cross-validation form, usually aptly named *Time-series Cross-Validation*. In this method, the train-test splits are done, making sure that the test data is always from a point in time further in time than the training data. An additional point to consider is using a Target variable with back-propagation, which is implemented in this paper and discussed in section GAPE Framework [4.2.4], in the target item. In this case, extra caution must be taken that a gap - of at least the same size as the back-propagation window - exists between the training and test data. The figure 27 illustrates this method, showing an approach with an expanding train set, above, and a constant size train set, below [13].

Figure 27: Time Series Cross-Validation Strategy



4.3.6 Refinement: Submodeling

To achieve better prediction capacity, the GAPE model will be the collection of several submodels. Each submodel will use a different target to predict the different performance errors (BA, AL, AS, HD, VS). The submodels' division also takes the flight tasks set (SLF, TURN, ROLL-OUT, VERT, COMBO, RO+LO) into account, and the pair flight task-performance error corresponds to an RMT (as shown in Figure 18).

4.3.7 Model interpretation

Machine Learning models are often treated as black boxes as long performance is satisfactory. However, opening this black box's cover may produce insights into the modeled system's cause-effect structure, possibly motivating further research. Model interpretability also makes it possible for non-data-scientists to use and rely on the model's decisions.

Feature Importance is a simple way of doing this. This technique consists of assigning each feature a score representing how useful the feature was to predict $\hat{Y}$. Typically this technique is used to make feature reduction by discarding the lowest scored features, which leads to a model with better computational performance. Nevertheless, feature importance also provides a better understanding of the data and the model.

SHAP values

Shapley Additive exPlanations, or simply SHAP values as defined by Lundberg and Lee [49], produce more detailed information about the model's features. It does that by calculating each feature's marginal contribution to the prediction $\hat{Y}$ of the model.

Each observation receives it is SHAP values; therefore, this method provides local and global interpretability. Knowing the contribution to each observation greatly increases model transparency. We can tell why an answer $\hat{Y}$ was given and if it was either positively or negatively correlated with a certain feature. Of course, by using mean values, global SHAP values provide global information about the features.

To calculate SHAP values, we take each feature's marginal contribution in predicting $\hat{Y}$. This can be done for each possible feature order, leading to different values for each permutation. The mean of all those individual values constitutes the model SHAP value [49].

PART III

RESULTS: GAPE MODEL

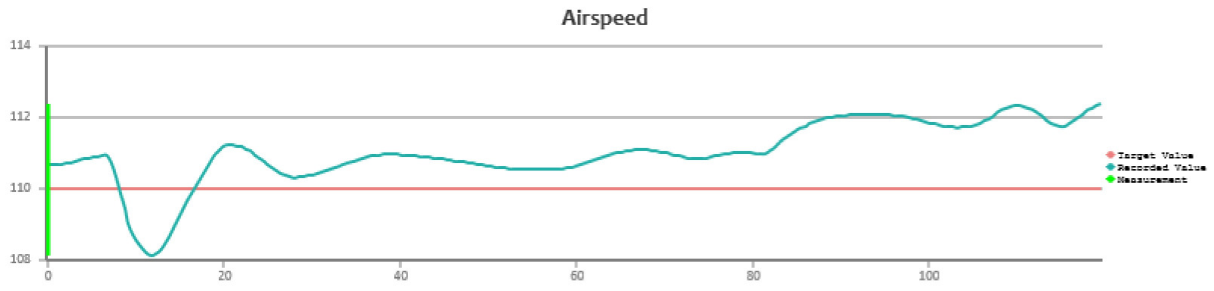
5 PREAMBLE

This section compilation of tests yielded the best outcomes out of a series of attempts that were done, with slightly different metrics and configurations. Some of these included changes to the proposed methodology as, for one reason or another, there were obstacles that imposed necessary changes. In the preamble, we detail the most important difficulties faced during the implementation, the course of action chosen, and adaptations to progress to a final solution.

5.1 Target Variable

The GAPE method 4.2.4 proposed using a Target variable based on the pilot Performance Reports, which contain information about a pilot's desired vs. real performance at a specific flight task and a specific instrument (altimeter, compass, ADI, etc.). Nevertheless, this information is encoded in a complex form, as the performance reports were only available to us as PDF files, including plotted graphs of the performance measurements. An example of such a graph is shown below in Figure 28, where the blue curve indicates the real performance, and the red line, the desired one.

Figure 28: Example of Raw Performance Report



As the GAPE methodology indicated in section **??**, to create the target variable, it was mandatory to extract these graphs in a numeric form (i.e., CSV file), rather than a visual one (i.e., PDF file), and this, in itself, posed a remarkable challenge during the course of our work. In this section, we briefly mention the steps are taken and the process to obtain this data in a more useful format. Still, it is important to mention that this ultimately led to the decision to change our approach and look for a different way to produce a Target variable. This alternative solution is discussed afterward.

5.2 Performance Report Digitizing

The problem we were facing could put simply, be resumed to digitizing information contained in several images in PDF reports of each cadet. This problem is usually called Graph Digitizing in the community, and there are some different approaches to solving it. Some quick numbers on the dimension of the problem: we were dealing with 43 PDF reports, each containing an average of 40 images (3 or 4 per page for 13 pages), which amounted to around 1720 images.

The sheer amount of images quickly showed the need for automation to help with the process. Still, a primary attempt would be to contact the team responsible for the experiment and ask for the original data files. A total of 4 options were explored to try to extract these images, and they are as follows, in order of priority:

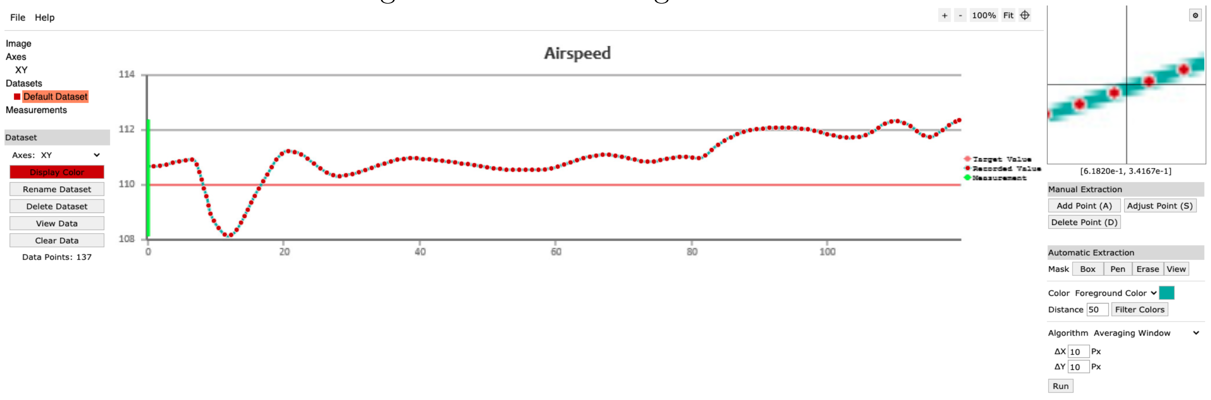
1. Contact the people responsible for the experiment and ask for the original data files 2. Hire a specialized programmer to do image processing and extract the data (OpenCV and similar solutions) 3. Find a Graph Digitizing software to do batch sets of images. Then, split the PDF files into images and run them through the software 4. Enlist the help of a team of people to do Graph Digitizing, either with some help of automation or totally by hand

Each of these options was explored. The experiments, done at the CAE flight academy in Leuven [15], were realized over two years ago, in 2017. Unfortunately, after contacting the Belgium team, we were informed that the original files were not saved anymore, which failed option 1.

The next attempt was to find a program that could use image processing to extract the data. After some searching, we found Fiverr® [2], a platform that offered good specialists in various subjects. We posted a problem description with details and images and were contacted by a programmer that showed interest in working with us. We hired the programmer for a total of EUR 160,00 and maintained contact with him weekly for 3 weeks, as he attempted to solve the problem. Regrettably, after this period, he contacted us to inform us that he could not extract the data and cancel the service, refunding our fee, which failed option 2.

The third solution included research on various websites until we found a solution at WebPlotDigitizer [3]. The Website shown in the image 29 allows for extracting tabular numeric data from graph images. We proceeded to separate all images in the PDF reports using a custom Python script and then process them one by one on the software. As we realized with time, although the process included automated parts, it still required us to calibrate X and Y axis with their respective coordinates in the images and indicate some parameters (color of the curve, margin of error, etc.) for each processed image. At that rate, each image took around 1-2min to process, and considering the total number of images, it would take anywhere from 40 to 60 hours of work to complete. Given the time constraints, this failed option (3).

Figure 29: WebPlotDigitizer Interface



Axes: XY

Data Points: 137

[6.1820e-1, 3.4167e-1]

Color Foreground Color

Distance 50

Algorithm Averaging Window

ΔX 10 Px

ΔY 10 Px

The fourth attempt was to follow option 3 but try to enlist other people's help and include some extra automation layers to speed the process. To pursue this solution, we took two main steps:

- We hosted two congregation events in the 03 and 04th of October in which we invited people and offered snacks and a small gift for helping with 1 hour of work in Graph Digitizing. We had an overwhelming show up of a total of 12 people over the course of two days, which allowed us to speed our work tremendously. All the participants are thanked by name in the acknowledgments. An image of the event invitation and some images of the events are in Annex E E - as these were events held during the Covid-19 Pandemic, all relevant health precautions were taken for the helpers that went in person
- A layer of automation running on top of the WebPlotDigitizer software was developed, based on a web-driver script written in the Selenium IDE [4]. This step

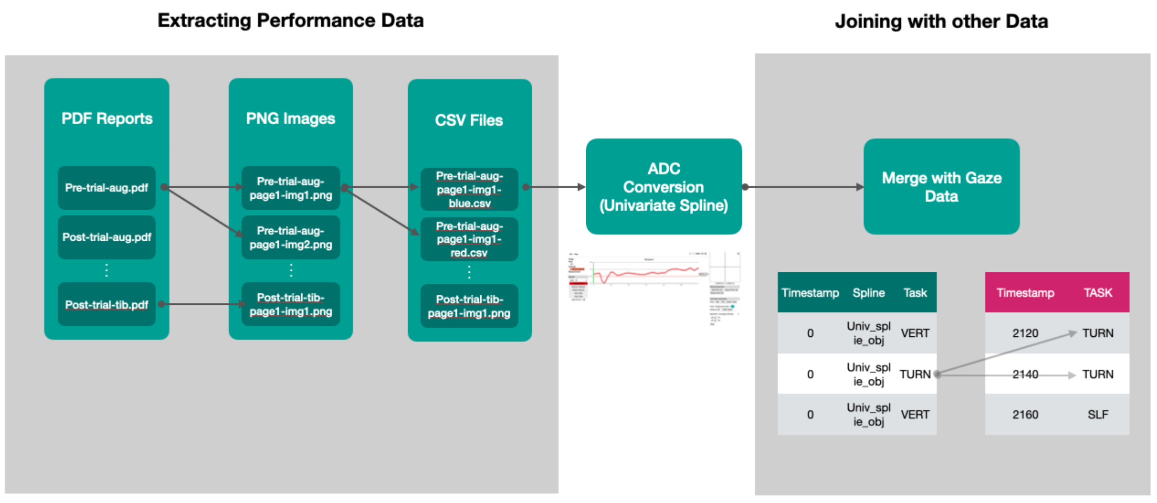
automated several repetitive tasks that had to be hand done, such as click-selecting the curve color, typing the error margin, writing the dataset name, and downloading the data. This made the processing time of a single image decrease to around 30s per image. A reproduction of the code in its entirety is in Annex B F.

With these initiatives, the total work-load per person reduced from around 30h to 40 min, which completely changed the perspective on getting the results on time. After the two events, all the graphs' data was successfully digitized, indicating the option (4). The next step is to integrate this graph information into the gaze dataset. This is the step that ultimately failed, as will be discussed hereafter.

5.3 Merging of Performance Report and Gaze Data

As mentioned, after the graph images were digitized, a process had to be followed to integrate these performance curves with gaze data. As shown in the next Figure 30, after digitizing the graphs (which creates discrete data points with a relatively low sampling rate), it is necessary to do a digital to analog conversion (which was done by spline interpolation), and then merge with gaze data, by evaluating the newly created spline functions at the correct timestamps in the gaze table.

Figure 30: Performance Report Data Extraction Processes



On the last block of Figure 30, we found the biggest challenge in the process, as the time frames of the gaze data and the performance reports were not stored in a compatible way, and matching proved a complex problem. Two main problems existed:

- As presented previously, time was recorded continuously (absolute time) for each flight session in the gaze tables. On the performance reports, nonetheless, time (taken from the x axis of the graphs) was recorded as relative to each flight task (starting at zero for each performance graph). This inhibits the most intuitive approach of using time as a matching factor to merge both datasets
- The performance reports did not include data on all the flight tasks in the flight sessions. Basically, only the more "performance-related" tasks were stored, i.e., TURN, VERT, and COMBO maneuvers. Tasks such as RO and RO+LO were not stored. This meant that the time axis had gaps between the graphs, and it was not possible to "attach" the beginning of one graph to the end of the previous one.

At this point, a basic analysis was done to realize that the tradeoff of time required to solve the merging of data versus the quality of the target variable obtained would not justify continuing this approach, as far as we had gone. Essentially, given the project time-frame, we decided to pursue an alternative Target variable, with less precise information but much quicker results. This alternative is discussed in the next section.

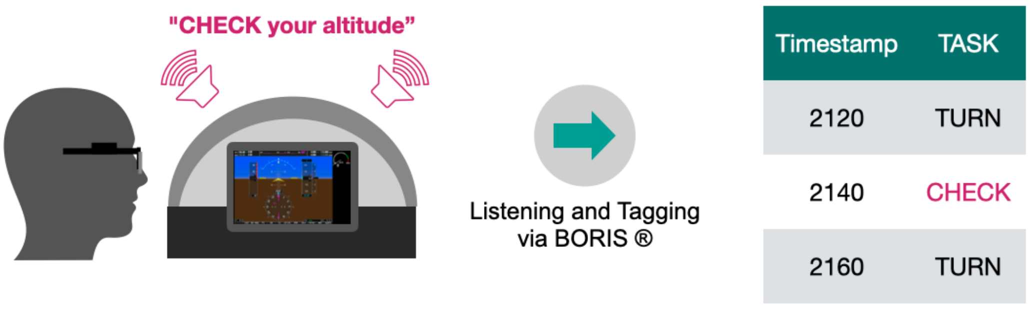
5.4 Alternative Target Variable

Given the previous decision, the alternative solution for a Target variable with less information but quicker implementation was to use the same target proposed in the Preliminary Model Annex B This variable is based on tagging "Check" instructions from the flight sessions' audio recordings. As mentioned in the preliminary model annex 4.2.3, these instructions reflect situations where the cadet was committing some performance flight error. The next figure 31 illustrates this process.

This choice of the target does not fully impede the refinement process indicated in the target refinement section 4.3.6, which was also done to create variations of the target with back-propagation for different time windows.

An important consequence of choosing this alternative is that we have lost information on each instrument's specific errors in the panel. This means that the approach - detailed in the target refinement section 4.3.6 - of creating the RMTs for a Task-Instrument pair will have to be adapted, as is described next.

Figure 31: "Check" Back-Propagation Target

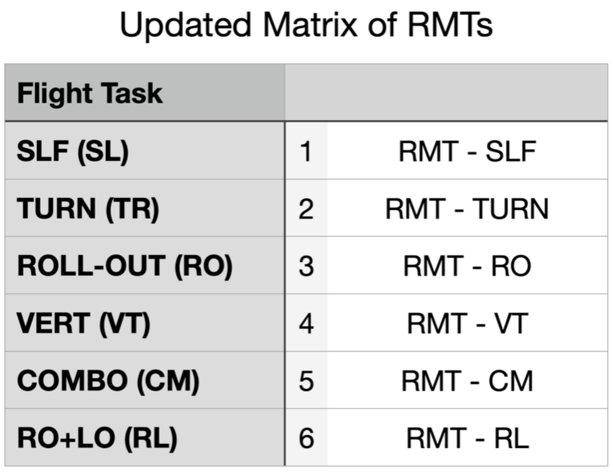


Timestamp	TASK
2120	TURN
2140	CHECK
2160	TURN

5.5 Modified RMTs

Given the choice of the new target variable, it became impossible to create task-specific, instrument-specific RMTs. With the information present, it is now only possible to create RMTs specific to one flight task type. This is an update on the proposed methodology, as illustrated by the next figure.

Figure 32: RMTs for back-propagation target



Updated Matrix of RMTs

Flight Task		
SLF (SL)	1	RMT - SLF
TURN (TR)	2	RMT - TURN
ROLL-OUT (RO)	3	RMT - RO
VERT (VT)	4	RMT - VT
COMBO (CM)	5	RMT - CM
RO+LO (RL)	6	RMT - RL

6 DATA PRESENTATION

6.1 Raw Data

The raw data contains columns with Recording Name (which can be easily converted to a pilot id), Timestamp, Eye Movement Type, Automap Score, GazeX, and GazeY. This data table has 3,065,203 rows (or entries).

6.1.1 Recording Name and ID

The column contains data of the Tobii® recording number. Using an auxiliary table, we can convert to an unique pilot identifier for each flight session. All IDs are shown:

Table 7: List of flight session IDs in original data

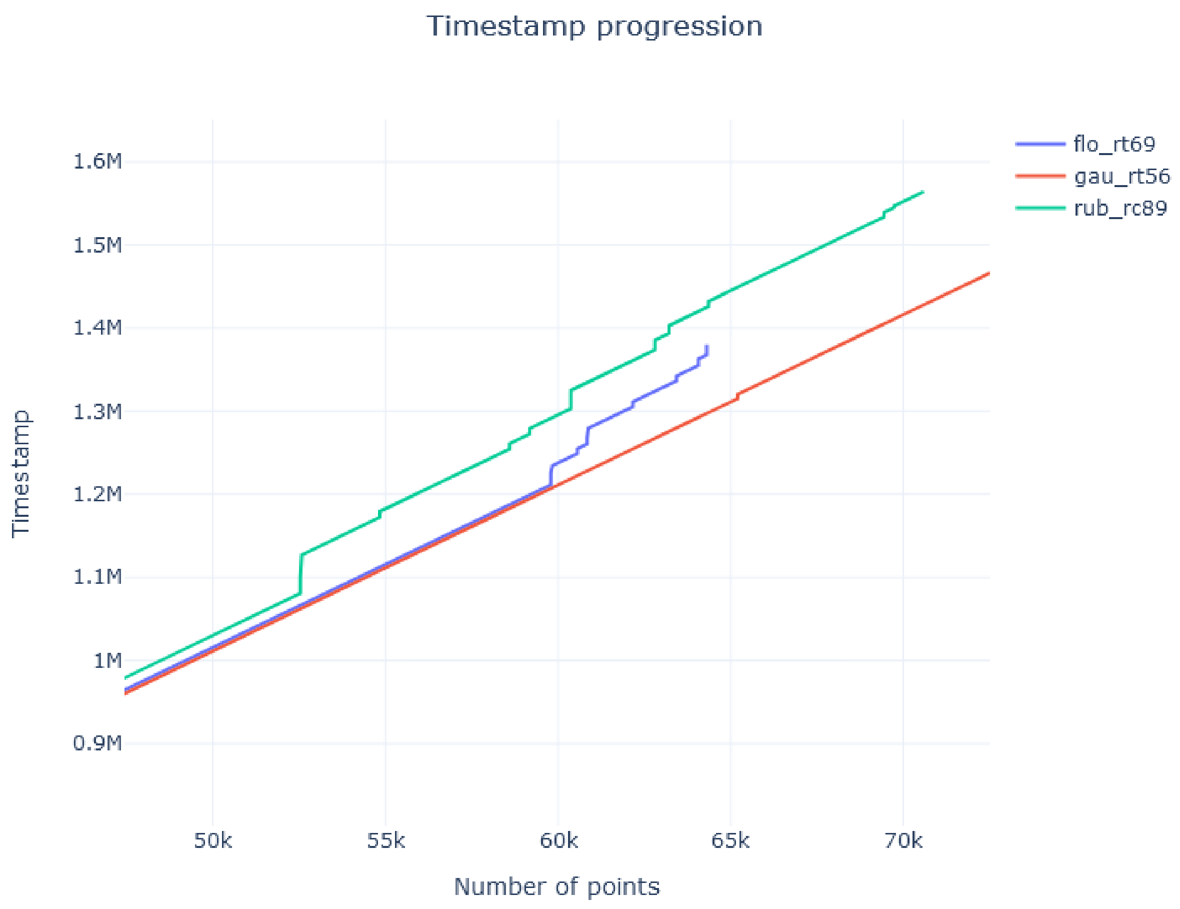
any_ot83	aug_ot66	aug_rt45	ben_ot80	ben_rt74
bre_ot61	bre_rt51	dav_oc91	dav_rc87	dav_ot81
dav_rt72	edo_ot58	edo_rt46	fab_rt50	fab_ot62
flo_ot77	flo_rt71	flo_ot78	flo_rt69	gau_rt56
gau_ot64	hel_oc90	hel_rc86	jar_ot82	jar_rt73
jea_ot84	jea_rt75	jen_ot63	jen_rt54	jul_ot59
jul_rt55	jer_ot85	jer_rt76	mic_ot79	mic_rt70
pp1_ot22	pp1_rt11	pp1_ot23	pp1_rt12	pp1_ot21
pp2_ot14	pp2_rt3	pp3_ot16	pp3_rt4	pp4_ot15
pp4_rt5	pp5_ot17	pp5_rt6	pp6_ot18	pp6_rt7
pp7_ot20	pp7_rt8	pp8_ot19	pp8_rt9	pp9_ot13
pp9_rt10	rub_oc93	rub_rc89	tib_oc92	tib_rc88
vin_ot65	vin_rt52	wil_ot60	wil_rt53	

6.1.2 Timestamp

This column contains the absolute time since the beginning of the flight session in milliseconds. Typical values vary from 0 to 1.5 million milliseconds (or 25 minutes) for a flight session.

By looking carefully, we noticed discontinuity in sampling. In the following graph, this phenomenon can be observed for some pilots. The discontinuity is represented by the steps in the timestamp value in the following image.

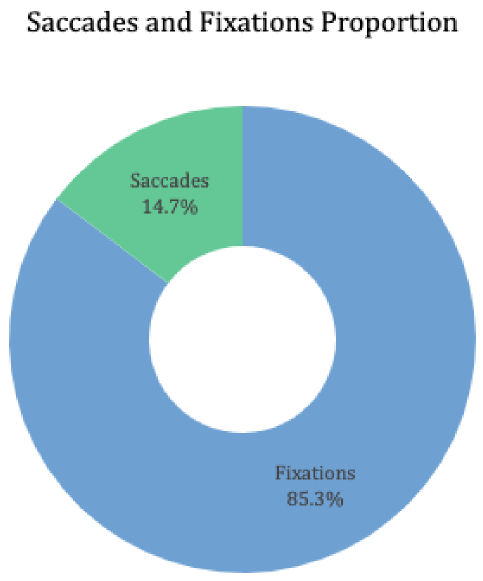
Figure 33: Timestamp discontinuity



6.1.3 Eye Movement Type

This column contains a Tobii® classification of the gaze movement based on their trademark IV-T Filter [9]. All data points receive one of four possible classifications: Saccade, Fixation, Eyes Not Found, and Unclassified. Saccades and Fixations have a clear definition, as mentioned in ??. However, Eyes not found and Unclassified data points can be ambiguous; one interpretation maybe a blinking event; another is a capture failure for excessive head movement or even eyelashes interference. Therefore, the data points with this classification are excluded from our dataset. Although, the majority of the data are fixation points.

Figure 34: Proportion of Eye Movement Types



6.1.4 Automap Score

The Tobii® glasses are attached to a pilot's head, and therefore subjected to movement. Automap is the automatic Tobii® mapping process from this non-inertial coordinates system to an inertial screen-fixed coordinates system. Tobii® glasses attributes a score to its automapping process.

6.1.5 Gaza and GazeY

These two columns contain the x and y position of the gaze point. The range of both is based on Garmin G1000 screen size: ranging from 0 to 1215 on X-axis and from 0 to 763 on Y-axis. Due to Eyes not found and Unclassified data points, there are 903,137 rows in both GazeX and GazeY columns with null values.

6.1.6 Flight Task

As stated in 4.1, the flight tasks start instants were classified by audio for each flight session. This is vital raw information since many features have task-wise aggregations, as detailed in section 7.

Task information is not present in the raw Tobii® database and it is merged with main data immediately after being read. Besides, some flight sessions do not have flight task data, hence, we remove from our data those pilots' flights, remaining with 39 flight sessions:

Table 8: Pilots remaining after Task drop

aug_ot66	aug_rt45	ben_ot80	ben_rt74	bre_ot61
bre_rt51	dav_ot81	dav_rc87	dav_rt72	edo_ot58
edo_rt46	fab_ot62	fab_rt50	flo_ot77	flo_ot78
flo_rt69	flo_rt71	gau_ot64	gau_rt56	hel_oc90
hel_rc86	jar_ot82	jar_rt73	jea_ot84	jea_rt75
jen_ot63	jen_rt54	jer_ot85	jer_rt76	jul_ot59
jul_rt55	mic_ot79	mic_rt70	rub_rc89	tib_rc88
vin_ot65	vin_rt52	wil_ot60	wil_rt53	

6.1.7 Target

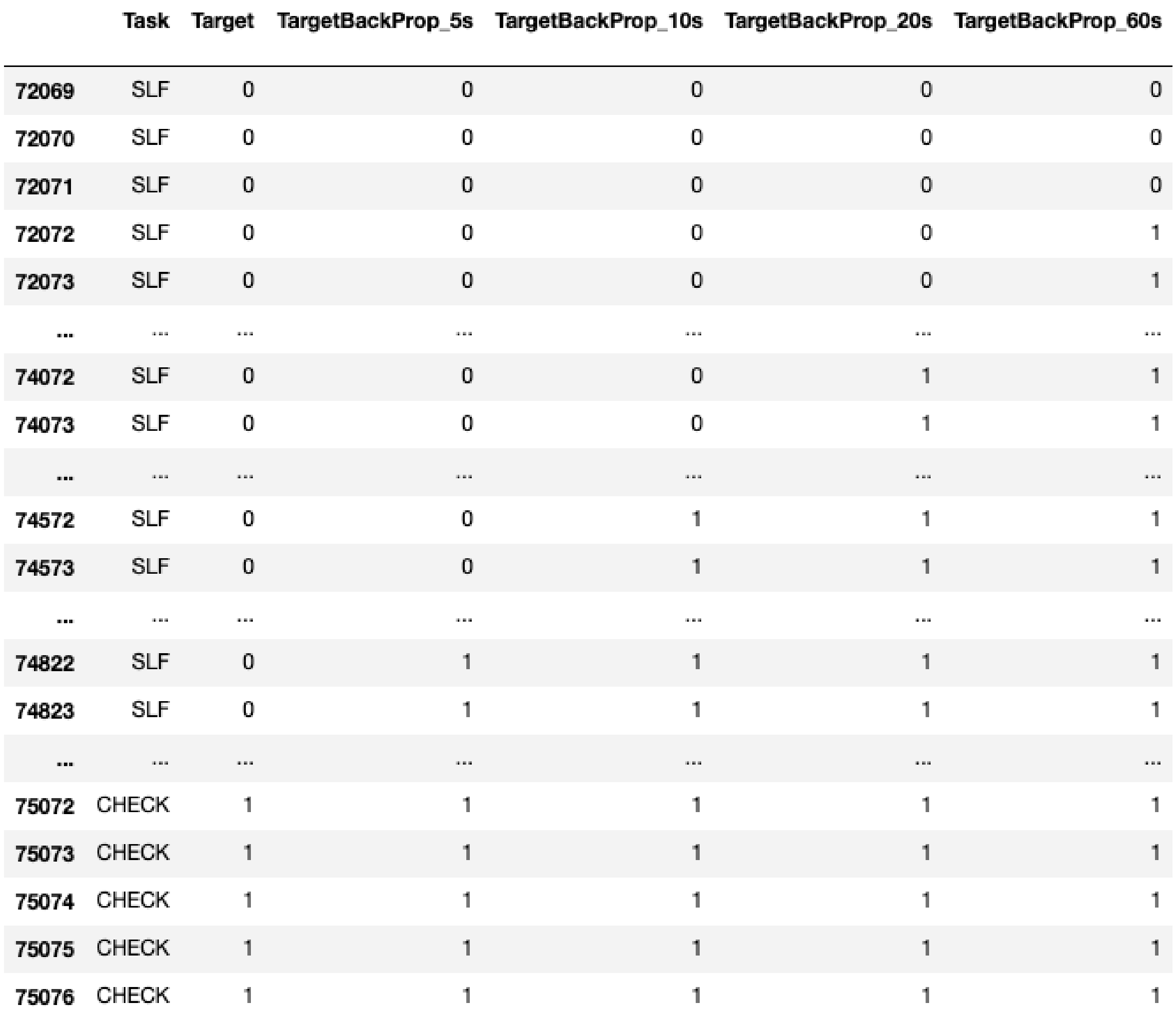
The target is generated by propagating backward in time a PEI as seen in the image 35. The period of back-propagation can be arbitrated. Different back-propagation periods are generated to determine the best combination of Task and Back-Propagation period. In the table 35 we can observe an example of Target creation.

6.2 Raw Data Table Processes

6.2.1 AOI

Based on the Garmin G1000 interface, seven regions were created, one for each AOI (ADI, ALT, BA, HDG, PWR, SPEED, and VS), and then we register each gaze point region. Technically, AOI is the first feature, but since this feature is very primal for other feature creation, we mention it here. The figure 36shows AOI's gaze points for a single flight session, which clearly demarks instruments in the Garmin instrument panel.

Figure 35: Back-propagation Target Columns



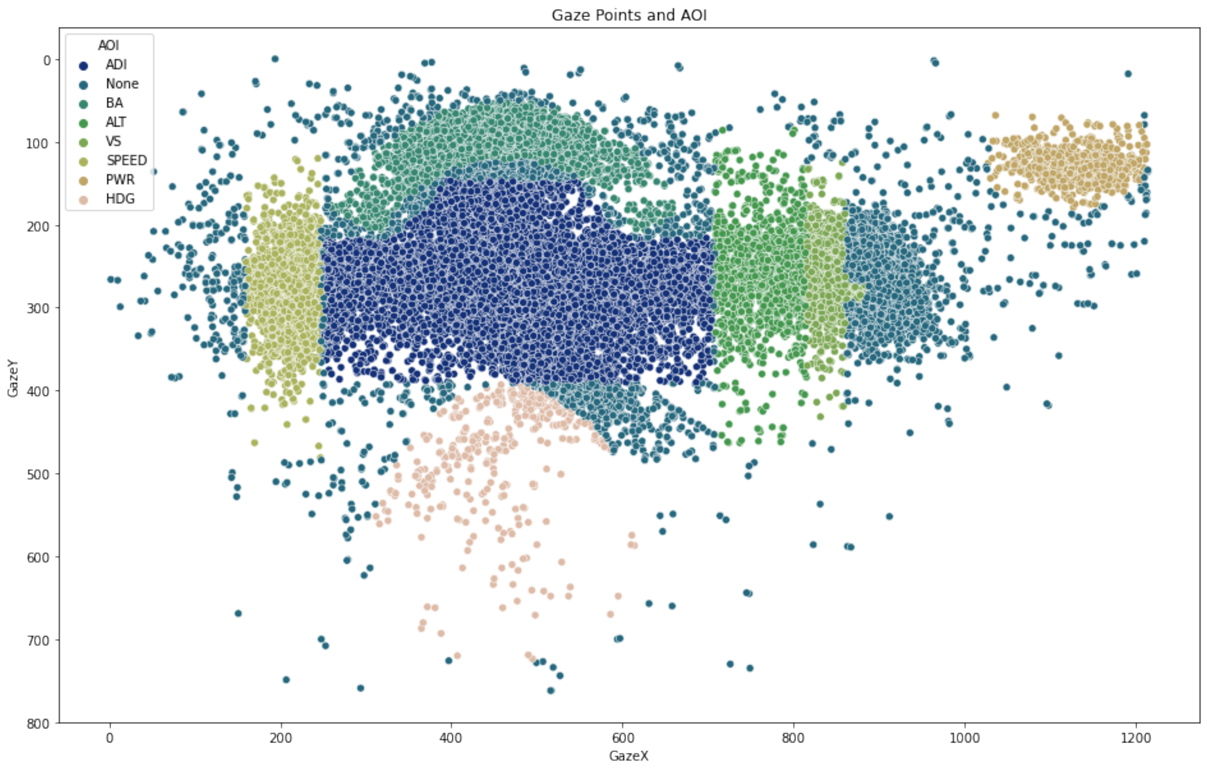
	Task	Target	TargetBackProp_5s	TargetBackProp_10s	TargetBackProp_20s	TargetBackProp_60s
72069	SLF	0	0	0	0	0
72070	SLF	0	0	0	0	0
72071	SLF	0	0	0	0	0
72072	SLF	0	0	0	0	1
72073	SLF	0	0	0	0	1
...	...	...	...	...	...	...
74072	SLF	0	0	0	1	1
74073	SLF	0	0	0	1	1
...	...	...	...	...	...	...
74572	SLF	0	0	1	1	1
74573	SLF	0	0	1	1	1
...	...	...	...	...	...	...
74822	SLF	0	1	1	1	1
74823	SLF	0	1	1	1	1
...	...	...	...	...	...	...
75072	CHECK	1	1	1	1	1
75073	CHECK	1	1	1	1	1
75074	CHECK	1	1	1	1	1
75075	CHECK	1	1	1	1	1
75076	CHECK	1	1	1	1	1

6.2.2 Data Removal and Fill

The data points with the 25% lower automap score are removed from the dataset to raise our data reliability. Additionally, null gaze points (Eyes not found and unclassified) and isolated are also dropped. A gaze point is considered isolated if the closest neighbor is farther than 50ms in time. 50ms is the same threshold used by Tobii® IV-T filter [9]. Also, tasks with less than 3 seconds of duration were removed because many features have windows of 5 and 10 seconds, and in this case, those features would present only null values. In this last case, not only some sparse points were removed, but the whole flight period.

After removing all this unreliable data, some null gaze points remained in the dataset: the non-isolated ones. Those points had their value-filled using linear interpolation based on its neighbors.

Figure 36: Gaze Point Coordinates Highlighted per AOI



6.2.3 Gaze Smoothing

Alongside the discontinuity and low-reliability problems, the gaze signal presented itself to be very noisy. We applied a moving average of 10 points for both x and y gaze coordinates to reduce noise. All master table contents are generated from the smooth gaze.

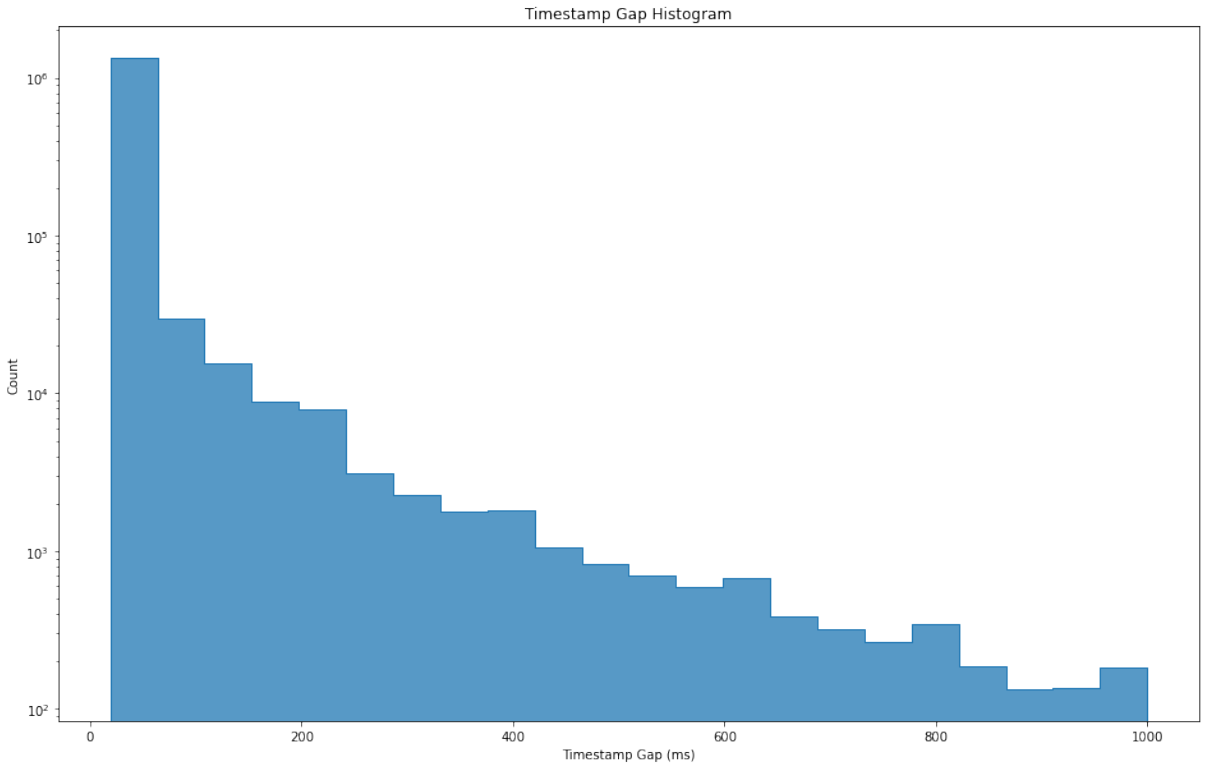
6.2.4 Sessions Splitting

Naturally, dropping the unreliable data introduces new time discontinuities and makes the time discontinuity problem (presented above in the Timestamp subsection) even worst. To ensure time continuity, each flight session is split into sub-flight sessions, called *parts*. The splitting is made when the gap between two consecutive points is greater than 1 second. Additionally, only parts with a duration longer than 10 seconds were kept. After splitting, we have 218 flight parts ids.

6.2.5 Data Resampling and Interpolation

After splitting the time between two consecutive gaze points is not constant; even its maximum value is now 1000 ms. The following histogram shows the counts for gap values between two consecutive data points.

Figure 37: Timestamp Gap Histogram after splitting



Clearly, 20 milliseconds is the most common sampling period value. We create new data points, forcing all data to be sampled 20 ms, which we call *resampling*.

After creating the data points' incorrect time positions, we interpolate GazeX and GazeY for them from existing neighbors. Once again, the interpolation used is linear. Spline interpolation was also tested but produced large distortions (such as negative gaze position values, which is absurd) when the gap is close to 1000 ms.

After all processes, we have an RDT containing the Spine columns, and therefore, in the same rows format of the spine.

7 FEATURES

As in the methodology section 4.2.1, the GAPE model has several features from which it will learn. *Flight* features calculation regards all past instants since the beginning of a flight session. *Task* aggregated features regard the elapsed time since the beginning of a flight task. *Windowed* features calculation regards a past interval of time, of arbitrary duration. Some specific windowed features are called autoregressive features.

Definition

An autoregressive feature is a feature with an elementary statistic aggregation, calculated over a time window, with a certain duration and lag.

Note that while all autoregressive features are windowed features, but not all windowed features are autoregressive.

Alongside the given features mentioned 6, new features were created, according to the following groups:

7.1 Gaze Movement

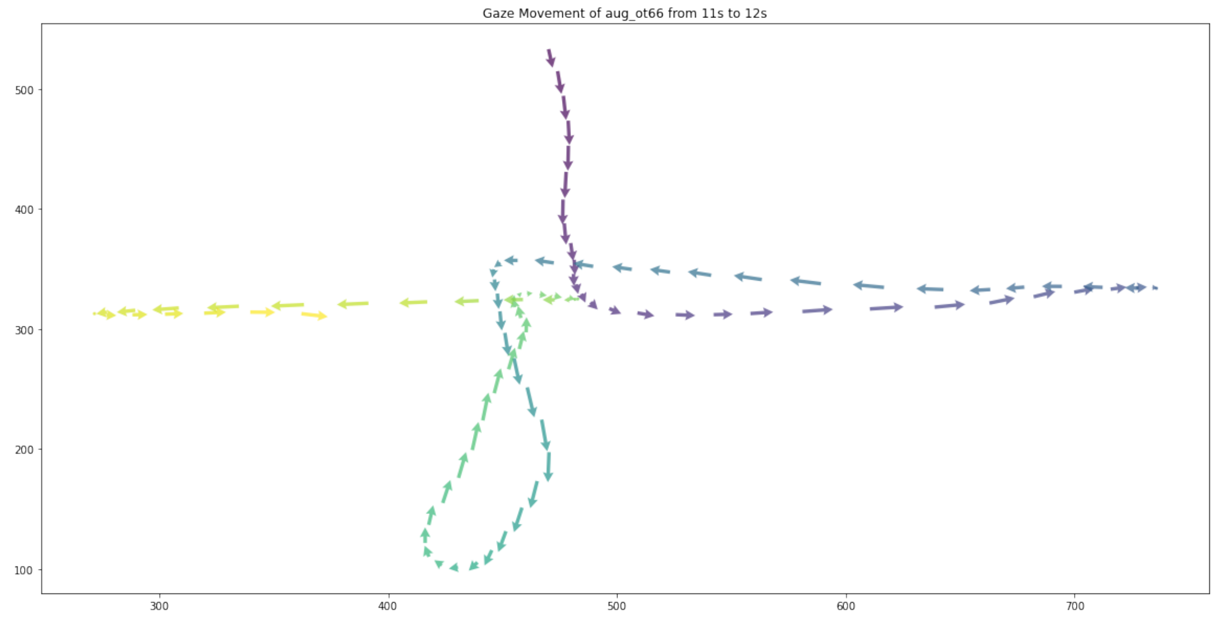
This set of features is directly related to gaze movement metrics, such as speed, traveled distance, and stopped time.

- **Gaze Position** is just the raw x and y coordinate of gaze point.
- **Gaze Speed** is the first time derivative of gaze position. The derivation is obtained through the central finite difference method. This feature has vertical, horizontal, and absolute versions, where the absolute represents the norm of the velocity vector defined by its horizontal and vertical components.
- **Gaze Acceleration** second time derivative of gaze position. The derivation is obtained through the second-order centered finite difference method. This feature has vertical, horizontal, and absolute versions.

- **Gaze Direction Change** is a flag indicating a change of movement direction. This feature has a version regarding only vertical movement, only horizontal and any change of movement.
- **Gaze Direction Change Frequecy** is the count of any movement direction changes in a given period, divided by the period duration. This feature has its flight-wise and some windowed versions.

Using both gaze position and gaze speed features allows us to observe a pilots' gaze trajectory in a certain time period.

Figure 38: Gaze Coordinates and Velocity during a Representative Flight Session



7.2 Dwell

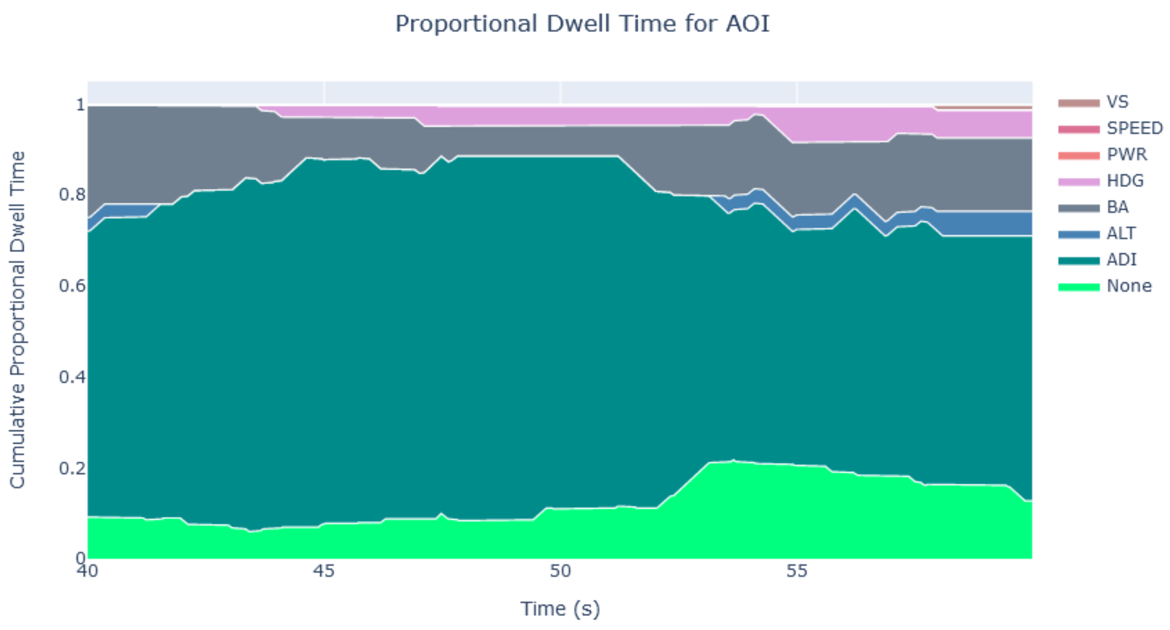
A dwell is the union of saccades and fixations within an AOI. When the AOI gazed upon changes, the dwell ends, and the next one starts.

- **Dwell Duration** is the duration of a dwell until the considered time instant. When the dwell ends, it resets to zero.
- **Dwell Duration Variance** is cumulative since the beginning of flight session variance of the dwell duration.

- **Dwell Count** is the number of dwells in a given period. This feature has flight, task, and windowed interval aggregations.
- **Dwell Frequency** is the dwelling count over the elapsed time in the same period.
- **Dwell Proportional Time** is the proportion of time dedicated looking an AOI. This feature has a version for each AOI and has flight, task, and windowed versions.

Here we can see an example of proportional dwell time (for all AOIs), for a pilot, in a time period.

Figure 39: Proportional Dwell Time during a Representative Flight Session



7.3 Fixations and Saccades

- **Fixation/Saccade Duration** is the cumulative duration of a gaze movement until the considered time instant. When the gaze movement ends, it resets to zero.
- **Fixation/Saccade Duration Variance** is the variance of gaze movement duration. It is calculated only regarding the whole flight. Item **Fixation/Saccade Traveled Distance** is the cumulative traveled distance within a gaze movement. When the gaze movement ends, it resets to zero.

- **Fixation/Saccade Traveled Distance Variance** is the variance of gaze movement traveled distance. It is calculated only regarding the whole flight.

- **Fixation Frequency** is the accumulated number of fixations, divided by the time elapsed time in a given period. The different periods considered for this feature are flight, task, and window.

- **Saccade Frequency** is the accumulated number of saccades, divided by the time elapsed time in a given period. The different periods considered for this feature are flight, task, and window.

8 MODEL

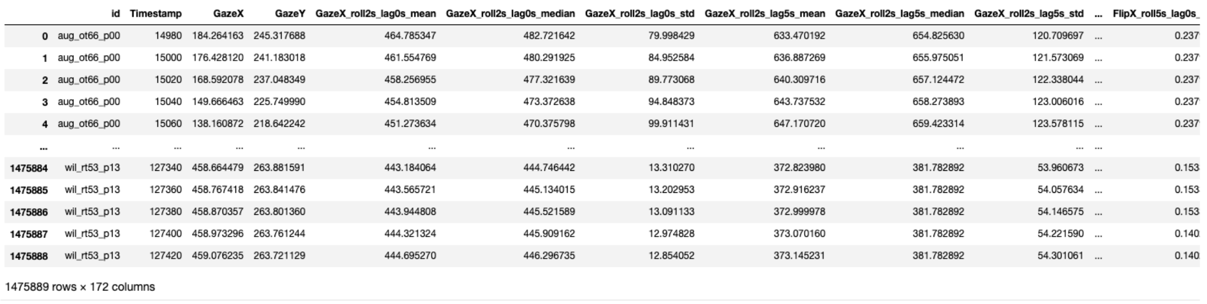
Definition

Dataframe: The primary Pandas [53] data structure. For two-dimensional, size-mutable, potentially heterogeneous tabular data.

8.1 Producing the Master Table

Producing a master table follows closely the steps outlined in the GAPE Framework section 4.2.4. It consists, essentially, of aggregating the Spine, Features (with auto-regressions), and Target variable in the same DataFrame. A table of all features is shown in annex D D. The fig 40 shows the master table after this merging step:

Figure 40: Final Master Table



	id	Timestamp	GazeX	GazeY	GazeX_roll2s_lag0s_mean	GazeX_roll2s_lag0s_median	GazeX_roll2s_lag0s_std	GazeX_roll2s_lag5s_mean	GazeX_roll2s_lag5s_median	GazeX_roll2s_lag5s_std	...	FlipX_roll5s_lag0s_
0	aug_ot66_p00	14980	184.264163	245.317688	464.785347	482.721642	79.998429	633.470192	654.825630	120.709697	...	0.237
1	aug_ot66_p00	15000	176.428120	241.183018	461.554769	480.291925	84.952584	636.887269	655.975051	121.573069	...	0.237
2	aug_ot66_p00	15020	168.592078	237.048349	458.256955	477.321639	89.773068	640.309716	657.124472	122.338044	...	0.237
3	aug_ot66_p00	15040	149.666463	225.749990	454.813509	473.372638	94.848373	643.737532	658.273893	123.006016	...	0.237
4	aug_ot66_p00	15060	138.160872	218.642242	451.273634	470.375798	99.911431	647.170720	659.423314	123.578115	...	0.237
...	...	...	...	...	...	...	...	...	...	...	...	
1475884	wil_rt53_p13	127340	458.664479	263.881591	443.184064	444.746442	13.310270	372.823980	381.782892	53.960673	...	0.153
1475885	wil_rt53_p13	127360	458.767418	263.841476	443.565721	445.134015	13.202953	372.916237	381.782892	54.057634	...	0.153
1475886	wil_rt53_p13	127380	458.870357	263.801360	443.944808	445.521589	13.091133	372.999978	381.782892	54.146575	...	0.153
1475887	wil_rt53_p13	127400	458.973296	263.761244	444.321324	445.909162	12.974828	373.070160	381.782892	54.221590	...	0.140
1475888	wil_rt53_p13	127420	459.076235	263.721129	444.695270	446.296735	12.854052	373.145231	381.782892	54.301061	...	0.140

1475889 rows × 172 columns

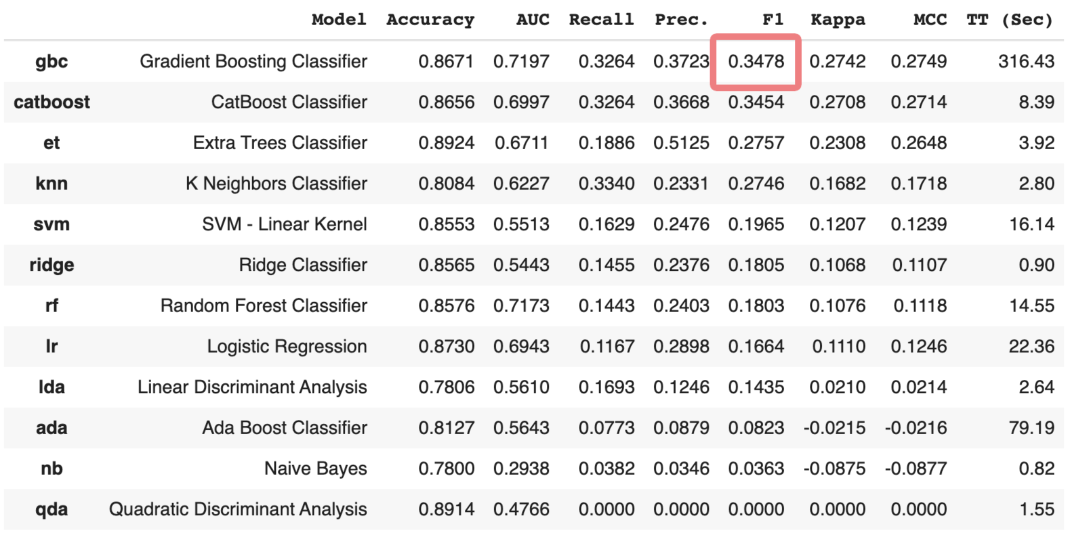
The important point is that, as mentioned in the section, 6.1.7several versions of the Target variable were created, with different windows of back-propagation (from 5 to 60 seconds). All these targets are included in the master table at this point so that the desired version can be chosen during the modeling in the following steps.

8.2 Fitting a Classifier

As mentioned in the 4.3.2.3, several methods offer varying responses and metrics to different machine learning problems. Support Vector Machines are particularly efficient with linearly separable data, while Random Forests offer high explainability. However, gradient boosting still provides more robust results for achieving high AUC and maximizing precision and recall. Thus, the algorithm of choice for the large part of the modeling section is **XGBoost**, stable and free implementation of Gradient Boosting Decision Trees for Python and other languages.

For the sake of analysis, a fitting of different classifiers is presented in the following table, showing the superiority of XGBoost in both AUC and F1 score. This particular table was created by providing the whole master table as input.

Figure 41: Comparison of different modeling algorithms



	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
gbc	Gradient Boosting Classifier	0.8671	0.7197	0.3264	0.3723	0.3478	0.2742	0.2749	316.43
catboost	CatBoost Classifier	0.8656	0.6997	0.3264	0.3668	0.3454	0.2708	0.2714	8.39
et	Extra Trees Classifier	0.8924	0.6711	0.1886	0.5125	0.2757	0.2308	0.2648	3.92
knn	K Neighbors Classifier	0.8084	0.6227	0.3340	0.2331	0.2746	0.1682	0.1718	2.80
svm	SVM - Linear Kernel	0.8553	0.5513	0.1629	0.2476	0.1965	0.1207	0.1239	16.14
ridge	Ridge Classifier	0.8565	0.5443	0.1455	0.2376	0.1805	0.1068	0.1107	0.90
rf	Random Forest Classifier	0.8576	0.7173	0.1443	0.2403	0.1803	0.1076	0.1118	14.55
lr	Logistic Regression	0.8730	0.6943	0.1167	0.2898	0.1664	0.1110	0.1246	22.36
lda	Linear Discriminant Analysis	0.7806	0.5610	0.1693	0.1246	0.1435	0.0210	0.0214	2.64
ada	Ada Boost Classifier	0.8127	0.5643	0.0773	0.0879	0.0823	-0.0215	-0.0216	79.19
nb	Naive Bayes	0.7800	0.2938	0.0382	0.0346	0.0363	-0.0875	-0.0877	0.82
qda	Quadratic Discriminant Analysis	0.8914	0.4766	0.0000	0.0000	0.0000	0.0000	0.0000	1.55

As expected, the higher performance of the XGBoost algorithm is shown, which suggests its use for the next sections. When utilizing a complex model such as this, various hyper-parameters are important and studied and chosen to maximize performance. Amongst the most important ones, in our case, were:

- Scale_pos_weight: control the balance of positive and negative weights, useful for unbalanced classes. A typical value to consider is the ratio of positive and negative instances:
$$\frac{\sum_{i=0}^{N} I_i(y=0)}{\sum_{i=0}^{N} I_i(y=1)}$$
,

  where i is a sample, y is the target value of a sample, N is the number of samples, $I(x = \dot{x})$ the identity function, and yields is 1 when $x = \dot{x}$ and 0 otherwise.

- Learning_rate: step size shrinkage used in the update to prevents overfitting. After each boosting step, we can directly get the weights of new features, and eta shrinks the feature weights to make the boosting process more conservative.

- Max_depth: maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit.

8.3 Train Test Splitting

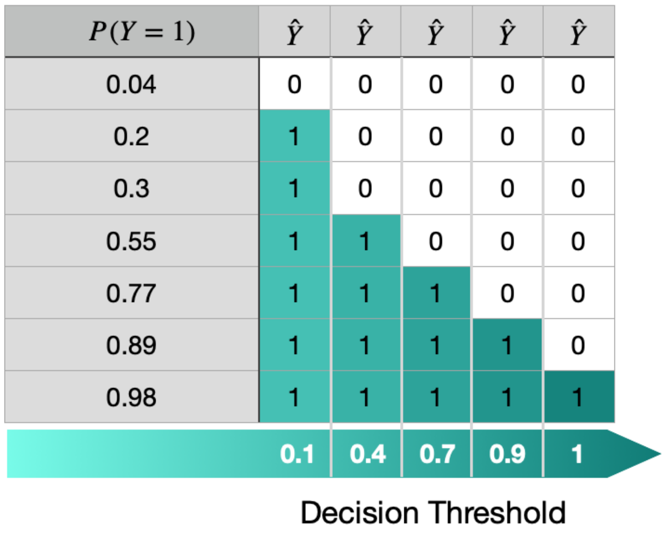
The next step when creating a model is to split the data into train and test sets. For this, the **Time-series Cross Validation** method explained in the section 4.3.5.4s used. A custom function was implemented to ensure the test set is in the future compared to the train set and has an adequate gap (5-20% larger than the back-propagation window) to the train data.

8.4 Model usefulness and Tuning

Typically in a Machine Learning problem, the threshold (defined in section 4.3.1) of a classifier is set as $D_t = 0.5$, which takes care of most situations in which class imbalance is not particularly strong. In our case, however, class imbalance is a strong matter, with values ranging from between $class_{imb} = 5 - 20\%$. Thus, moving the decision threshold allows for precise control over where the model starts to identify positive classes. The figreffig: threshold illustrates the process:

For example, the precision-recall curve is created by progressively varying the classification boundary and calculating the model's recall and precision at each step. An example

Figure 42: Preliminary Model Master Table Snapshot. (Source: Author).

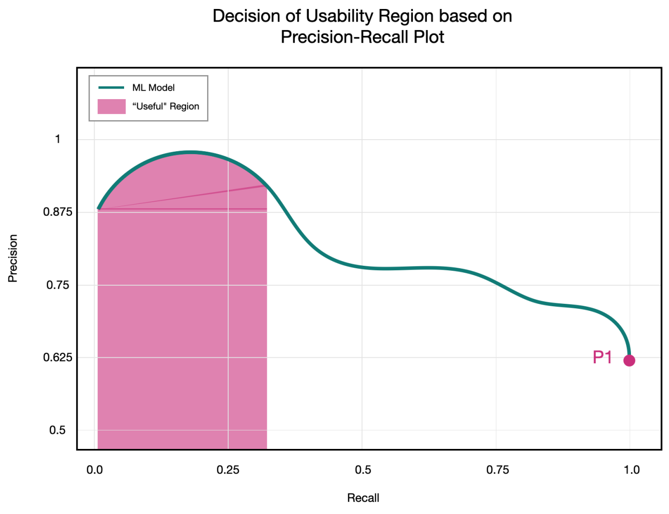


$P(Y=1)$	$\hat{Y}$	$\hat{Y}$	$\hat{Y}$	$\hat{Y}$	$\hat{Y}$
0.04	0	0	0	0	0
0.2	1	0	0	0	0
0.3	1	0	0	0	0
0.55	1	1	0	0	0
0.77	1	1	1	0	0
0.89	1	1	1	1	0
0.98	1	1	1	1	1
	0.1	0.4	0.7	0.9	1

Decision Threshold

of such a curve is shown in Figure 43. An important note is that, as we move to the right on the horizontal axis, the decision threshold is progress,“ very ”relaxed,“ which allows the model to classify more and more data points as positive. When a recall is maximum, highlighted as point P1, the model essentially classifies all data points as positive. At this point, trivially, the precision will be equal to the dataset imbalance itself.

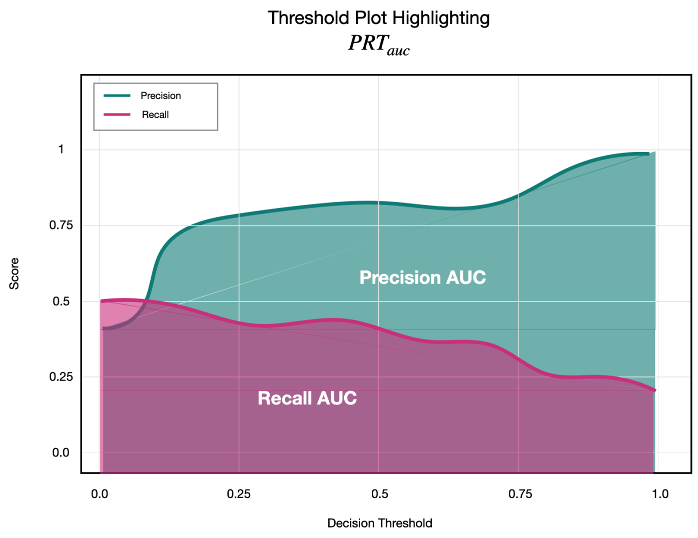
Figure 43: Example of Precision Recall curve



Our interest, as researchers, is to get a model that maximizes precision while pushing recall as far as possible, or, in other words, to only predict errors when they, in fact, will happen (high precision), and to also capture the most amount of errors that could happen (high recall). While that is not always possible, a model can still be considered useful to which has relatively high precision even if that is on a specific ”region“ of the precision-recall curve. The aim 43 above also highlights in pink a part of the graph that suits this criterion.

Once this region of interest is defined, the next important step is to find what classification threshold yields such results. For that, a less common plot, usually called a threshold curve plot, can be used. The figure 44shows such a curve, where the precision and recall plots are done for classification boundary from 0 to 1. The same region of interest is highlighted in pink, and in this case, it is possible to identify the actual value of the threshold that provides optimal performance. It is interesting to note that a model that would show a high Area Under the Curve (AUC) in both precision and recall curves could be considered a good performance model for our purposes.

Figure 44: Example of model prediction for different classification thresholds



In the interest of finding such good performance models, a new metric, named PRT_{AUC} is proposed by the authors, like the following:

$$PRT_{AUC} = Precision_{AUC} + Recall_{AUC} \tag{8.1}$$

Maximizing such metrics will be an objective of the next sections.

8.5 Choosing RMTs and a Modeling Subset

At this point, the Master Table can be filtered by Flight Task to generate each RMT. Basically, given an RMT, one choice remains before fitting the data with an algorithm, the

Table 9: RMT Sizes

Task	Size (rows)
COMBO	436204
RO+LO	70665
ROLL-OUT	51089
SLF	426995
T0	5580
TURN	120999
VERT	528453

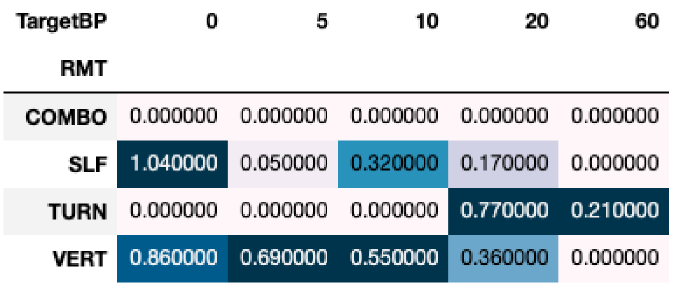
target variable backpropagation. Different backpropagation values could produce better models for different flight tasks. Thus, a comparison of different back-propagation for each RMT was made to choose the best combination in each case.

Each RMT is a filter of the Master Table for a given flight task. Some metrics of each RMT are presented below:

When considering the above Table, an important point is related to the available amount of data and the limitations of pursuing a RO and RO+LO flight task model. After the aforementioned pre-processing steps were complete, the remaining amount of data for the RO and RO+LO reduced master tables was only 102 seconds and 141 seconds. After some testing, these values were deemed too small to consider the fitting of a model, as the relatively small number of rows coupled with the class imbalance of the problem made the number of positive classes shown to the model too scarce. Thus, a decision was reached to move on only with the SLF, TURN, VERT, and COMBO reduced master tables in the next step.

The chosen evaluation metric for the success of an (RMT, back-propagation pair) is the PRT-auc detailed in the previous section - as it takes into account the precision and recall of the resulting model. Fitting an XGBoost classifier to each RMT with a specific back-propagated target yields the image results 45.

Figure 45: Results for RMT and Back-propagation pairs



TargetBP	0	5	10	20	60
RMT					
COMBO	0.000000	0.000000	0.000000	0.000000	0.000000
SLF	1.040000	0.050000	0.320000	0.170000	0.000000
TURN	0.000000	0.000000	0.000000	0.770000	0.210000
VERT	0.860000	0.690000	0.550000	0.360000	0.000000

Observing the results above, two main conclusions can be taken:

- VERT, TURN and SLF have acceptable PRT-auc values and show some optimal choices of Target back-propagation that maximize the model performance
- COMBO, RO, and RL demonstrate low PRT-auc for all variations of target back-propagation. This suggests that these are flight tasks for which the ability to produce a usable model is questionable.

This scenario offers the unfortunate conclusion that not all proposed models will show good results. While this is not ideal, a point can be made to the positive aspect of breaking down the problem into small subparts, as proposed in the Methodology. Doing this allowed us to have some failed models yet still obtain some successful ones that will be invested on.

An important discussion about the overall quality of results is presented further, but some possible factors, particular to the RO, RO+LO, COMBO models, include:

- Little amount of data: the RMTs corresponding to these three models is also the ones with the least data-points amongst all. In a Machine Learning problem, as famously put by Halevy et al. [10].

  Invariably, simple models and many data trump more elaborate models based on less data

  -- Alon Halevy, The Unreasonable Effectiveness of Data

- Stronger class imbalance: class imbalance varies according to the chosen dataset and the amount of positive versus negative labels in the sample. For COMBO, the positive class represents only 2,42%, while RO+LO and COMBO are also amongst the highest.

The next step is the actual fitting and evaluation of model performance for the RMTs and Targets that showed success thus far. These pairs and the name was given to each of the models to be produced are:

- Task: TURN, Target Back-propagation of 20s Name: TR20
- Task: SLF, Target Back-propagation of 10s Name: SL10
- Task: VERT, Target Back-propagation of 5s Name: VT0

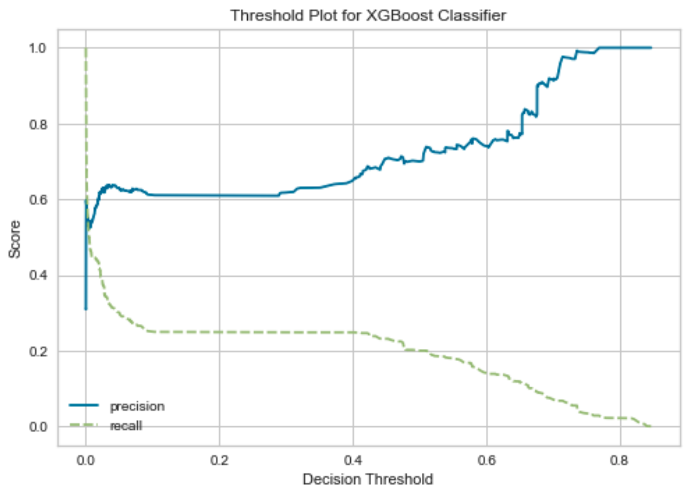
The process of fitting, tuning, and evaluating each of these models is presented in the following sections.

8.6 TR20 Model

An XGBoost classifier is trained on the TURN-RMT with a Back-propagated target of 20 seconds. After tuning and several experiments, a set of hyper-parameters was defined and used for the next steps. For reproducibility, the specific hyper-parameters are included in Annex A. With such a model fitted, the performance, as well as the decision of classification boundary and interpretation of this model is shown below

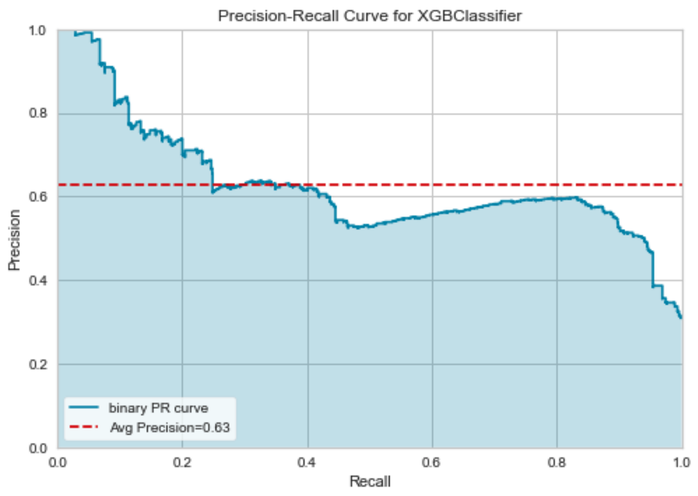
8.6.1 Threshold, Precision, and Recall

Figure 46: TR20 Threshold curve. (Source: Author).



The curve presents good results, with high precision overall, but remarkably high precision for thresholds from 60% to 80%. While holding high precision, the recall is around 15% in this same region. The relation between precision and recall is shown below:

Figure 47: TR20 Precision Recall Curve. (Source: Author).

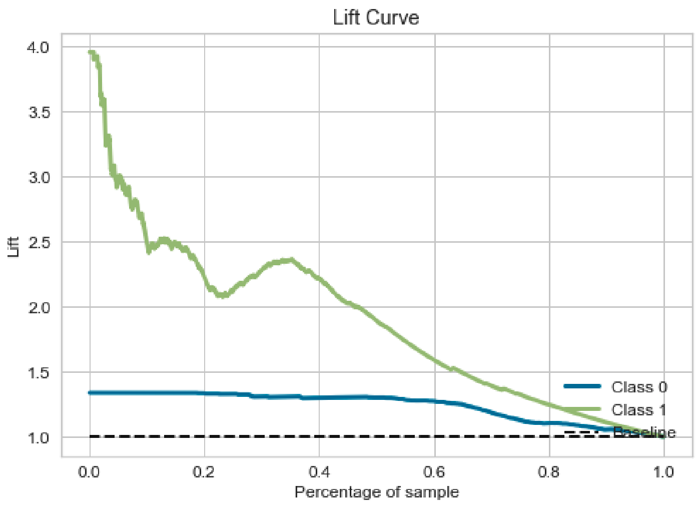


At this point, the decision threshold of this classifier is set to:

$$D_{t,TR20} = 61\%$$

8.6.2 Lift

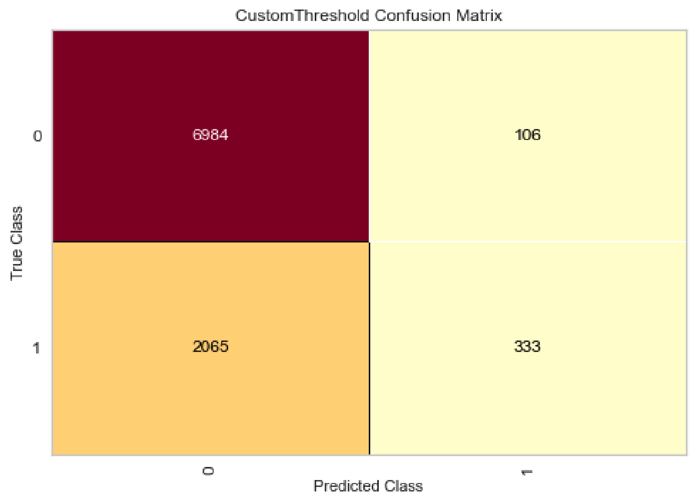
Figure 48: TR20 Lift Curve. (Source: Author).



The lift curve - detailed in the section 4.3.3 - reflects how much the model's performance surpasses a random guess for the same classification problem. The plot reveals a positive result, with a remarkable lift value of around 2.5 to 4 times, high for general ML problems.

8.6.3 Confusion Matrix

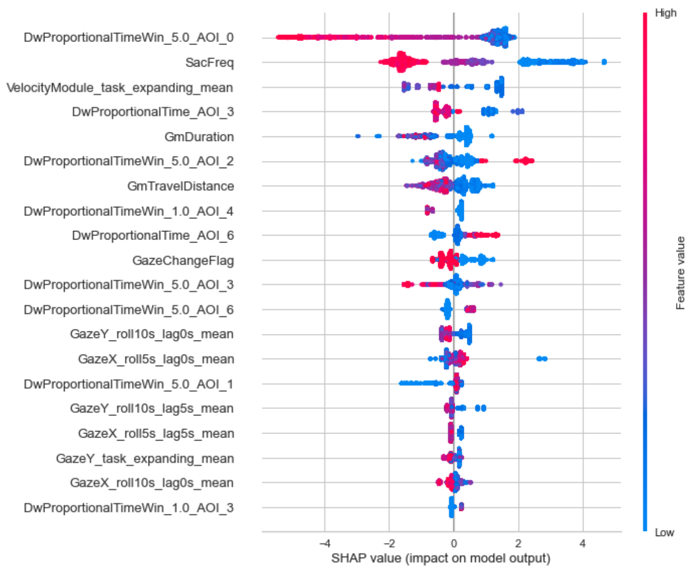
Figure 49: TR20 Confusion Matrix. (Source: Author).



The confusion matrix - detailed in the section 4.3.3 - demonstrated the model's ability to correctly classify positive labels. It is important to notice that tweaking the scale_pos_weight parameter was done to apply stronger penalties to a misclassification of the positive class, so the number of False Negatives is minimized.

8.6.4 SHAP Values

Figure 50: TR20 SHAP Values. (Source: Author).



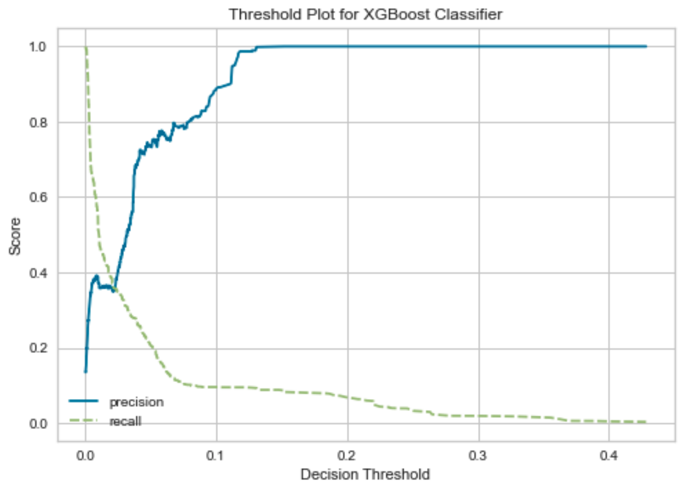
The SHAP values - detailed in the model interpretation section 4.3.7 - show an important aspect of the algorithm's classification solving: it demonstrates which features are considered the most important and how their values impact the model itself. For instance, it is noticeable that high Proportional Dwell Times outside of any important AOI impact the model substantially, pushing the chances of a performance error. At the same time, high Saccade Frequencies do the same.

8.7 SL10 Model

An XGBoost classifier is trained on the SLF-RMT with a Back-propagated target of 10 seconds. After tuning and several experiments, a set of hyper-parameters was defined and used for the next steps. For reproducibility, the specific hyper-parameters are included in Annex A. With such a model fitted, the performance, as well as the decision of classification boundary and interpretation of this model is shown below

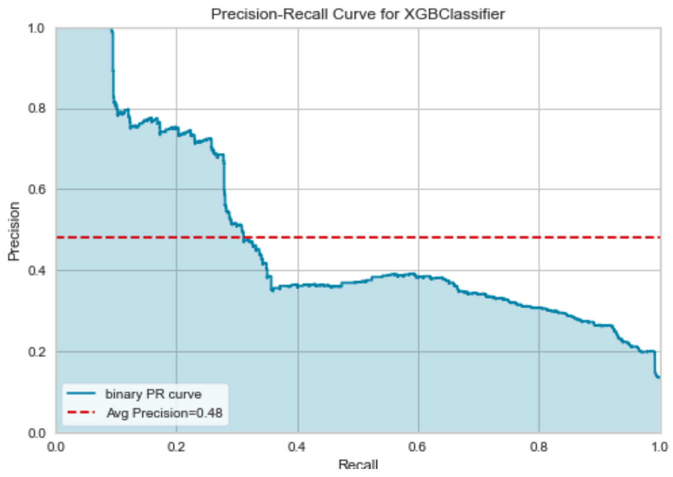
8.7.1 Threshold, Precision, and Recall

Figure 51: SL10 Threshold curve. (Source: Author).



The curve presents good results, with high precision overall, but even higher precision for thresholds from 5% to 10%. While holding high precision, the recall is around 20% in this same region. The relation between precision and recall is shown below:

Figure 52: SL10 Precision Recall curve. (Source: Author).

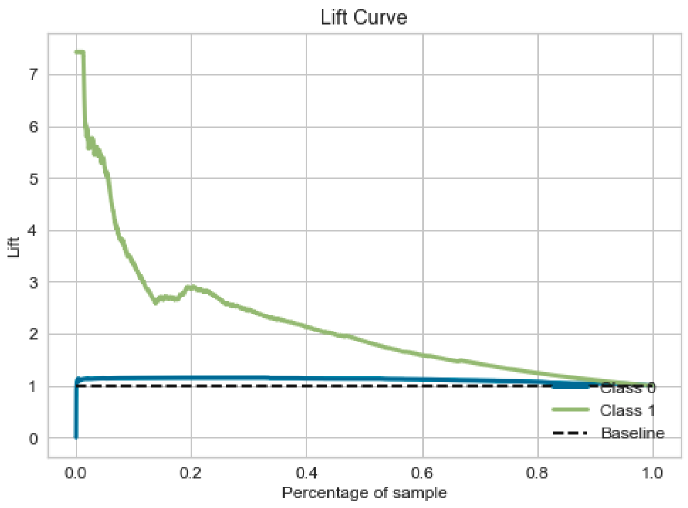


At this point, the decision threshold of this classifier is set to:

$$D_{t,SL10} = 5,65\%$$

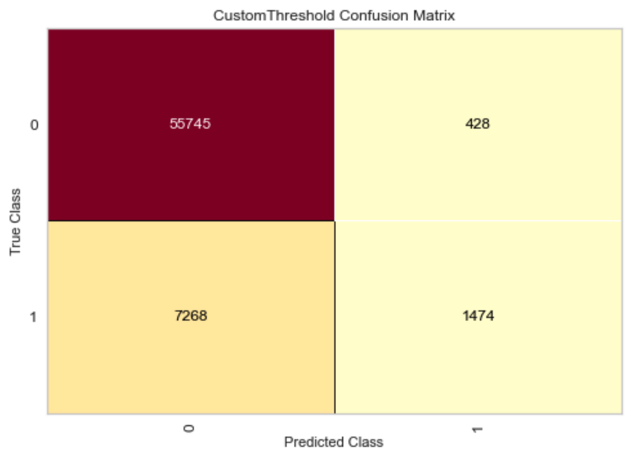
8.7.2 Lift

Figure 53: SL10 Lift Curve. (Source: Author).



The lift curve reflects how much the model's performance surpasses a random guess for the same classification problem. The plot reveals a positive result, with a remarkable lift value of around 5 to 7, which is very high for general ML problems.

Figure 54: SL10 Confusion Matrix. (Source: Author).

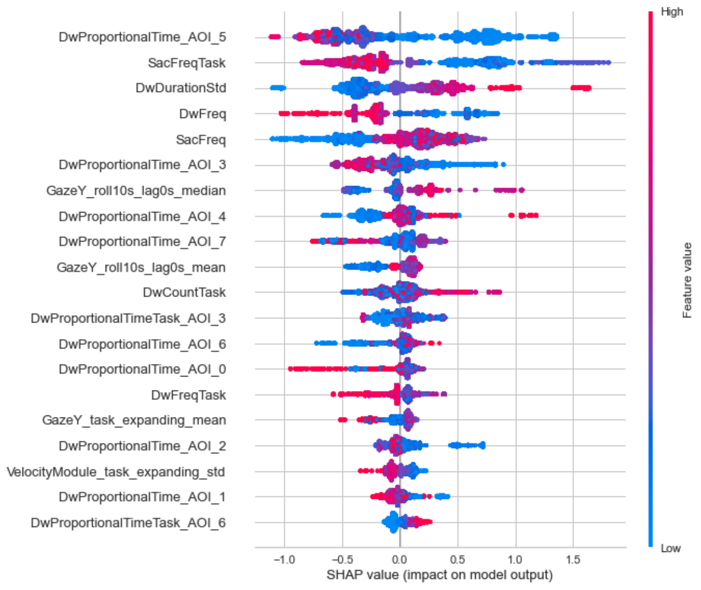


8.7.3 Confusion Matrix

The confusion matrix demonstrated the ability of the model to classify positive labels correctly. It is important to notice that tweaking the scale_pos_weight parameter was done to apply stronger penalties to a misclassification of the positive class, so the number of False Negatives is minimized.

8.7.4 SHAP Values

Figure 55: SL10 SHAP Values. (Source: Author).



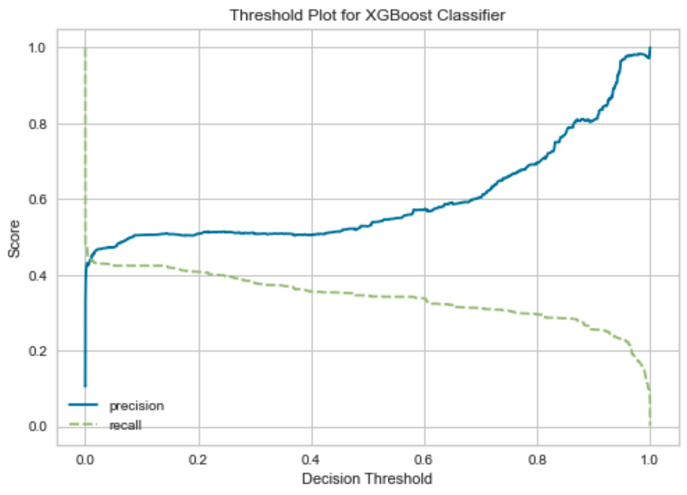
The SHAP values - detailed in the section 4.3.3 - show an important aspect of the algorithm's classification solving: it demonstrates which features are considered the most important and how their values impact the model itself. For instance, it is noticeable that high Proportional Dwell Times in the PWR instrument substantially impact the model, pushing the chances of a performance error. At the same time, high Saccade Frequencies within SLF tasks do the same.

8.8 VT0 Model

An XGBoost classifier is trained on the SLF-RMT with a Back-propagated target of 10 seconds. After tuning and several experiments, a set of hyper-parameters was defined and used for the next steps. For reproducibility, the specific hyper-parameters are included in Annex A. With such a model fitted, the performance and the decision of classification boundary and interpretation of this model are shown below.

8.8.1 Threshold, Precision, and Recall

Figure 56: VT0 Threshold Curve. (Source: Author).



The curve presents good results, with high precision overall, but remarkably high precision for thresholds from 75% to 90%. While holding high precision, the recall is around 30% in this same region. The relation between precision and recall is shown below:

At this point, the decision threshold of this classifier is set to:

$$D_{t,VT0} = 89,5\%$$

Figure 57: VT0 Precision Recall Curve. (Source: Author).

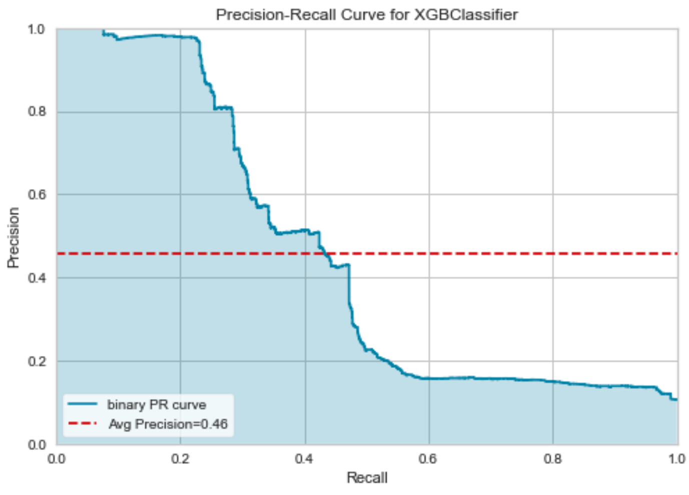
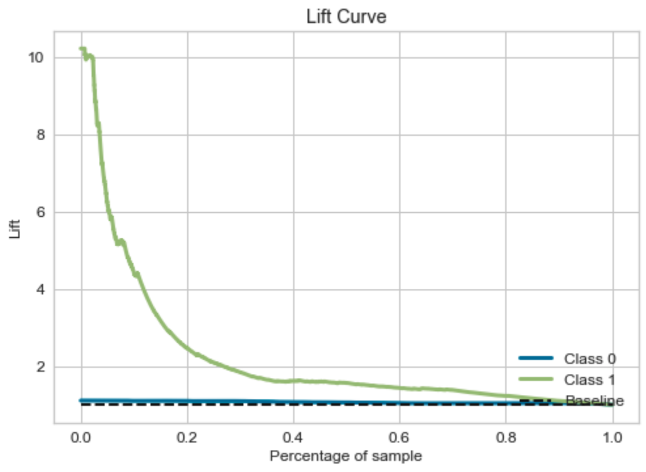


Figure 58: VT0 Lift Curve. (Source: Author).



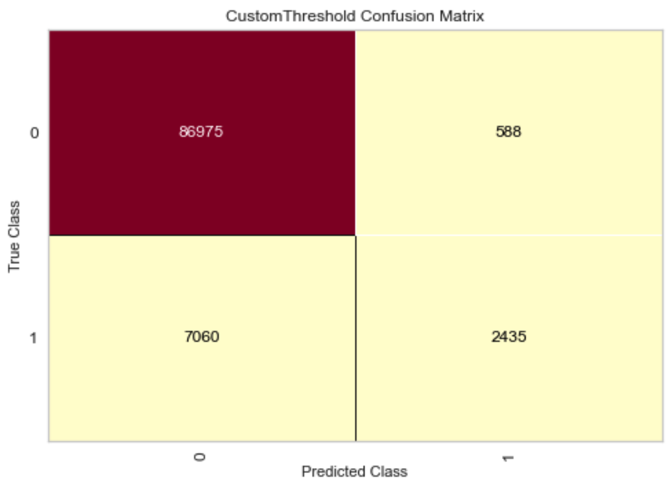
8.8.2 Lift

The lift curve - detailed in the section 4.3.3 - reflects how much the model's performance surpasses a random guess for the same classification problem. The plot reveals a positive result, with a remarkable lift value of around 6 to 10, high for general ML problems.

8.8.3 Confusion Matrix

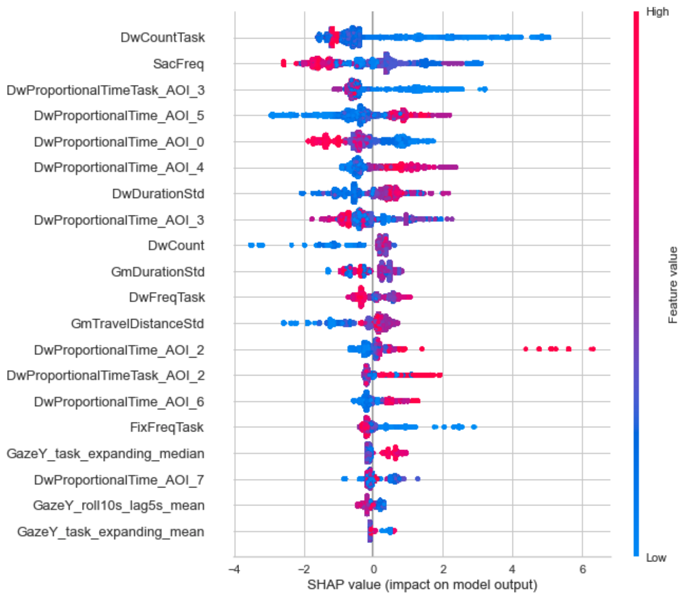
The confusion matrix demonstrated the ability of the model to classify positive labels correctly. It is important to notice that tweaking the scale_pos_weight parameter was done to apply stronger penalties to a misclassification of the positive class, so the number of False Negatives is minimized.

Figure 59: VT0 Confusion Matrix. (Source: Author).



8.8.4 SHAP Values

Figure 60: VT0 SHAP Values. (Source: Author).



The SHAP values - detailed in the section 4.3.3 - show an important aspect of the algorithm's classification solving: it demonstrates which features are considered the most important and how their values impact the model itself. For instance, it is noticeable that a high low Dwell number of dwells within the vertical tasks substantially impacts the model, reducing the chances of a performance error. At the same time, high Saccade Frequencies increase the chance of error.

PART IV

DISCUSSION AND CONCLUSION

For various fields of human activity, forecasting future events of failure or accident is the utmost objective. Researchers look for methods to assess and prevent problems before they occur, from machines breaking down to stock market crashes. Machine Learning has risen as one of the most prominent contenders in this context, often providing more accurate results than previously used statistical or deterministic models. Thus, it is a fruitful field of exploration, and these advanced analytics methods have not yet achieved their highest maturity or even became commonplace in the whole industry. This paper intended to push this boundary of applications where Machine Learning is used by applying it to Gaze Behavior analysis, specifically in Aeronautics.

As mentioned, the benefits of making real-time, in-flight predictions of pilot performance errors, merely from tracking their eyes over the cockpit, could provide more safety and reduce accidents - largely caused by human error. More importantly, maybe, this research can provide a basis for future works were the connection between performance and Gaze Behavior via the use of Machine Learning methods that can be reproduced in different fields.

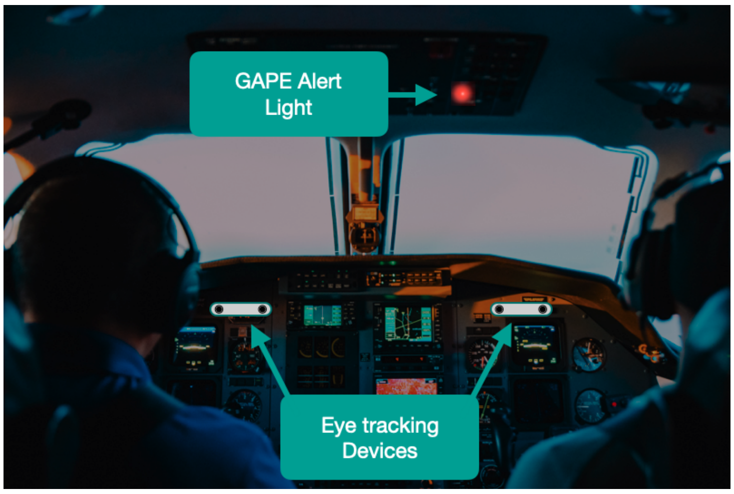
Initially, we present an overview of the context of Gaze Behavior research in aviation. A historical review is presented, highlighting important developments in the gaze analysis techniques and the state-of-the-art in the field, including incipient initiatives with Machine Learning and artificial intelligence. We also take the chance to explain the motivation behind research in this field by presenting the importance of pilot information gathering in the cockpit.

A novel methodology is proposed to produce a Machine Learning model trained only on Gaze variables. It can predict performance errors during specific flight tasks (straight and level flight, turn, vertical maneuvers). Considering a real experimental case from KU Leuven, all the necessary steps are explained, from data gathering, processing to fitting a model via selected algorithms, and interpreting results.

Finally, applying such methods to the gathered data, we trained three ML models for predicting performance errors in straight and level flight, vertical maneuvers, and turn maneuvers, respectively. Their performance metrics are presented and interpreted, which indicated successful prediction ability in the three cases. Although this is a subset of the initially proposed models, we consider such results a gratifying achievement, given their real-world potential.

Based on the resulting models obtained, a real-life application with in-flight detection and warning of possible accidents can certainly be envisioned and merits a part in this paper's concluding section. As mentioned in Results, it was possible to obtain some high-precision-low-recall models, which would essentially mean that a warning system, while not capturing all possible errors in a flight, when the system predicted an error, it would do so with high precision and could, for instance, produce a light or sound signal in the cockpit to warn the Pilot. The figreffig:gape$_alertshowsasketchofsuchasystem$:

Figure 61: GAPE Alert System Suggestion of Implementation. (Photo by Caleb Woods [56])



This would certainly require real-time eye-tracking, but modern externally mounted optical-tracking sensors could be installed on the cockpit facing the pilot, not taking much space. Such a system would allow pilots to have an additional monitoring device that alerts and reduces the chances of accidents, especially those directly related to lack of information capturing from the cockpit instruments. Eye-tracking devices fitted in cars to detect sleepy drivers [20] show an example of an early-stage application of this concept.

An additional yet still important comment on the results obtained by this paper is related to a deeper understanding of what are the most relevant variables (amongst the tested ones) for a Machine Learning algorithm to extract valuable data from Gaze Behavior. As shown in section 4.3.7, the Feature Importances of the given models show how much they influence the model's result and classification capacity. This ranking of features increased our ability to tune the model further, understand its behavior, and hold value for other researchers. The features presented here as most important could have a higher priority on designing future Gaze Behavior work and experiments.

Throughout the work presented in this paper, the importance of correct information extraction in the cockpit was explored, a novel machine learning methodology was proposed, with a robust framework for model creation, and a dataset of a real-life experiment was utilized to fit and produce classification models to predict pilot performance error based solely on gaze behavior. This process proved a profound and rewarding dive into the mechanisms of modern advanced analytics, as well as the power they can provide when tackling complex real-world problems.

9 FUTURE WORK

This section will describe possible improvements for the GAPE model, other than constructing a pilot warning device, as mentioned in the previous section. Those improvements should raise model reliability and performance.

9.1 Gaze Data

As mentioned in the section 5, in this dataset, Gaze Data is sparse in some time intervals, and this makes a flight session discontinuous. Discontinuity in-flight sessions can be solved by recording gaze data from new flight sessions. The recording must present constant data density over time.

Furtherly, the addition of information on blinking events can make the model more robust by, generate new features, such as mean blink duration and blink frequency (in flight, task, and window), and creating confidence in time sampling regularity, because blinking instants will appear explicitly appear in time instants, in opposition to the ambiguous *Eyes not Found* present in our data.

Nonetheless, recordings from pilots in real flight, other than just simulated sessions, can improve model connection with reality. Thus, real flight data could be another improvement in data.

9.2 Target Data

Once again, as mentioned 5, the flight sessions' performance data does not cover all the flight instants and is formatted as an image. For a full GAPE method application, the performance data should be collected in tabular data, continuously over the flight and with absolute time and the same initial instant as gaze data, enabling coupling of gaze data and performance data. Once again, new experiments should be realized to obtain such data.

Therefore, we consider the perfect data for the GAPE method with the following characteristics:

- Gaze data and performance data with constant sampling rate overall flight time
- Gaze data with blinking events information
- Gaze data collected in real flight data
- Performance data measured in absolute timescale and matching gaze data initial instant

9.3 New Features

To improve model performance, we suggest the creation of new features:

Order of Attention

- **First, Second, and Third AOIs looked** within a flight task
- **Number of executions of a known pattern** is counting known scan patterns, such as T-Scan and Artificial Horizon to Power. This feature should have regular, task, and windowed variations. This feature should have variations for each scan pattern. Identifying a scan pattern may be a challenge; a simple approach we suggest defining an ideal behavior (in terms of x and y gaze position) of each pattern and then use a Dynamic Time Warping algorithm (there are python packages for it) to score the resemblance between the measured gaze x and y behavior to the ideal one. A more sophisticated way to identify such patterns is to understand a collection of subsequent gaze points as images and then use a Deconvolutional Neural Network (DNN) to identify the scan patterns (DNN is an artificial intelligence technique identify patterns in images).
- **Frequency Trajectories** is the number of executions of each identified pattern, divided by the time elapsed since the beginning or in the time interval. This should have flight, task, windowed time interval variations.

Complex features

- **Entropy**, calculated as
$$-\sum_{i=1}^{n} P(x_i).log_2(P(x_i))$$
, where $P(x_i)$ is the probability of occurrence of a given gaze point (in bits, as seen in [7]). This probability is, of course, estimated within the training dataset. This feature has regular and windowed variations.

- **Gaze Quality** is a raw data from Tobii Pro ® [8] that score the reliability of the gaze metric itself. This feature has autoregressive variations.

- **Eye Movement Energy**, is the classical mechanic's movement energy, calculated for the pilot eyes.

- **Proportion Long Gazes**: first, we classify the gaze movement as long or short. Then we calculate the proportion of occurrence of each type. This feature can have aggregation by flight, task, and window.

- **AOI area overlap with foveal area** as calculated in Brams [15] work.

BIBLIOGRAPHY

[1] 2020. URL: https://www.tobiipro.com/product-listing/tobii-pro-glasses-2/.

[2] 2020. URL: https://www.tobiipro.com/product-listing/tobii-pro-glasses-2/.

[3] 2020. URL: https://apps.automeris.io/wpd/.

[4] 2020. URL: http://seleniumide.org/.

[5] URL: https://www.cae.com/civil-aviation/locations/cae-brussels/.

[6] URL: https://www.tobiipro.com/.

[7] Tole; J. R.; Stephens; A. T.; Vivaudou; M.; Harris; R. L.; Ephrath; A. 'Entropy, instrument scan, and pilot workload'. English. In: (1982). URL: https://www.researchgate.net/publication/23888979_Entropy_instrument_scan_and_pilot_workload/link/00b7d52b0931fbcdc5000000/download.

[8] Tobii Pro AB. *Tobii Pro Lab*. Computer software. Version 1.145. Danderyd, Stockholm, 2014. URL: http://www.tobiipro.com/.

[9] Tobii Pro AB. *Tobii Pro Lab User Manual*. Version 1.145. Danderyd, Stockholm, 2014. URL: http://www.tobiipro.com/.

[10] Halevy; Alon, Norvig; Peter, and Pereira; Fernando. 'The Unreasonable Effectiveness of Data'. In: *IEEE Intelligent Systems* (2009), pp. 8–12. URL: https://doi.org/10.1109/MIS.2009.36.

[11] Richard Bellman. *Dynamic programming*. Princeton University Press, 1984.

[12] Avner Bendheim. 'The Effects of Automation on Human Performance in High-Risk Environments: A Design Research Case Study on Cockpit Automation in Commercial Aircrafts in Israel'. In: *Frontiers in Human Neuroscience* 12 (2018). DOI: 10.3389/conf.fnhum.2018.227.00071.

[13] I.I. Bittencourt et al. *Artificial Intelligence in Education: 21st International Conference, AIED 2020, Ifrane, Morocco, July 6–10, 2020, Proceedings, Part I*. Lecture Notes in Computer Science. Springer International Publishing, 2020. ISBN: 9783030522377. URL: https://books.google.com.br/books?id=n1nvDwAAQBAJ.

[14] Michael Boss et al. ‘Systemic Risk Monitor: A Model for Systemic Risk Analysis and Stress Testing of Banking Systems’. In: *Financial Stability Report* 11 (2006), pp. 83–95. URL: https://ideas.repec.org/a/onb/oenbfs/y2006i11b2.html.

[15] Stephanie Brams et al. ‘Eye-tracking in aviation: a new method for detecting learned visual scan patterns of cockpit instrument in simulated flight’. en. In: *Eye-Tracking in Aviation. Proceedings of the 1st International Workshop (ETAVI 2020).* Ed. by Vsevolod Peysakhovich et al. 1st International Workshop on Eye-Tracking in Aviation (ETAVI 2020); Conference cancelled due to Corona virus (COVID-19). Toulouse; Zurich: ISAE-SUPAERO, Université de Toulouse; Institute of Cartography and Geoinformation (IKG), ETH Zurich, 2020-03, pp. 69–77. DOI: 10.3929/ethz-b-000407654.

[16] Johnson; N.; Wiegmann; D.; Wickens; C. ‘Effects of advanced cockpit displays on general aviation pilots’ decisions to continue visual flight rules flight into instrument meteorological conditions’. English. In: (2006).

[17] Alexander; A. L.; Wickens; C. D. ‘Integrated hazard displays: individual differences in visual scanning and pilot performance’. English. In: (2006).

[18] Kramer; A.; Tham; M.; Konrad; C.; Wickens; C.; Lintern; G.; Marsh; R.; Fox; J.; Merwin; D. ‘Instrument scan and pilot expertise’. English. In: (1994).

[19] Oseguera-Lohr; R. M.; Nadler; E. D. ‘Effects of an Approach Spacing Flight Deck Tool on Pilot Eyescan’. English. In: (2004).

[20] *Eyetracker warns against momentary driver drowsiness - Press Release Oktober 12, 2010.* en. URL: https://www.fraunhofer.de/en/press/research-news/2010/10/eye-tracker-driver-drowsiness.html (visited on 11/27/2020).

[21] FAA Federal Aviation Administration. ‘ADMINISTRATION, F. Pilot’s Handbook of Aeronautical Knowledge.’ English. In: (2016). URL: https://www.faa.gov/regulations_policies/handbooks_manuals/aviation/phak/media/pilot_handbook.pdf.

[22] FAA Federal Aviation Administration. ‘FAA Middle School Activities: Tachometer’. English. In: (2016). URL: https://www.faa.gov/education/educators/activities/middle/media/Tachometer.pdf.

[23] Liu; X.; Chen; T.; Xie; G.; Liu; G. ‘Contact-Free Cognitive Load Recognition Based on Eye Movement’. English. In: (2016).

[24] James E. Gentle, ed. *Handbook of computational statistics: concepts and methods.* eng. 2., rev. and upd. ed. Springer handbooks of computational statistics. Berlin: Springer, 2012. ISBN: 978-3-642-21551-3 978-3-642-21550-6.

[25] James E. Gentle, ed. *Handbook of computational statistics: concepts and methods.* eng. 2., rev. and upd. ed. Springer handbooks of computational statistics. Berlin: Springer, 2012. ISBN: 978-3-642-21551-3 978-3-642-21550-6.

[26] Mackenzie G. Glaholt. 'Eye Tracking in the Cockpit: a Review of the Relationships between Eye Movements and the Aviators Cognitive State'. English. In: (2014). URL: `https://apps.dtic.mil/docs/citations/AD1000097`.

[27] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[28] Li; J.; Lib; H.; Umer; W.; Wang; H.; Xing; X.; Zhao; S.; Houg; J. 'Identification and classification of construction equipment operators' mental fatigue using wearable eye-tracking technology'. English. In: (2020).

[29] Nauroisa; C. J.; Bourdin; C.; Stratulat; A.; Diaz; E.; Verchera; J. 'Detection and prediction of driver drowsiness using artificial neural network models'. English. In: (2019).

[30] Dennis H. Jones. 'An Error-Dependent Model of Instrument-Scanning Behavior in Commercial Airline Pilots'. English. In: (1985). URL: `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19850024452.pdf`.

[31] King; A. J.; Bol; N.; Cummins; R. G.; John; K. K. 'Improving Visual Behavior Research in Communication Science: An Overview; Review;and Reporting Recommendations for Using Eye-Tracking Methods'. English. In: (2019).

[32] Zennifa; F.; Ageno; S.; Hatano; S.; Iramina; K. 'Hybrid System for Engagement Recognition during Cognitive Tasks Using a CFS + KNN Algorithm'. English. In: (2018).

[33] Hayashi; M.; Ravinder; U.; McCann; R. S.; Beutter; B.; Spirkovska; L. 'Evaluating Fault Management Operations Concepts for Next-Generation Spacecraft: What Eye Movements Tell Us'. English. In: (2009).

[34] Madleňák; R.; Mašek; J.; Madleňáková; L. 'An experimental analysis of the driver's attention during train driving'. English. In: (2019).

[35] Hayashi; M. 'Hidden Markov models for analysis of pilot instrument scanning and attention switching'. English. In: (2004).

[36] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT Press, 2012.

[37] Caussea; M.; Lancelota; F.; Maillanta; J.; Behrendc; J.; Cousyd; M.; Schneiderb; N. 'Encoding decisions and expertise in the operator's eyes: Using eye-tracking as input for system adaptation'. English. In: (2018).

[38] Jack Nicas and Zach Wichter. 'A Worry for Some Pilots: Their Hands-On Flying Skills Are Lacking'. en-US. In: *The New York Times* (Mar. 2019). ISSN: 0362-4331. URL: https://www.nytimes.com/2019/03/14/business/automated-planes.html (visited on 03/29/2020).

[39] Endsley; M. R.; Kiris; E. O.; 'The Out-of-the-Loop Performance Problem and Level of Control in Automation'. English. In: (1995). URL: https://journals.sagepub.com/doi/10.1518/001872095779064555.

[40] Dick; A. O. 'Instrument scanning and controlling: using eye movement data to understand pilot behavior and strategies'. English. In: (1980).

[41] Flemisch; O.F. and Onken; R. 'Detecting usability problems with eye tracking in airborne battle management suppor'. English. In: (2000).

[42] Clinton V. Oster, John S. Strong, and Kurt Zorn. 'Why Airplanes Crash: Causes of Accidents Worldwide'. In: 1430-2016-118679 (2010), p. 20. DOI: 10.22004/ag.econ.207282. URL: http://ageconsearch.umn.edu/record/207282.

[43] *Overfitting — Definition of Overfitting by Oxford Dictionary on Lexico.com also meaning of Overfitting*. en. URL: https://www.lexico.com/definition/overfitting (visited on 10/01/2020).

[44] Schaudt; W. A.; Caufield; K. J.; Dyre; B. P. 'Effects of a virtual air speed error indicator on guidance accuracy and eye movement control during simulated flight'. English. In: (2002).

[45] Fabian Pedregosa et al. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[46] James; G.; Witten; D.; Hastie; T.; Tibshirani; R. *An Introduction to Statistical Learning*. 8., rev. and upd. ed. Springer handbooks of computational statistics. London: Springer, 2013. ISBN: 978-1-4614-7138-7 978-1-4614-7137-0.

[47] Tole; J. R.; Stephens; A. T.; Vivaudou; M.; Ephrath; A.; Young; L. R. 'Visual scanning behavior and pilot workload'. English. In: (1983).

[48] Keith Rayner. ‘Eye Movements in Reading and Information Processing: 20 Years of Research’. English. In: (1998).

[49] Lundberg; S. M.; Lee; S.; ‘A Unified Approach to Interpreting Model Predictions’. English. In: (2016). URL: `https://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf`.

[50] Wickens; C. D.; Alexander; A. L.; Thomas; L. C.; Horrey; W. J.; Nunes; A.; Hardy; T. J.; Zheng; X. S. ‘Synthetic vision system display: The effects of clutter on performance and visual attention allocation’. English. In: (2004).

[51] Garmin subsidiaries. *G1000® — Avionics — Garmin.* 2020. URL: `https://buy.garmin.com/en-US/US/p/6420`.

[52] Schnell; T.; Kwon; Y.; Merchants; S.; Etherington; T. ‘Improved Flight Technical Performance in Flight Decks Equipped With Synthetic Vision Information System Display’. English. In: (2004).

[53] The pandas development team. *pandas-dev/pandas: Pandas.* Version latest. Feb. 2020. DOI: `10.5281/zenodo.3509134`. URL: `https://doi.org/10.5281/zenodo.3509134`.

[54] ‘To be assigned’. In: ().

[55] Paul Tune. *The Unreasonable Ineffectiveness of Deep Learning on Tabular Data.* en. May 2020. URL: `https://towardsdatascience.com/the-unreasonable-ineffectiveness-of-deep-learning-on-tabular-data-fd784ea29c33` (visited on 10/01/2020).

[56] Unsplash. *Photo by Caleb Woods on Unsplash.* en. Library Catalog: unsplash.com. URL: `https://unsplash.com/photos/R21CJwGyqPQ` (visited on 03/30/2020).

[57] Carbonell; J. R.; Ward; J. L.; Senders; J. W. ‘A queueing model of visual sampling: Experimental validation’. English. In: *None* (1968).

[58] Senders; J. W. ‘A re-analysis of pilot eye-movement data’. English. In: (1966).

[59] Senders; J. W. ‘The human operator as a monitor and controller of multi-degree of freedom systems’. English. In: (1964).

[60] Williams; K. W. ‘Impact of aviation highway-in-the-sky displays on pilot situation awareness’. English. In: (2002).

[61] Feng; C.; Wanyan; X.; Yang; K.; Zhuang; D.; Wu; X. ‘A comprehensive prediction and evaluation method of pilot workload’. English. In: (2018).

[62] McClung; S. N. Kang; Z. 'Characterization of Visual Scanning Patterns in Air Traffic Control'. English. In: (2016).

[63] Gal Ziv. 'Gaze Behavior and Visual Attention: A Review of Eye Tracking Studies in Aviation'. English. In: (2016). URL: https://www.tandfonline.com/doi/full/10.1080/10508414.2017.1313096.

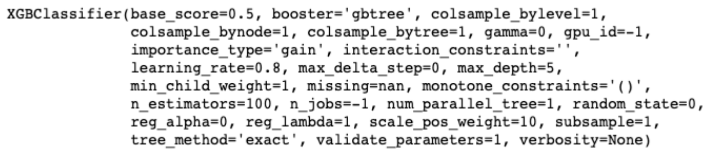
PART V

APENDIX

APPENDIX A – MODEL HYPERPARAMETERS

Hyper-parameters for the TR20 model:

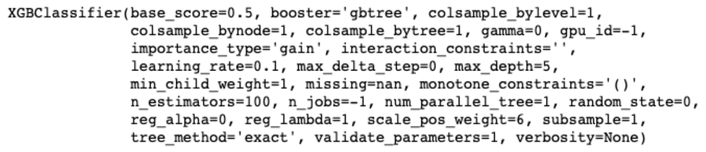
Figure 62: TR20 XGBoost Parameters. (Source: Author).



```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.8, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=-1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=10, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

Hyper-parameters for the SL10 model:

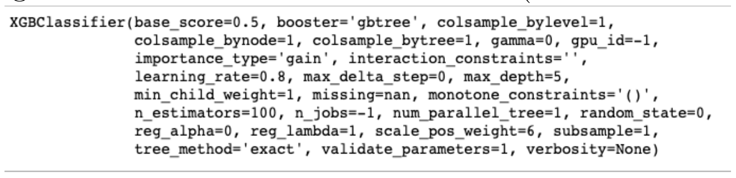
Figure 63: SL10 XGBoost Parameters. (Source: Author).



```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=-1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=6, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

Hyper-parameters for the VT0 model:

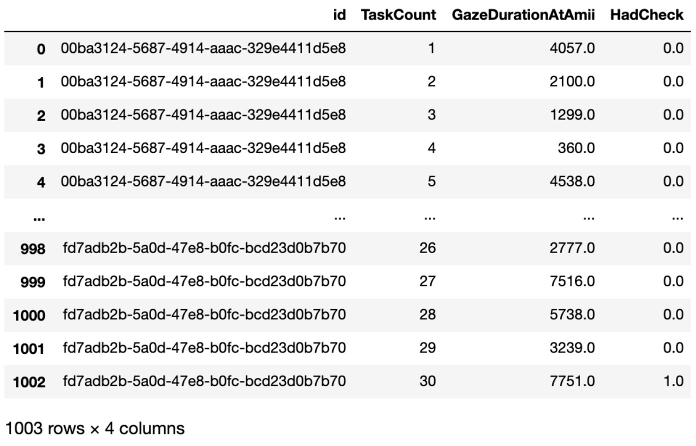
Figure 64: VT0 XGBoost Parameters. (Source: Author).



```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.8, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=-1, num_parallel_tree=1, random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=6, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

APPENDIX B – PRELIMINARY MODEL

As mentioned in the section 4.2.3, a simplified model was created to demonstrate a signal between gaze related data and flight performance and offer a baseline of model precision. To this effect, the methods of the section 4.2.1 were followed to yield a Master Table:



	id	TaskCount	GazeDurationAtAmii	HadCheck
0	00ba3124-5687-4914-aaac-329e4411d5e8	1	4057.0	0.0
1	00ba3124-5687-4914-aaac-329e4411d5e8	2	2100.0	0.0
2	00ba3124-5687-4914-aaac-329e4411d5e8	3	1299.0	0.0
3	00ba3124-5687-4914-aaac-329e4411d5e8	4	360.0	0.0
4	00ba3124-5687-4914-aaac-329e4411d5e8	5	4538.0	0.0
...	...	...	...	...
998	fd7adb2b-5a0d-47e8-b0fc-bcd23d0b7b70	26	2777.0	0.0
999	fd7adb2b-5a0d-47e8-b0fc-bcd23d0b7b70	27	7516.0	0.0
1000	fd7adb2b-5a0d-47e8-b0fc-bcd23d0b7b70	28	5738.0	0.0
1001	fd7adb2b-5a0d-47e8-b0fc-bcd23d0b7b70	29	3239.0	0.0
1002	fd7adb2b-5a0d-47e8-b0fc-bcd23d0b7b70	30	7751.0	1.0

1003 rows × 4 columns

Figure 65: Preliminary Model Master Table Snapshot. (Source: Author).

B.1 Data Balancing

The first step, given this Master Table, is to examine the dataset balance or the proportion of classes in the target variable. Here, this is equivalent to examining the proportion of tasks with and without performance error in the Target of the Master Table. This lead to a value of:

$$D_{imb} = 9.27\% \tag{B.1}$$

This means approximately 10% of the tasks presented performance errors (ones in the Target), while 90% did not (zeros in the Target).

This represents a dataset imbalance, as explained in the section 4.3.3 and the *under-sampling* technique was used to address the issue. This yielded a smaller Master Table, with 186 rows instead of 1003, an 81% reduction. The undersampling process can be visualized in Figure 66.

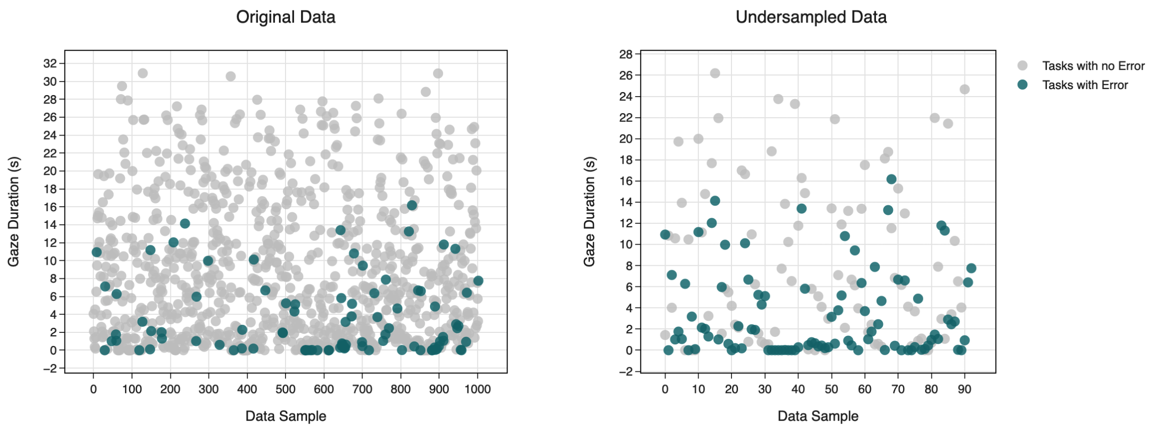


Figure 66: Unbalanced and Balanced Prelimnary Model Datasets. (Source: Author).

B.2 Algorithm Testing

Once the dataset was balanced, a 70% train test split was applied (as described in section 4.3.5), and different Machine Learning algorithms were tested with mostly standard parameters. The obtained accuracy metrics for the tested algorithms are as such:

Algorithm	**Prediction Accuracy**
Logistic Regression	0.696429
SVM	0.660714
Random Forest	0.589286
KNN	0.500000
Perceptron	0.446429

Table 10: Preliminary Model Algorithm Accuracy Comparison. (Source: Author).

A simple Logistic Regression model 4.3.3was able to yield a 69% accuracy in this very simplistic model. Although this may seem like a modest result, it is important to mention that this model was created in the stage; this is considered enough to demonstrate a directional validation in Machine Learning Methods for the problem at hand. An SVM (Support Vector Machine) also showed good performance and is further examined as well.

A deeper exploration of the models can show some important elements of how it behaves. Firstly, for the SVM, the confusion matrix - explained in the section 4.3.3 - in Figure 67 shows higher values in the main diagonal, as expected, and, as a positive addition, low values when it comes to the True Negative values. This is especially important in aviation - and human-centered tasks, in general - where the risk of an actual failure where no failure is detected is unacceptable.

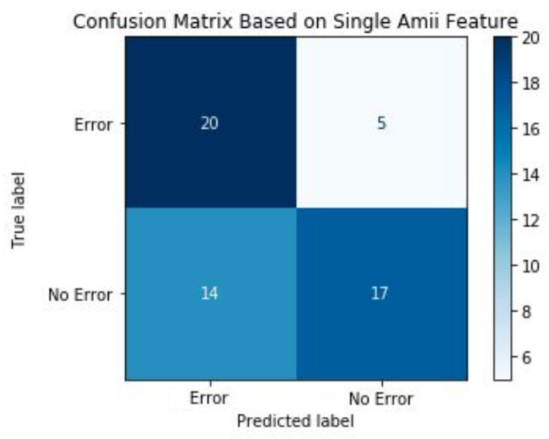


Figure 67: Preliminary Model Confusion Matrix. (Source: Author).

We can then observe the Receiver Operator Characteristics curve - details in the section 4.3.3 - which shows the model's ability to correctly predict failure cases, as it progressively incorrectly detects them. In this Logistic Regression model, the curve follows a trend above random, including an AUC - also explained at fsec:perfo$_{e}valf$.

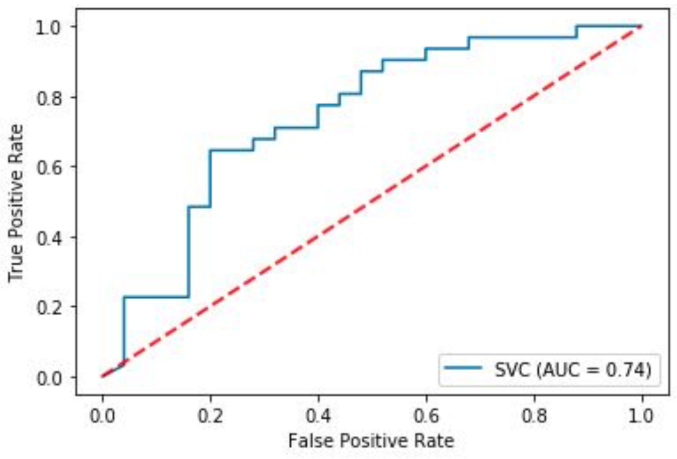


Figure 68: Preliminary Model Receiver Operator Characteristics Curve. (Source: Author).

B.3 Interpretation and Validation

Considering the Logistic Regression model, quantitatively, two main metrics can be used in evaluating the model performance, namely, accuracy and area under the ROC curve (AUC) - section 4.3.3 includes explanations for such metrics. The first had a value of 69%, indicating that this proportion of samples were correctly classified as having had an error. This value should be compared to a 50% accuracy in a random classification. The second metric is the AUC, which was calculated as 0.74 for the same model, compared to a 0.5 value in a no-skill classifier.

Based on what is pointed in the section 4.3.3 these values do not represent high performance but can be good indicators of a detectable signal between gaze behavior and performance error. It is important to remark that, at this point, the creation of a Preliminary Model had this as its primary objective, and in that sense, it could be considered to have fulfilled its requirements - section 4.2.3.

In a qualitative view, considering the Support Vector Machine model and its confusion matrix - explanation in the section 4.3.3 - in Figure 67, it is perceptible that the False Negative values are the lowest. These indicate flight tasks that were predicted to have no error but did have them. This certainly is a positive trait of the model, as it means there are very few occasions in which a potential threat goes undetected. This value must remain low in the models we develop, as there could be human life risks associated with errors that go undetected by the model. At this point, this low value, although a good directional indicator for this paper, was obtained without any tuning, and later, in the application of the GAPE methodology, a precision-recall imposition will be made to address this concern, as mentioned in section 4.3.3.

B.4 Possible Improvements

The model demonstrates interesting results but considering information obtained from the literature, some improvements can be made for the Preliminary Model's next iteration. Some of the most relevant ones would be:

- Apply model prediction for complete dataset, instead of the under-sampled one (may yield better results)
- Use precision and recall metrics instead of accuracy for performance validation (allows for direct comparison with baseline)
- Verify and mitigate possible leakages, especially considering the reaction time of the simulator instructions

APPENDIX C – PILOTS PARTS IDS

Table 11: List of Pilot Sessions parts

aug_ot66_p00	aug_ot66_p01	aug_ot66_p02	aug_ot66_p03	aug_ot66_p04	aug_rt45_p02
aug_rt45_p04	aug_rt45_p05	aug_rt45_p08	ben_ot80_p00	ben_ot80_p01	ben_ot80_p04
ben_ot80_p05	ben_ot80_p06	ben_ot80_p07	ben_rt74_p00	ben_rt74_p01	ben_rt74_p04
ben_rt74_p05	ben_rt74_p07	ben_rt74_p08	ben_rt74_p09	ben_rt74_p10	ben_rt74_p11
ben_rt74_p12	bre_ot61_p00	bre_ot61_p09	bre_ot61_p10	bre_ot61_p11	bre_ot61_p15
bre_ot61_p18	bre_ot61_p20	bre_ot61_p22	bre_rt51_p01	bre_rt51_p16	bre_rt51_p28
dav_ot81_p03	dav_ot81_p09	dav_rc87_p00	dav_rc87_p01	dav_rc87_p02	dav_rc87_p03
dav_rc87_p05	dav_rc87_p06	dav_rt72_p00	dav_rt72_p01	dav_rt72_p02	dav_rt72_p03
dav_rt72_p04	dav_rt72_p05	dav_rt72_p06	dav_rt72_p07	dav_rt72_p08	edo_ot58_p00
edo_ot58_p02	edo_ot58_p04	edo_ot58_p05	edo_ot58_p07	edo_ot58_p08	edo_ot58_p10
edo_ot58_p13	edo_ot58_p14	edo_rt46_p00	edo_rt46_p02	edo_rt46_p03	edo_rt46_p04
edo_rt46_p05	edo_rt46_p06	edo_rt46_p08	edo_rt46_p10	edo_rt46_p14	edo_rt46_p15
fab_ot62_p00	fab_ot62_p01	fab_ot62_p02	fab_ot62_p05	fab_rt50_p00	fab_rt50_p01
fab_rt50_p02	fab_rt50_p03	fab_rt50_p04	fab_rt50_p05	fab_rt50_p08	flo_ot77_p00
flo_ot77_p06	flo_ot77_p09	flo_ot77_p10	flo_ot77_p12	flo_ot77_p15	flo_ot77_p17
flo_ot77_p18	flo_ot78_p01	flo_rt69_p00	flo_rt69_p01	flo_rt69_p02	flo_rt69_p03
flo_rt69_p04	flo_rt69_p05	flo_rt69_p07	flo_rt69_p08	flo_rt71_p01	flo_rt71_p09
flo_rt71_p11	flo_rt71_p13	flo_rt71_p14	flo_rt71_p16	flo_rt71_p17	flo_rt71_p18
flo_rt71_p19	gau_ot64_p00	gau_ot64_p01	gau_ot64_p02	gau_ot64_p03	gau_ot64_p04
gau_ot64_p10	gau_ot64_p24	gau_rt56_p09	gau_rt56_p12	gau_rt56_p13	hel_oc90_p00
hel_oc90_p01	hel_oc90_p02	hel_oc90_p04	hel_oc90_p05	hel_oc90_p06	hel_rc86_p07
hel_rc86_p08	jar_ot82_p19	jar_ot82_p24	jar_ot82_p25	jar_ot82_p29	jar_rt73_p08
jar_rt73_p11	jar_rt73_p13	jar_rt73_p19	jar_rt73_p35	jea_ot84_p04	jea_ot84_p05
jea_ot84_p06	jea_ot84_p07	jea_ot84_p08	jea_ot84_p09	jea_ot84_p10	jea_ot84_p11
jea_ot84_p12	jea_ot84_p13	jea_rt75_p01	jea_rt75_p03	jea_rt75_p04	jea_rt75_p05
jea_rt75_p06	jea_rt75_p07	jea_rt75_p09	jea_rt75_p14	jen_ot63_p01	jen_ot63_p03
jen_ot63_p06	jen_ot63_p07	jen_ot63_p08	jen_rt54_p02	jen_rt54_p03	jen_rt54_p04
jen_rt54_p05	jen_rt54_p07	jer_ot85_p01	jer_ot85_p02	jer_ot85_p03	jer_ot85_p08
jer_ot85_p11	jer_ot85_p12	jer_ot85_p13	jer_ot85_p20	jer_rt76_p08	jer_rt76_p15
jer_rt76_p25	jer_rt76_p33	jul_ot59_p00	jul_ot59_p01	jul_ot59_p02	jul_ot59_p03
jul_ot59_p05	jul_ot59_p06	jul_ot59_p09	jul_rt55_p08	mic_ot79_p02	mic_ot79_p08
mic_ot79_p12	mic_rt70_p00	mic_rt70_p01	mic_rt70_p02	mic_rt70_p03	mic_rt70_p05
mic_rt70_p07	mic_rt70_p08	rub_rc89_p02	rub_rc89_p05	rub_rc89_p06	rub_rc89_p07
rub_rc89_p09	rub_rc89_p10	rub_rc89_p12	rub_rc89_p13	rub_rc89_p14	tib_rc88_p00
tib_rc88_p02	tib_rc88_p03	tib_rc88_p04	tib_rc88_p06	vin_ot65_p03	vin_rt52_p03
vin_rt52_p15	wil_ot60_p09	wil_ot60_p16	wil_rt53_p04	wil_rt53_p05	wil_rt53_p09
wil_rt53_p10	wil_rt53_p13				

APPENDIX D – ALL GAPE FEATURES

Table 12: Master Table Complete Feature List

id	DwProportionalTimeWin_0.5_AOI_1	VelocityModule_task_expanding_mean
Timestamp	DwProportionalTimeWin_1.0_AOI_1	VelocityModule_task_expanding_std
GazeX	DwProportionalTimeWin_5.0_AOI_1	VelocityModule_task_expanding_quantile_90
GazeY	DwProportionalTime_AOI_2	Task
GazeX_roll2s_lag0s_mean	DwProportionalTimeTask_AOI_2	TaskCount
GazeX_roll2s_lag0s_median	DwProportionalTimeWin_0.5_AOI_2	GmDuration
GazeX_roll2s_lag0s_std	DwProportionalTimeWin_1.0_AOI_2	GmDurationStd
GazeX_roll2s_lag5s_mean	DwProportionalTimeWin_5.0_AOI_2	GmTravelDistance
GazeX_roll2s_lag5s_median	DwProportionalTime_AOI_3	GmTravelDistanceStd
GazeX_roll2s_lag5s_std	DwProportionalTimeTask_AOI_3	GazeChangeFlag
GazeX_roll5s_lag0s_mean	DwProportionalTimeWin_0.5_AOI_3	FixFreq

Table 12 continued from previous page

GazeX_roll5s_lag0s_median	DwProportionalTimeWin_1.0_AOI_3	FixFreqTask
GazeX_roll5s_lag0s_std	DwProportionalTimeWin_5.0_AOI_3	FixFreqWin1s
GazeX_roll5s_lag5s_mean	DwProportionalTime_AOI_4	FixFreqWin5s
GazeX_roll5s_lag5s_median	DwProportionalTimeTask_AOI_4	FixFreqWin10s
GazeX_roll5s_lag5s_std	DwProportionalTimeWin_0.5_AOI_4	SacFreq
GazeX_roll10s_lag0s_mean	DwProportionalTimeWin_1.0_AOI_4	SacFreqTask
GazeX_roll10s_lag0s_median	DwProportionalTimeWin_5.0_AOI_4	SacFreqWin1s
GazeX_roll10s_lag0s_std	DwProportionalTime_AOI_5	SacFreqWin5s
GazeX_roll10s_lag5s_mean	DwProportionalTimeTask_AOI_5	SacFreqWin10s
GazeX_roll10s_lag5s_median	DwProportionalTimeWin_0.5_AOI_5	GmDuration_roll5s_lag0s_mean
GazeX_roll10s_lag5s_std	DwProportionalTimeWin_1.0_AOI_5	GmDuration_roll5s_lag0s_quantile_10
GazeY_roll2s_lag0s_mean	DwProportionalTimeWin_5.0_AOI_5	GmDuration_roll5s_lag0s_quantile_90
GazeY_roll2s_lag0s_median	DwProportionalTime_AOI_6	GmTravelDistance_roll5s_lag0s_mean
GazeY_roll2s_lag0s_std	DwProportionalTimeTask_AOI_6	GmTravelDistance_roll5s_lag0s_quantile_10
GazeY_roll2s_lag5s_mean	DwProportionalTimeWin_0.5_AOI_6	GmTravelDistance_roll5s_lag0s_quantile_90
GazeY_roll2s_lag5s_median	DwProportionalTimeWin_1.0_AOI_6	GmTravelDistance_task_expanding_mean
GazeY_roll2s_lag5s_std	DwProportionalTimeWin_5.0_AOI_6	GmTravelDistance_task_expanding_quantile_10
GazeY_roll5s_lag0s_mean	DwProportionalTime_AOI_7	GmTravelDistance_task_expanding_quantile_90
GazeY_roll5s_lag0s_median	DwProportionalTimeTask_AOI_7	FlipX

Table 12 continued from previous page

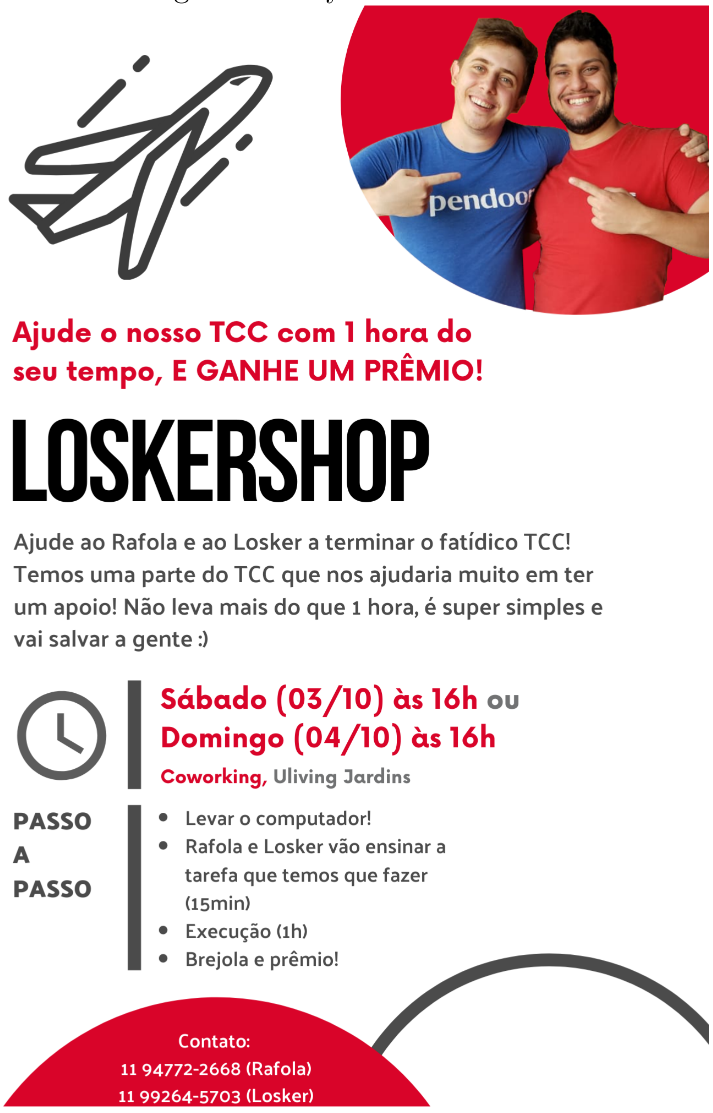
GazeY_roll5s_lag0s_std	DwProportionalTimeWin_0.5_AOI_7	FlipY
GazeY_roll5s_lag5s_mean	DwProportionalTimeWin_1.0_AOI_7	FlipXY
GazeY_roll5s_lag5s_median	DwProportionalTimeWin_5.0_AOI_7	FlipXYFreq
GazeY_roll5s_lag5s_std	AOI	FlipY_roll5s_lag0s_mean
GazeY_roll10s_lag0s_mean	SpeedX	FlipY_roll5s_lag0s_std
GazeY_roll10s_lag0s_median	SpeedY	FlipXY_roll5s_lag0s_mean
GazeY_roll10s_lag0s_std	AccelX	FlipXY_roll5s_lag0s_std
GazeY_roll10s_lag5s_mean	AccelY	FlipX_roll5s_lag0s_mean
GazeY_roll10s_lag5s_median	VelocityAngleTheta	FlipX_roll5s_lag0s_std
GazeY_roll10s_lag5s_std	VelocityModule	FlipXYFreq_roll5s_lag0s_mean
GazeY_task_expanding_mean	VelocityRotationAlpha	FlipXYFreq_roll5s_lag0s_std
GazeY_task_expanding_median	SpeedX_roll2s_lag0s_mean	FlipXYFreq_task_expanding_mean
GazeY_task_expanding_std	SpeedX_roll2s_lag0s_std	FlipXYFreq_task_expanding_std
DwCount	SpeedX_roll2s_lag0s_quantile_90	GazeMoveType
DwFreq	SpeedY_roll2s_lag0s_mean	GazeMoveCount
DwCountTask	SpeedY_roll2s_lag0s_std	Interpolated
DwFreqTask	SpeedY_roll2s_lag0s_quantile_90	ResampleInterpolated
DwCountWin0.5s	AccelX_roll2s_lag0s_mean	AutomapScore
DwFreqWin0.5s	AccelX_roll2s_lag0s_std	Target

Table 12 continued from previous page

DwCountWin1.0s	AccelX_roll2s_lag0s_quantile_90	TargetBackProp_5s
DwFreqWin1.0s	AccelY_roll2s_lag0s_mean	TargetBackProp_10s
DwCountWin5.0s	AccelY_roll2s_lag0s_std	TargetBackProp_20s
DwFreqWin5.0s	AccelY_roll2s_lag0s_quantile_90	TargetBackProp_60s
DwDuration	VelocityRotationAlpha_roll2s_lag0s_mean	TargetBackProp_120s
DwDurationStd	VelocityRotationAlpha_roll2s_lag0s_std	TargetBackProp_180s
DwProportionalTime_AOI_0	VelocityRotationAlpha_roll2s_lag0s_quantile_90	TargetBackProp_240s
DwProportionalTimeTask_AOI_0	VelocityAngleTheta_roll2s_lag0s_mean	TargetBackProp_300s
DwProportionalTimeWin_0.5_AOI_0	VelocityAngleTheta_roll2s_lag0s_std	TargetBackProp_360s
DwProportionalTimeWin_1.0_AOI_0	VelocityAngleTheta_roll2s_lag0s_quantile_90	TargetBackProp_420s
DwProportionalTimeWin_5.0_AOI_0	VelocityModule_roll2s_lag0s_mean	TargetBackProp_480s
DwProportionalTime_AOI_1	VelocityModule_roll2s_lag0s_std	TargetBackProp_540s
DwProportionalTimeTask_AOI_1	VelocityModule_roll2s_lag0s_quantile_90	TargetBackProp_600s

APPENDIX E – *LOSKERSHOP* EVENT

Figure 69: Flyer for Event



Ajude o nosso TCC com 1 hora do seu tempo, E GANHE UM PRÊMIO!

LOSKERSHOP

Ajude ao Rafola e ao Losker a terminar o fatídico TCC! Temos uma parte do TCC que nos ajudaria muito em ter um apoio! Não leva mais do que 1 hora, é super simples e vai salvar a gente :)

Sábado (03/10) às 16h ou **Domingo (04/10) às 16h**

Coworking, Uliving Jardins

PASSO A PASSO

- Levar o computador!
- Rafola e Losker vão ensinar a tarefa que temos que fazer (15min)
- Execução (1h)
- Brejola e prêmio!

Contato:
11 94772-2668 (Rafola)
11 99264-5703 (Losker)

APPENDIX F – CODE

F.1 Programming Language, code structure and libraries

This paper proposes using Machine Learning as a relatively novel approach to Gaze Behavior analysis. To implement such a solution and the proposed framework, different tools, and programming languages were considered. As a modern language with extensive and growing support for Machine Learning, Python was chosen as the primary programming language.

Some major libraries focused on Machine Learning for Python are:

- Pandas: provides data manipulation and analysis, offering data structures and operations for manipulating numerical tables and time series
- Sklearn: machine learning library featuring various classification, regression and clustering algorithms
- Plotly Seaborn, Matplotlib and Yellowbrick: python libraries focused on data visualization with extensive support for different plot types

F.2 Code structure and Files

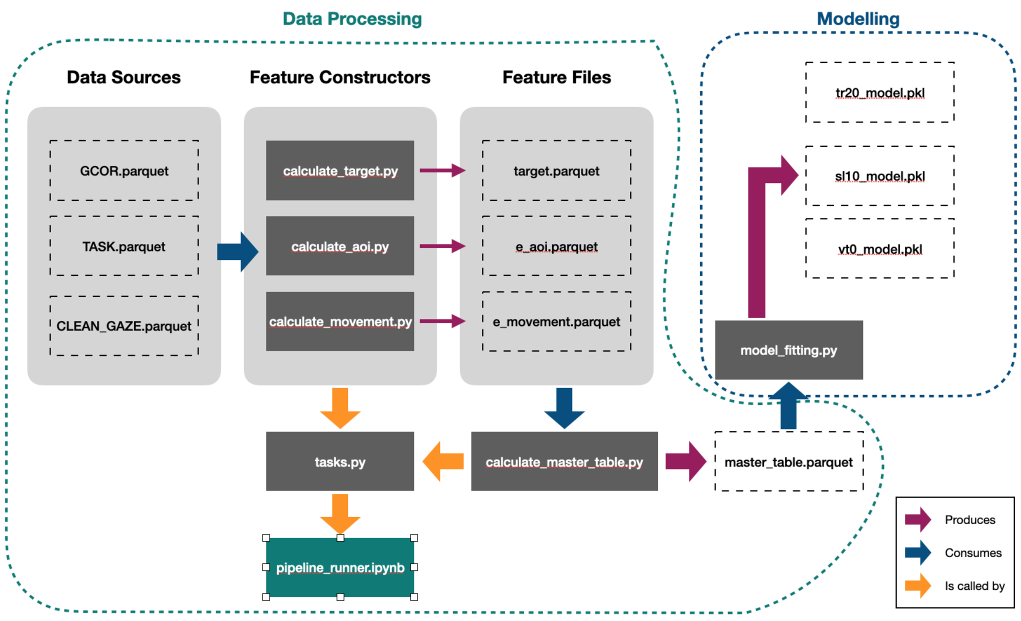


Figure 70: Code Structure Diagram

F.2.1 Pipeline

Listing F.1: Main Pipeline Running Jupyter Notebook

```
import sys
sys.path.append("../../src")

import pandas as pd
import tasks
pd.options.mode.chained_assignment = None

%load_ext autoreload
%autoreload 2

pd.set_option('display.max_rows', 500)

# DATABASES
tasks.make_clean_gaze()
tasks.make_smooth_gaze()
tasks.make_rdt()
```

```
# BASE FEATURES
tasks.make_aoi()
tasks.make_target()
# model features
tasks.make_movement()
tasks.make_dwell()
tasks.make_fixades()
tasks.make_master_table()
```

Listing F.2: Model Fit and Evaluation Jupyter Notebook

```



import numpy as np
from loguru import logger
import matplotlib.pyplot as plt
import seaborn as sns
from xgboost import XGBClassifier

import pandas as pd
# import auxiliary as aux
pd.set_option("display.max_rows", 500)

from xgboost import XGBClassifier
from sklearn.metrics import precision_recall_curve, f1_score, auc
import matplotlib.pyplot as plt
from datetime import datetime
import pickle

import xgboost as xgb
xgb.__version__

## Aux functions

configs = {
    "sample_period": 20,
    "seconds_per_row": 0.02,
    # master table parameters
    "task_begin_threshold": 10000,
    # target
    "back_prop_seconds_list": [5, 10, 20, 60, 120, 180, 240, 300,
        ↪ 360, 420, 480, 540, 600],
}



def _make_reduced_master_table(df_master_table, preproc: bool,
    ↪ flight_task: str = "VERT"):

    if preproc:

        reduced_master_table = df_master_table[df_master_table[
            ↪ flight_task] == 1].reset_index(
            drop=True)
        # drop Task Flag columns
        reduced_master_table = reduced_master_table.drop(
            columns=["COMBO", "RO+LO", "ROLL-OUT", "SLF", "TURN", "
                ↪ VERT"])

    else:
        reduced_master_table = df_master_table[df_master_table["Task
            ↪ "] == flight_task].reset_index(
            drop=True)

    return reduced_master_table



def _split_time_series(
    df,
    time_gap_seconds: int,
    train_perc: float = 0.8,
    full_return: bool = False,
    verbose: bool = False,
    y=None,
    groups=None,
):

    timestamp = pd.Series(df["Timestamp"])
    time_gap_ms = time_gap_seconds * 1000

    # print(time_gap_ms)

    # gap_size_proportion = (time_gap_seconds / configs["
        ↪ seconds_per_row"]) / (
    # df.shape[0]
    # )

# print(train_perc)

    # adjusted_train_perc = train_perc - 15 * gap_size_proportion
    adjusted_train_perc = train_perc

    # print(gap_size_proportion, adjusted_train_perc, df.shape[0])

# print("adjusted_bagulhation", adjusted_train_perc)

    train_end_rows = int(df.shape[0] * adjusted_train_perc)
    # print(df.shape[0])
    # print(train_end_rows)
    train = df.iloc[:train_end_rows]

    # print(train_end_rows)
    # display(train.head())

    test_start_ms = train.Timestamp.max() + time_gap_ms

# print("test_start_ms", test_start_ms)

    test = df[df["Timestamp"] >= test_start_ms]

    # display(test.head())
    train_prop = round(100 * (train.shape[0] / df.shape[0]), 2)
    test_prop = round(100 * (test.shape[0] / df.shape[0]), 2)

    if verbose:
        logger.info(
            "Effective Train-Test Split-> TRAIN: "
            + str(train_prop)
            + "% TEST: "
            + str(test_prop)
            + "%"
        )

    if full_return:
        return train, test, train_prop, test_prop
    else:
        return train, test

# Import Data

# Import Master Table
## importing master table without preprocessing
# FOR COLAB
df_master_table = pd.read_parquet("/content/drive/MyDrive/TCC/Colab
    ↪  Coding/data/full_MT.parquet")
# df_master_table = pd.read_parquet("../../data/master_tables/
    ↪ full_MT.parquet")
# df_master_table = pd.read_parquet("../../data/master_tables/
    ↪ full_preprocessed_MT.parquet")

# find and format target
target_cols = [col for col in df_master_table.columns if "Target"
    ↪ in col]

display("Targets:", target_cols)
display(df_master_table.head())
display(df_master_table.shape)

# XGBoost

# change taxonomy
df_experiment = df_master_table

# define options
target_choice = 10
split_time_gap = (target_choice)
rmt_choice = "SLF"

# define target col
if target_choice==0:
  target_col = "Target"
else:
  target_col = "TargetBackProp_"+str(target_choice)+"s"

# Simple pre-processing ----------------

# sort values
df_experiment = df_experiment.sort_values("Timestamp").reset_index(
    ↪ drop=True)

## format cols
df_experiment[target_col] = df_experiment[target_col].astype(int)
df_experiment["GazeMoveType"] = df_experiment["GazeMoveType"].
    ↪ replace("Fixation",0).replace("Saccade",1).astype(int)

## define columns to drop
targets_to_drop = [col for col in target_cols if col != target_col]
assert len(targets_to_drop) == (len(target_cols)-1)
cols_to_drop = [*targets_to_drop, "Task", "TaskCount", "
    ↪ GazeMoveCount",
               "Interpolated", "ResampleInterpolated"]
```

```

# Scope Selection ---------------

# # a. filter ids without any target
# target_counts = df_experiment.groupby("id")[target_col].sum()
# id_choice = list(target_counts[target_counts!=0].index)
# df_experiment = df_experiment[df_experiment["id"].isin(id_choice)
    ↪ ]

# b. select by n_rows, n_cols
# n_rows = 700000
# df_experiment = df_experiment.iloc[0:n_rows]
# df_experiment = df_experiment.drop(columns=drop_extra)

# d. select by Pilot
# df_experiment = df_experiment[df_experiment["id"]=="dav_rc87_p00
    ↪ "]

# c. select by Task (RMT)
df_experiment = _make_reduced_master_table(df_experiment,
                                           flight_task=rmt_choice,
                                           preproc=False)



# e. select handful of columns
# df_experiment = df_experiment[["id","Timestamp",'Gazedf', 'GazeY
    ↪ ','AOI','GazeMoveType', target_col]]

logger.info("Experiment Scoped! Target: " + target_col + "; Task: "
    ↪  + rmt_choice)
display(df_experiment.head())
display(df_experiment.shape)

# Effectively Dropping rows and cols
## drop cols and nulls
df_experiment = df_experiment.dropna()
df_experiment = df_experiment.drop(columns=cols_to_drop)
df_experiment = df_experiment.reset_index(drop=True)
logger.info("Pre-processing Done!")

# ## Prepare Train and Test

# # split train and test dfs

train_df, test_df = _split_time_series(df_experiment,
    ↪ time_gap_seconds = split_time_gap, train_perc=0.8, verbose=
    ↪ True)

logger.info("Train Test Split done! Last Train timestamp: " + str(
    ↪ train_df["Timestamp"].max()) + " First Test timestamp: " +
    ↪ str(test_df["Timestamp"].min()))

# drop id and Timestamp
test_df = test_df.drop(columns = ["id","Timestamp"])
train_df = train_df.drop(columns = ["id","Timestamp"])

print("Target Proportions")
display(test_df[target_col].value_counts(normalize=True))

# train set
X_train = train_df.drop(columns = target_col)
y_train = train_df[target_col]

# test set
X_test = test_df.drop(columns = target_col)
y_test = test_df[target_col]

# learning_list = [0.8, 0.3, 0.1]
# max_depth_list = [2, 5, 9]
# scale_pos_weight_list = [6, 10, 12]

learning_list = [0.1]
```

```
max_depth_list = [5]
scale_pos_weight_list = [6]

model_df = pd.DataFrame(columns=["opt","model"])

models = []
opts = []

for learning_rate in learning_list:
  for max_depth in max_depth_list:
    for scale_pos_weight in scale_pos_weight_list:

      print("Start Run: "+str(datetime.now()))

      # instantiate and fit model
      model = XGBClassifier(learning_rate=learning_rate,
                            max_depth=max_depth,
                            scale_pos_weight=scale_pos_weight,
                            objective="binary:logistic",
                            n_jobs=-1)



      # fit data
      model.fit(X_train,
                y_train,
                verbose=True,
      # early_stopping=10,
                eval_metric="auc")

      # opts.append(opt)
      models.append(model)
      # prt.append([precision, recall, thresholds])

model_df["opt"] = opts
model_df["model"] = models
model_df["prt"] = models
```

```
from sklearn.metrics import auc
for i,row in model_df.iterrows():
    model = row['model']
    print(row)
    # evaluate model
    y_probas = model.predict_proba(X_test)
    y_probas_class_1 = y_probas[:,1]

    # Precision Recall
    precision, recall, thresholds = precision_recall_curve(y_test,
        ↪ y_probas_class_1)
    prt = pd.DataFrame({"precision":precision[:-1], "recall":recall
        ↪ [:-1], "threshold":thresholds})
    s1 = round(auc(x=thresholds, y =precision[:-1]),2)
    s2 = round(auc(x=thresholds, y =recall[:-1]),2)
    s3 = s1+s2
    plt.figure(figsize=(8,6))
    ax = sns.lineplot(data=prt.set_index("threshold"))
    ax.set_title('Score=' + str(s3))



# learning_list = [0.8, 0.3, 0.1]
# max_depth_list = [2, 5, 9]
# scale_pos_weight_list = [6, 10, 12]

final_model=row["model"]
# save model to file
pickle.dump(final_model, open("/content/drive/MyDrive/TCC/Colab
    ↪ Coding/data/SL10.pickle.dat", "wb"))
# train set
X_train.to_csv("/content/drive/MyDrive/TCC/Colab Coding/data/
    ↪ SL10_xtrain.csv", index=False)
y_train.to_csv("/content/drive/MyDrive/TCC/Colab Coding/data/
    ↪ SL10_ytrain.csv", index=False)
```

```
# test set
X_test.to_csv("/content/drive/MyDrive/TCC/Colab Coding/data/
    ↪ SL10_xtest.csv", index=False)
y_test.to_csv("/content/drive/MyDrive/TCC/Colab Coding/data/
    ↪ vt0_ytest.csv", index=False)

# # evaluate model
# y_probas = model.predict_proba(X_test)
# y_probas_class_1 = y_probas[:,1]

# # Precision Recall
# precision, recall, thresholds = precision_recall_curve(y_test,
    ↪ y_probas_class_1)
# prt = pd.DataFrame({"precision":precision[:-1], "recall":recall
    ↪ [:-1], "threshold":thresholds})
```

Listing F.3: Pipeline Tasks Orchestrator

```
import sys
import os

sys.path.append("../../src")

from calc_databases.clean_gaze import calculate_clean_gaze
from calc_databases.smooth_gaze import calculate_smooth_gaze
from calc_databases.rdt import calculate_rdt
from calc_features.target import calculate_target
from calc_features.movement import calculate_movement
from calc_features.aoi import calculate_aoi
from calc_features.dwell import calculate_dwell
from calc_features.fixades import calculate_fixades
from master_table import calculate_master_table

import pandas as pd
import auxiliary as aux
from loguru import logger

configs = aux.get_configs()
```

```
def make_clean_gaze():

    # Import Data
    # read name correspondence table
    id_info = pd.read_csv("../../data/infos/id_infos.csv")

    # read raw data
    r_tobii_gaze = pd.read_csv(
        "../../../z_large_files/Other/full_raw_without_AOI_hit.csv"
    )

    # read tasks per pilot table
    tasks_table = pd.read_parquet("../../data/databases/TASK_raw.
        ↪ parquet")

    # read matching information
    result_matches = pd.read_csv("../../data/infos/final_matches.csv
        ↪ ")

    # Create Features and Tables
    e_clean_gaze, gcor_raw = calculate_clean_gaze(
        id_info=id_info,
        r_tobii_gaze=r_tobii_gaze,
        tasks_table=tasks_table,
        result_matches=result_matches,
    )

    # Save Features and Tables
    e_clean_gaze.to_parquet("../../data/databases/CLEAN_GAZE.parquet
        ↪ ")
    gcor_raw.to_parquet("../../data/databases/GCOR_RAW.parquet")


def make_smooth_gaze():
    # Import Data

    # import raw gaze coordinates
    gcor_raw = pd.read_parquet("../../data/databases/GCOR_RAW.
        ↪ parquet")

    # Create Features and Tables
    smooth_gaze = calculate_smooth_gaze(gcor=gcor_raw)

    # Save Features and Tables
    smooth_gaze.to_parquet("../../data/databases/SMOOTH_GAZE.parquet
        ↪ ")


def make_rdt():

    # Import Data

    # import clean_gaze
    clean_gaze = pd.read_parquet("../../data/databases/CLEAN_GAZE.
        ↪ parquet")

    # import smooth gaze
    smooth = pd.read_parquet("../../data/databases/SMOOTH_GAZE.
        ↪ parquet")

    # import tasks sequence per pilot
    tasks_table = pd.read_parquet("../../data/databases/TASK.parquet
        ↪ ")

    # Create Features and Tables
    (
        e_spine,
        e_gaze_metadata,
        e_rec_metadata,
        e_task,
        e_task_with_checks,
        gcor,
    ) = calculate_rdt(clean_gaze=clean_gaze, smooth=smooth,
        ↪ tasks_table=tasks_table)

    # Save Features and Tables
    e_spine.to_parquet("../../data/features/e_spine.parquet")
    e_gaze_metadata.to_parquet("../../data/features/e_gaze_metadata.
        ↪ parquet")
    e_rec_metadata.to_parquet("../../data/features/e_rec_metadata.
        ↪ parquet")
    e_task.to_parquet("../../data/features/e_task.parquet")
    e_task_with_checks.to_parquet("../../data/features/
        ↪ e_task_with_checks.parquet")
    gcor.to_parquet("../../data/databases/GCOR.parquet")


def make_target():

    # Import Data

    # import tasks
    e_task_with_checks = pd.read_parquet(
        "../../data/features/e_task_with_checks.parquet"
    )

    # get parameters
    back_prop_seconds_list = configs["back_prop_seconds_list"]
    seconds_per_row = configs["seconds_per_row"]

    # Create Features and Tables
    e_target = calculate_target(
        e_task_with_checks=e_task_with_checks,
        back_prop_seconds_list=back_prop_seconds_list,
        seconds_per_row=seconds_per_row,
    )

    # Save Features and Tables
    aux.check_save_feature(
        e_df=e_target, file_path="../../data/target/e_target.parquet
            ↪ "
    )


def make_movement():

    # Import Data
    # import raw gaze coordinates
    gcor = pd.read_parquet("../../data/databases/GCOR.parquet")

    # import spine
    e_spine = pd.read_parquet("../../data/features/e_spine.parquet")

    # import tasks
    e_task = pd.read_parquet("../../data/features/e_task.parquet")

    # Create Features and Tables
    e_gaze_positions, e_velocity, e_direction_change =
        ↪ calculate_movement(
        gcor=gcor, e_spine=e_spine, e_task=e_task
    )

    # Save Features and Tables
    aux.check_save_feature(
        e_df=e_velocity, file_path="../../data/features/e_velocity.
            ↪ parquet"
    )
    aux.check_save_feature(
        e_df=e_gaze_positions,
        file_path="../../data/features/e_gaze_positions_autoregs.
            ↪ parquet",
    )
    aux.check_save_feature(
        e_df=e_direction_change,
        file_path="../../data/features/e_direction_change.parquet",
    )


def make_aoi():

    # Import Data
    # import gaze
    gcor = pd.read_parquet("../../data/databases/GCOR.parquet")

    # Create Features and Tables
    e_aoi = calculate_aoi(gcor)

    # Save Features and Tables
    aux.check_save_feature(
        e_df=e_aoi, file_path="../../data/features/e_aoi.parquet",
    )


def make_dwell():

    # Import Data
    e_aoi = pd.read_parquet("../../data/features/e_aoi.parquet")
    e_task = pd.read_parquet("../../data/features/e_task.parquet")

    # get parameters
    seconds_per_row = configs["seconds_per_row"]

    # calculate
    e_dwell = calculate_dwell(e_aoi, e_task, seconds_per_row=
        ↪ seconds_per_row)

    # save
    aux.check_save_feature(
        e_df=e_dwell, file_path="../../data/features/e_dwell.parquet
            ↪ "
    )


def make_fixades():
    # import
    e_gaze_metadata = pd.read_parquet("../../data/features/
        ↪ e_gaze_metadata.parquet")
    e_task = pd.read_parquet("../../data/features/e_task.parquet")
    gcor = pd.read_parquet("../../data/databases/GCOR.parquet")

    # get parameters
    seconds_per_row = configs["seconds_per_row"]

    # calculate
    e_fixades = calculate_fixades(
        e_gaze_metadata, e_task, gcor, seconds_per_row=
            ↪ seconds_per_row
    )

    # save
    aux.check_save_feature(
        e_df=e_fixades, file_path="../../data/features/e_fixades.
            ↪ parquet"
    )


def make_master_table():

    # Import Data

    # get parameters
    sample_period = configs["sample_period"]
    task_begin_threshold = configs["task_begin_threshold"]
```

```
feature_blacklist = [
    "e_spine",
    "e_rec_metadata",
    "e_what_to_drop",
    "e_movement",
    "e_task_with_checks",
    # "e_fixades",
    # "e_dwell",
]

# extract list of created features
feature_list = os.listdir("../../data/features/")
feature_list = [
    feature.split(".")[0]
    for feature in feature_list
    if feature.endswith(".parquet")
]
feature_list = [
    feature for feature in feature_list if feature not in
       ↪ feature_blacklist
]

# import all features to dict
fdfs = {}
for feature in feature_list:
    fdfs[feature] = pd.read_parquet("../../data/features/" +
       ↪ feature + ".parquet")

# import spine
e_spine = pd.read_parquet("../../data/features/e_spine.parquet")

# import target
e_target = pd.read_parquet("../../data/target/e_target.parquet")

# Calculate
```

```
    output = calculate_master_table(
        e_spine=e_spine,
        fdfs=fdfs,
        e_target=e_target,
        feature_list=feature_list,
        sample_period=sample_period,
        mask_task_start=True,
        dropna=True,
        preprocess=True,
        make_rmts=False,
        task_begin_threshold=task_begin_threshold,
    )

    # Save
    logger.info("Saving RMTS")
    for mt in output:
        output[mt].to_parquet(
            "../../data/master_tables/" + mt + "_preprocessed_MT.
                ↪ parquet"
        )


######
# def make_movement():

# Import Data

# Create Features and Tables

# Save Features and Tables
```

Listing F.4: Master Table Constructor

```

## Libraries and Defines

import sys
```

```
sys.path.append("../../")

import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import seaborn as sns
from IPython.core.display import HTML, display
from loguru import logger
from tqdm.notebook import tqdm

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import minmax_scale
from sklearn.preprocessing import MaxAbsScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import QuantileTransformer
from sklearn.preprocessing import PowerTransformer


def calculate_master_table(
    e_spine,
    fdfs,
    e_target,
    feature_list,
    sample_period,
    mask_task_start: bool,
    dropna: bool,
    preprocess: bool,
    make_rmts: bool,
    task_begin_threshold,
    scale_method: "Any of StandardScaler, MinMaxScaler, MaxAbsScaler
        ↪ , RobustScaler, \
    PowerTransformer, PowerTransformer, QuantileTransformer,
        ↪ QuantileTransformer, Normalizer" = "MinMaxScaler",
):

    # Merge all Features

    # start from spine
    master_table = e_spine

    logger.info("Merging all features")
    # merge all features
    for feature in tqdm(feature_list):

        master_table = master_table.merge(
            fdfs[feature], on=["id", "Timestamp"], how="left"
        )

    # Add Target

    master_table = master_table.merge(e_target, on=["id", "Timestamp
        ↪ "], how="left")

    # Mask Task Beginnings
    if mask_task_start:

        task_time = (
            master_table.groupby(["id", "TaskCount"])["Timestamp"].
                ↪ cumcount() + 1
        )
        task_time = task_time * sample_period
        master_table["TaskTime"] = task_time
        del task_time

        master_table = master_table[master_table["TaskTime"] >
            ↪ task_begin_threshold]
```

```
# Drop all NaN
if dropna:

    # nan_counts = {}
    # for col in master_table:

    # col_series = master_table[col]
    # nan_count = col_series[col_series.isna()].shape[0]
    # nan_counts[col] = nan_count

    master_table = master_table.dropna()
    master_table = master_table.reset_index(drop=True)

# Pre-Process Variables
if preprocess:

    preproc_master_table = master_table.copy()

    # Categorical

    ## task
    task_dummies = pd.get_dummies(preproc_master_table["Task"])
    preproc_master_table = pd.concat([preproc_master_table,
       ↪ task_dummies], axis=1)

    ## aoi
    aoi_dummies = pd.get_dummies(preproc_master_table["AOI"],
       ↪ prefix="AOI")
    preproc_master_table = pd.concat([preproc_master_table,
       ↪ aoi_dummies], axis=1)

    # drop original cols
    preproc_master_table = preproc_master_table.drop(columns=["
       ↪ Task", "AOI"])

    ## gazeMoveType
```

```
        preproc_master_table["GazeMoveType"] = (
            preproc_master_table["GazeMoveType"]
            .replace("Fixation", 1)
            .replace("Saccade", 0)
            .astype(int)
        )

        # define feature scope
        cols_blacklist = ["id", "Timestamp", "AutomapScore", "
           ↪ TaskCount", "Task"]
        feature_cols = [
            col for col in preproc_master_table.columns if col not
               ↪ in cols_blacklist
        ]

        # Numerical vs. Binary Features
        binary_cols = [
            col for col in feature_cols if preproc_master_table[col
               ↪ ].nunique() <= 2
        ]
        numeric_cols = [
            col for col in feature_cols if preproc_master_table[col
               ↪ ].nunique() > 2
        ]

        # scale preprocess Features
        preproc_master_table = _scale_numeric_features(
            df=preproc_master_table, numeric_cols=numeric_cols,
               ↪ method=scale_method
        )

        master_table = preproc_master_table

    output = {}

    # Reduced Master Tables
    if make_rmts:

        # save relevant parameters
        task_list = list(task_dummies.columns)

        for task in task_list:
            output[task] = preproc_master_table[
                preproc_master_table[task] == 1
            ].reset_index(drop=True)

    else:
        output["full"] = master_table

    return output


## AUX FUNCS


def _scale_numeric_features(df, numeric_cols, method):

    scalers = {
        "StandardScaler": StandardScaler(),
        "MinMaxScaler": MinMaxScaler(),
        "MaxAbsScaler": MaxAbsScaler(),
        "RobustScaler": RobustScaler(quantile_range=(25, 75)),
        "PowerTransformer": PowerTransformer(method="yeo-johnson"),
        "PowerTransformer": PowerTransformer(method="box-cox"),
        "QuantileTransformer": QuantileTransformer(
            ↪ output_distribution="normal"),
        "QuantileTransformer": QuantileTransformer(
            ↪ output_distribution="uniform"),
        "Normalizer": Normalizer(),
    }

    scaler = scalers[method]
    for col in tqdm(numeric_cols):
        current_col = np.array(df[col]).reshape(-1, 1)
        new_col = scaler.fit_transform(current_col)
        df[col] = new_col

    return df
```

F.2.2 Features

Listing F.5: AOI Features Calculator

```
import pandas as pd
import numpy
from PIL import Image


def calculate_aoi(gcor):

    # import masks
    mask_path = "../../../z_large_files/Other/masks/"
    mask_files = [
        "BA_mask.png",
        "VS_mask.png",
        "PWR_mask.png",
        "SPEED_mask.png",
        "HDG_mask.png",
        "ALT_mask.png",
        "ADI_mask.png",
    ]
    mask_names = ["BA", "VS", "PWR", "SPEED", "HDG", "ALT", "ADI"]
    mask_dict = {}

    for mask_file, mask_name in zip(mask_files, mask_names):
        mask_dict[mask_name] = get_image_pixels(mask_path +
            ↪ mask_file)
```

```
# change taxonomy
e_dt = gcor

# Match Gaze

matched_gaze = []
for gaze_x, gaze_y in zip(e_dt["GazeX"], e_dt["GazeY"]):

    # i+=1
    gaze_x = int(gaze_x)
    gaze_y = int(gaze_y)

    if mask_dict["ADI"][gaze_y, gaze_x] > 0:
        matched_gaze.append(1)
    elif mask_dict["ALT"][gaze_y, gaze_x] > 0:
        matched_gaze.append(2)
    elif mask_dict["BA"][gaze_y, gaze_x] > 0:
        matched_gaze.append(3)
    elif mask_dict["HDG"][gaze_y, gaze_x] > 0:
        matched_gaze.append(4)
    elif mask_dict["PWR"][gaze_y, gaze_x] > 0:
        matched_gaze.append(5)
    elif mask_dict["SPEED"][gaze_y, gaze_x] > 0:
        matched_gaze.append(6)
    elif mask_dict["VS"][gaze_y, gaze_x] > 0:
        matched_gaze.append(7)
    else:
        matched_gaze.append(0)

e_dt["AOI"] = matched_gaze

# Assign output
e_aoi = e_dt[["id", "Timestamp", "AOI"]]

return e_aoi
```

```
## AUX FUNCS



def get_image_pixels(image_path, channel="Alpha"):
    "Get a numpy array of an image so that one can access values[x][
       ↪ y]."
    image = Image.open(image_path, "r")
    width, height = image.size
    pixel_values = list(image.getdata())
    if image.mode == "RGBA":
        channels = 4
    elif image.mode == "L":
        channels = 1
    else:
        print("Unknown mode: %s" % image.mode)
        return None
    pixel_values = numpy.array(pixel_values).reshape((height, width,
       ↪  channels))
    if channel == "Alpha":
        pixel_values = pixel_values[0:, 0:, 3]
    return pixel_values
```

Listing F.6: Dwell Features Calculator

```
import pandas as pd
import sys
import numpy as np

sys.path.insert(1, "../../src/")
import auxiliary as aux



def calculate_dwell(e_aoi, e_task, seconds_per_row):
    e_dwell = e_aoi
    e_dwell = e_dwell.merge(
        e_task, on=["id", "Timestamp"], how="left", left_index=True,
           ↪  right_index=True
    )

    # Constants and Definitions

    # CONSTANTS DEFINITION
    sampling_period = 1000 * seconds_per_row # miliseconds
    windows = pd.Series([0.5, 1, 5]) # this is in seconds
    windows_sizes = 1000 * windows / sampling_period # this is in
        ↪ number of rows

    # Count and Frequency

    # Create AOI Change Flag
    e_dwell["AOIChange"] = (
        (e_dwell.groupby("id")["AOI"].shift(-1) - e_dwell["AOI"]).
           ↪ fillna(0).astype(int)
    )
    e_dwell["AOIChange"] = (e_dwell["AOIChange"] != 0).astype(int)

    ## Flight

    # dwell count
    e_dwell["DwCount"] = e_dwell.groupby("id")["AOIChange"].cumsum()

    # duration until now
    e_dwell["Duration"] = e_dwell.groupby("id")["Timestamp"].
        ↪ cumcount() + 1
    e_dwell["Duration"] = e_dwell["Duration"] * sampling_period #
        ↪ miliseconds

    # frequency calculation
    e_dwell["DwFreq"] = e_dwell["DwCount"] / e_dwell["Duration"]
    e_dwell = e_dwell.drop(columns="Duration")

    ## Task
```

```
# dwell count
e_dwell["DwCountTask"] = e_dwell.groupby(["id", "TaskCount"])["
    ↪ AOIChange"].cumsum()

# duration until now in task to calculate frequency
e_dwell["DurationTask"] = (
    e_dwell.groupby(["id", "TaskCount"])["Timestamp"].cumcount()
        ↪  + 1
)
e_dwell["DurationTask"] = e_dwell["DurationTask"] *
    ↪ sampling_period # miliseconds

# frequency calculation
e_dwell["DwFreqTask"] = e_dwell["DwCountTask"] / e_dwell["
    ↪ DurationTask"]
e_dwell = e_dwell.drop(columns="DurationTask")

## Window

# CREATE DWELL COUNT AND FREQUECY IN WINDOW
for i in range(len(windows_sizes)):
    # define columns names and window for agregation
    count_name = "DwCountWin" + str(windows[i]) + "s"
    freq_name = "DwFreqWin" + str(windows[i]) + "s"

    # dwell count until now in window
    e_dwell[count_name] = e_dwell.groupby("id")["AOIChange"].
        ↪ apply(
        lambda x: x.rolling(int(windows_sizes[i])).sum()
    )

    # duration until now in window to calculate frequency
    e_dwell["DurationWin"] = e_dwell.groupby("id")["Timestamp"].
        ↪ apply(
        lambda x: x.rolling(int(windows_sizes[i])).count() + 1
    )
    e_dwell["DurationWin"] = e_dwell["DurationWin"] *
        ↪ sampling_period # miliseconds

    # frequency calculation
    e_dwell[freq_name] = e_dwell[count_name] / e_dwell["
        ↪ DurationWin"]
e_dwell = e_dwell.drop(columns="DurationWin")

# Duration

# CREATE DWELL DURATION
e_dwell["DwDuration"] = e_dwell.groupby(["id", "DwCount"]).
    ↪ cumcount()
e_dwell["DwDuration"] = e_dwell["DwDuration"] * sampling_period

# with leakage prevention and considering only last dwell value
# select only the duration points regarding the end of the dwell

e_dwell["MaxDuration"] = e_dwell.groupby(["id", "DwCount"])["
    ↪ DwDuration"].transform(
    "max"
)
e_dwell.loc[
    e_dwell["MaxDuration"] != e_dwell["DwDuration"], ["
        ↪ MaxDuration"]
] = np.nan
e_dwell["MaxDuration"] = e_dwell["MaxDuration"].shift(1)

# actually calculate Std
e_dwell["DwDurationStd"] = e_dwell.groupby("id")["MaxDuration"].
    ↪ apply(
    lambda x: x.expanding().std()
)
e_dwell["DwDurationStd"] = e_dwell["DwDurationStd"].fillna(0)
e_dwell = e_dwell.drop(columns="MaxDuration")
```

```
# Proportional Dwell Time (for AOI)

# CREATE PROPORTIONAL DWELL TIME
AOI_flags = pd.get_dummies(e_dwell["AOI"], prefix="AOI")
e_dwell = e_dwell.merge(AOI_flags, how="left", left_index=True,
    ↪ right_index=True)

for aoi in AOI_flags.columns:
    # DWELL PROPORTION IN FLIGHT
    dwell_prop_name = "DwProportionalTime_" + aoi

    # this code doesn't consider the sampling time:
    # AOIprop = (AOIcount*sample_time)/(TotalCount*sample_time)
    # Therefore AOIprop = AOIcount/TotalCount

    # create aoi cumulative dwell line count in flight
    e_dwell["AOITotalTime"] = e_dwell.groupby("id")[aoi].cumsum
        ↪ ()
    # create elapsed time since begining of flight
    e_dwell["TotalTime"] = e_dwell.groupby("id")["Timestamp"].
        ↪ cumcount() + 1
    # calculate dwell proportion for AOI in flight
    e_dwell[dwell_prop_name] = e_dwell["AOITotalTime"] / e_dwell
        ↪ ["TotalTime"]

    # DWELL PROPORTION IN TASK
    dwell_prop_name = "DwProportionalTimeTask_" + aoi
    # create aoi cumulative dwell line count in flight
    e_dwell["AOITotalTime"] = e_dwell.groupby(["id", "TaskCount"
        ↪ ])[aoi].cumsum()
    # create elapsed time count since begining of task
    e_dwell["TotalTime"] = e_dwell.groupby(["id", "TaskCount"])[
        "Timestamp"
    ].cumcount()
    # calculate dwell proportion for AOI in task
    e_dwell[dwell_prop_name] = e_dwell["AOITotalTime"] / e_dwell
```

```
        ↪ ["TotalTime"]

    # DWELL PROPORTION IN WINDOW
    for i in range(len(windows)):
        dwell_prop_name = "DwProportionalTimeWin_" + str(windows
            ↪ [i]) + "_" + aoi
        # create aoi cumulative dwell time in window
        e_dwell["AOITotalTime"] = e_dwell.groupby("id")[aoi].
            ↪ apply(
            lambda x: x.rolling(int(windows_sizes[i])).sum()
        )
        # create elapsed time since begining of window
        e_dwell["TotalTime"] = e_dwell.groupby("id")["Timestamp"
            ↪ ].apply(
            lambda x: x.rolling(int(windows_sizes[i])).count() +
                ↪ 1
        )
        # calculate dwell proportion for AOI in window
        e_dwell[dwell_prop_name] = e_dwell["AOITotalTime"] /
            ↪ e_dwell["TotalTime"]

drop_cols = [
    "AOI",
    "Task",
    "TaskCount",
    "AOIChange",
    "AOI_0",
    "AOI_1",
    "AOI_2",
    "AOI_3",
    "AOI_4",
    "AOI_5",
    "AOI_6",
    "AOI_7",
    "AOITotalTime",
    "TotalTime",
    ]

    e_dwell = e_dwell.drop(columns=drop_cols)
    return e_dwell
```

Listing F.7: Fixation Features Calculator

```
import pandas as pd
import sys
import numpy as np

sys.path.insert(1, "../../src/")
import auxiliary as aux
import autoregressor as autoreg


def calculate_fixades(e_gaze_metadata, e_task, gcor,
   ↪ seconds_per_row):
    # Constants and Definitions
    sampling_period = 1000 * seconds_per_row # ms
    windows = pd.Series([1, 5, 10]) # this is in seconds
    windows_sizes = 1000 * windows / sampling_period # this is in
       ↪ number of rows

    # feature base creation
    e_fixades = e_gaze_metadata
    e_fixades = e_fixades.merge(
        e_task,
        on=["id", "Timestamp"],
        how="inner",
        left_index=True,
        right_index=True,
        validate="one_to_one",
    )
    e_fixades = e_fixades.merge(
        gcor,
        on=["id", "Timestamp"],
        how="inner",
        left_index=True,
        right_index=True,
        validate="one_to_one",
    )
    # select cols
    e_fixades = e_fixades[
        [
            "id",
            "Timestamp",
            "GazeMoveType",
            "GazeMoveCount",
            "Task",
            "TaskCount",
            "GazeX",
            "GazeY",
        ]
    ]

    # Gaze Movement Duration

    e_fixades["GmDuration"] = (
        e_fixades.groupby(["id", "GazeMoveCount"])["Timestamp"].
           ↪ cumcount() + 1
    )
    e_fixades["GmDuration"] = e_fixades["GmDuration"] *
        ↪ sampling_period # miliseconds

    # GM Std (with leakage prevention and considering only last GM
        ↪ value)
    # select only the duration points regarding the end of the GM

    # flag rows to include in STD calculation
    e_fixades["MaxDuration"] = e_fixades.groupby(["id", "
        ↪ GazeMoveCount"])[
        "GmDuration"
    ].transform("max")
```

```
e_fixades.loc[
    e_fixades["MaxDuration"] != e_fixades["GmDuration"], "
        ↪ MaxDuration"
] = np.nan
e_fixades["MaxDuration"] = e_fixades["MaxDuration"].shift(1)

# actually calculate Std
e_fixades["GmDurationStd"] = e_fixades.groupby("id")["
    ↪ MaxDuration"].apply(
    lambda x: x.expanding().std()
)

# drop aux cols
e_fixades = e_fixades.drop(columns="MaxDuration")

# Gaze Movement Travel distance

e_fixades["DeltaX"] = (
    e_fixades.groupby("id")["GazeX"].shift(-1) - e_fixades["
        ↪ GazeX"]
)
e_fixades["DeltaY"] = (
    e_fixades.groupby("id")["GazeY"].shift(-1) - e_fixades["
        ↪ GazeY"]
)

# create euclidean distance
e_fixades["LocalDistance"] = np.sqrt(
    e_fixades["DeltaX"] * e_fixades["DeltaX"]
    + e_fixades["DeltaY"] * e_fixades["DeltaY"]
)
e_fixades["GmTravelDistance"] = e_fixades.groupby(["id", "
    ↪ GazeMoveCount"])[
    "LocalDistance"
].cumsum()
# e_fixades['GmAmplitude'] = e_fixades.groupby(['id','
    ↪ GazeMoveCount'])["Distance"].apply(_amplitude)

# flag rows to include in STD calculation
e_fixades["MaxTravelDistance"] = e_fixades.groupby(["id", "
    ↪ GazeMoveCount"])[
    "GmTravelDistance"
].transform("max")
e_fixades.loc[
    e_fixades["MaxTravelDistance"] != e_fixades["
        ↪ GmTravelDistance"],
    "MaxTravelDistance",
] = np.nan
e_fixades["MaxTravelDistance"] = e_fixades["MaxTravelDistance"].
    ↪ shift(1)

# actually calculate Std
e_fixades["GmTravelDistanceStd"] = e_fixades.groupby("id")[
    "MaxTravelDistance"
].apply(lambda x: x.expanding().std())

# drop aux cols
e_fixades = e_fixades.drop(
    columns=["MaxTravelDistance", "DeltaX", "DeltaY", "
        ↪ LocalDistance"]
)

# Frequency

# create frequency flags
e_fixades["GazeChangeFlag"] = (
    e_fixades.groupby("id")["GazeMoveCount"].shift(-1) -
        ↪ e_fixades["GazeMoveCount"]
).fillna(0)
e_fixades["GazeChangeFlag"] = (e_fixades["GazeChangeFlag"] != 0)
    ↪ .astype(int)
```

```
## Fixation

# fixation flags
e_fixades["FixFlag"] = (e_fixades["GazeMoveType"] == "Fixation")
   ↪ .astype(int)
e_fixades["FixCountFlag"] = e_fixades["GazeChangeFlag"] &
   ↪ e_fixades["FixFlag"]

# Full Flight

# create fixation count
e_fixades["FixCount"] = e_fixades.groupby("id")["FixCountFlag"].
   ↪ cumsum()

# duration until now to calculate frequency
e_fixades["Duration"] = e_fixades.groupby("id")["Timestamp"].
   ↪ cumcount() + 1
e_fixades["Duration"] = e_fixades["Duration"] * sampling_period
   ↪ # miliseconds

# frequency calculation
e_fixades["FixFreq"] = e_fixades["FixCount"] / e_fixades["
   ↪ Duration"]

# Task

# create fixation count
e_fixades["FixCount"] = e_fixades.groupby(["id", "TaskCount"])[
    "FixCountFlag"
].cumsum()

# duration until now to calculate frequency
e_fixades["Duration"] = (
    e_fixades.groupby(["id", "TaskCount"])["Timestamp"].cumcount
       ↪ () + 1
)
```

```
e_fixades["Duration"] = e_fixades["Duration"] * sampling_period
    ↪ # miliseconds

# frequency calculation
e_fixades["FixFreqTask"] = e_fixades["FixCount"] / e_fixades["
    ↪ Duration"]

# Window

for i in range(len(windows_sizes)):

    # define columns names and window for agregation
    freq_name = "FixFreqWin" + str(windows[i]) + "s"

    # fixations count until now in window
    e_fixades["FixCount"] = e_fixades.groupby("id")["
        ↪ FixCountFlag"].apply(
        lambda x: x.rolling(int(windows_sizes[i])).sum()
    )

    # duration until now in window to calculate frequency
    e_fixades["Duration"] = e_fixades.groupby("id")["Timestamp"
        ↪ ].apply(
        lambda x: x.rolling(int(windows_sizes[i])).count() + 1
    )
    e_fixades["Duration"] = e_fixades["Duration"] *
        ↪ sampling_period # miliseconds

    # frequency calculation
    e_fixades[freq_name] = e_fixades["FixCount"] / e_fixades["
        ↪ Duration"]

# drop aux cols
e_fixades = e_fixades.drop(
    columns=["Duration", "FixCount", "FixFlag", "FixCountFlag"]
)
```

```
## Saccade

# saccade flags
e_fixades["SacFlag"] = (e_fixades["GazeMoveType"] == "Saccade").
    ↪ astype(int)
e_fixades["SacCountFlag"] = (
    e_fixades["GazeChangeFlag"] & e_fixades["SacFlag"]
).cumsum()

# Full Flight

# create sacadde count
e_fixades["SacCount"] = e_fixades.groupby("id")["SacCountFlag"].
    ↪ cumsum()

# duration until now to calculate frequency
e_fixades["Duration"] = e_fixades.groupby("id")["Timestamp"].
    ↪ cumcount() + 1
e_fixades["Duration"] = e_fixades["Duration"] * sampling_period
    ↪ # miliseconds

# frequency calculation
e_fixades["SacFreq"] = e_fixades["SacCount"] / e_fixades["
    ↪ Duration"]

# Task

# create sacadde count
e_fixades["SacCount"] = e_fixades.groupby(["id", "TaskCount"])[
    "SacCountFlag"
].cumsum()

# duration until now to calculate frequency
e_fixades["Duration"] = e_fixades.groupby(["id", "TaskCount"])[
    "Timestamp"
].cumcount()
e_fixades["Duration"] = e_fixades["Duration"] * sampling_period
    ↪ # miliseconds

# frequency calculation
e_fixades["SacFreqTask"] = e_fixades["SacCount"] / e_fixades["
    ↪ Duration"]

# Window

for i in range(len(windows_sizes)):

    # define columns names and window for agregation
    freq_name = "SacFreqWin" + str(windows[i]) + "s"

    # fixations count until now in window
    e_fixades["SacCount"] = e_fixades.groupby("id")["
        ↪ SacCountFlag"].apply(
        lambda x: x.rolling(int(windows_sizes[i])).sum()
    )

    # duration until now in window to calculate frequency
    e_fixades["Duration"] = e_fixades.groupby("id").apply(
        lambda x: x.rolling(int(windows_sizes[i])).count() + 1
    )
    e_fixades["Duration"] = e_fixades["Duration"] *
        ↪ sampling_period # miliseconds

    # frequency calculation
    e_fixades[freq_name] = e_fixades["SacCount"] / e_fixades["
        ↪ Duration"]

# drop aux cols
e_fixades = e_fixades.drop(
    columns=["Duration", "SacCount", "SacFlag", "SacCountFlag"]
)
```

```
e_fixades_new = autoreg.make_autoregressive(
    df=e_fixades,
    grouping_col="id",
    columns=["GmDuration", "GmTravelDistance"],
    operations=["mean", "quantile_10", "quantile_90"],
    window_sizes_seconds=[5],
    window_lags_seconds=[0],
    task_window=True,
)

e_fixades_new = e_fixades_new[
    [
        "id",
        "Timestamp",
        "GmDuration",
        "GmDurationStd",
        "GmTravelDistance",
        "GmTravelDistanceStd",
        "GazeChangeFlag",
        "FixFreq",
        "FixFreqTask",
        "FixFreqWin1s",
        "FixFreqWin5s",
        "FixFreqWin10s",
        "SacFreq",
        "SacFreqTask",
        "SacFreqWin1s",
        "SacFreqWin5s",
        "SacFreqWin10s",
        "GmDuration_roll5s_lag0s_mean",
        "GmDuration_roll5s_lag0s_quantile_10",
        "GmDuration_roll5s_lag0s_quantile_90",
        "GmTravelDistance_roll5s_lag0s_mean",
        "GmTravelDistance_roll5s_lag0s_quantile_10",
        "GmTravelDistance_roll5s_lag0s_quantile_90",
            "GmTravelDistance_task_expanding_mean",
            "GmTravelDistance_task_expanding_quantile_10",
            "GmTravelDistance_task_expanding_quantile_90",
        ]
    ]
    return e_fixades_new


# auxiliary.validete_bunda(e_fixades)
```

Listing F.8: Movement Features Calculator

```
import sys

sys.path.append("../../src")

import numpy as np
import pandas as pd
import autoregressor as autoregressor

# from IPython.core.display import display



def calculate_movement(gcor, e_spine, e_task):

    # break into raw and smooth (moving averaged) gaze
    # raw_gcor = gcor[["id", "Timestamp", "GazeX", "GazeY"]].copy()
    smooth_gcor = gcor[["id", "Timestamp", "GazeX", "GazeY"]].copy()

    # rename cols
    smooth_gcor.columns = ["id", "Timestamp", "GazeX", "GazeY"]

    # Feature Creation

    # calculate features for smooth gaze
    gcor = smooth_gcor

    # remove dropabble rows
```

```
gcor = gcor[~gcor["GazeX"].isna()].reset_index(drop=True)

# Create Time Delta
gcor["TimeDelta"] = (
    gcor.groupby("id")["Timestamp"].shift(-1) - gcor["Timestamp"
        ↪ ]
).fillna(0)

## Pure Features

## Visual Angle
gcor["VisualAngularGazeX"] = gcor["GazeX"].apply(
    ↪ _px_to_visual_angle)
gcor["VisualAngularGazeY"] = gcor["GazeY"].apply(
    ↪ _px_to_visual_angle)

## Speeds and Accelerations

# derivative variables
# ISSUE - THIS ASSUMES UNIFORM TIME DELTAS
dt = 20 / 1000 # 20ms in Seconds

# 1. Pixel Speeds
gaze_x = gcor.groupby("id")["GazeX"]
gaze_y = gcor.groupby("id")["GazeY"]

## 1.1. Rectangular
# speed as derivative of position
gcor["SpeedX"] = gaze_x.transform(np.gradient, dt, edge_order=2)
gcor["SpeedY"] = gaze_y.transform(np.gradient, dt, edge_order=2)

speed_x = gcor.groupby("id")["SpeedX"] # px/sec
speed_y = gcor.groupby("id")["SpeedY"] # px/sec

# acceleration as derivative of speed
gcor["AccelX"] = speed_x.transform(np.gradient, 2) # px/sec^2
```

```
gcor["AccelY"] = speed_y.transform(np.gradient, 2) # px/sec^2

## 1.2. Polar
# velocity as composed speed_x and speed_y in polar coordinates
# angle theta in radians
gcor["VelocityAngleTheta"] = np.arctan2(gcor["SpeedY"], gcor["
    ↪ SpeedX"])
gcor["VelocityModule"] = np.sqrt(gcor["SpeedX"] ** 2 + gcor["
    ↪ SpeedY"] ** 2)

# speed of rotation of velocity vector
velocity_theta = gcor.groupby("id")["VelocityAngleTheta"] # rad
gcor["VelocityRotationAlpha"] = velocity_theta.transform(
    np.gradient, dt, edge_order=2
)

# 2. Visual Angle Speeds

## 2.1 Rectangular

visual_angular_gaze_x = gcor.groupby("id")["VisualAngularGazeX"]
visual_angular_gaze_y = gcor.groupby("id")["VisualAngularGazeY"]

# speed as derivative of position
gcor["VisualAngularSpeedX"] = visual_angular_gaze_x.transform(
    np.gradient, dt, edge_order=2
)
gcor["VisualAngularSpeedY"] = visual_angular_gaze_y.transform(
    np.gradient, dt, edge_order=2
)

visual_angular_speed_x = gcor.groupby("id")["VisualAngularSpeedX
    ↪ "] # px/sec
visual_angular_speed_y = gcor.groupby("id")["VisualAngularSpeedY
    ↪ "] # px/sec
```

```
# acceleration as derivative of speed
gcor["VisualAngularAccelX"] = visual_angular_speed_x.transform(
    np.gradient, 2
) # px/sec^2
gcor["VisualAngularAccelY"] = visual_angular_speed_y.transform(
    np.gradient, 2
) # px/sec^2

# Direction Changes

# direction changes in X and Y coordinates (1 indicates
    ↪ direction change)
gcor["FlipX"] = (
    gcor.groupby("id")["SpeedX"].apply(np.sign)
    - gcor.groupby("id")["SpeedX"].shift(1).apply(np.sign)
).fillna(0)
gcor["FlipX"] = (gcor["FlipX"] != 0).astype(int)
gcor["FlipY"] = (
    gcor.groupby("id")["SpeedY"].apply(np.sign)
    - gcor.groupby("id")["SpeedY"].shift(1).apply(np.sign)
).fillna(0)
gcor["FlipY"] = (gcor["FlipY"] != 0).astype(int)

# direction changes in both coordinates
gcor["FlipXY"] = gcor["FlipX"] & gcor["FlipY"]

# frequency of direction change
# gcor["FlipXFreq"] = _calculate_windowed_freq(df=gcor, var_col
    ↪ ="FlipX", window_size_seconds=2)
# gcor["FlipYFreq"] = _calculate_windowed_freq(df=gcor, var_col
    ↪ ="FlipY", window_size_seconds=2)
gcor["FlipXYFreq"] = _calculate_windowed_freq(
    df=gcor, var_col="FlipXY", window_size_seconds=2
)

# display(gcor)
# returning dropped rows (don't have smoothed gaze data)
gcor = e_spine.merge(gcor, on=["id", "Timestamp"], how="left")

# display(gcor)

# drop aux cols
aux_cols = ["TimeDelta", "TimeIndex"]
e_movement = gcor.drop(columns=aux_cols)

# determine final dfs and make Autoregs

# remove dropabble rows
e_movement = e_movement[~e_movement["GazeX"].isna()].reset_index
    ↪ (drop=True)

# Create Time Delta
e_movement["TimeDelta"] = (
    e_movement.groupby("id")["Timestamp"].shift(-1) - e_movement
        ↪ ["Timestamp"]
).fillna(0)

# display(e_movement)

# create dataframes to save autoregs
## 1. gaze positions
e_gaze_positions = e_movement[["id", "Timestamp", "GazeX", "
    ↪ GazeY"]].copy()
## 2. visual angular gaze positions
e_visual_angular_gaze_positions = e_movement[
    ["id", "Timestamp", "VisualAngularGazeX", "
        ↪ VisualAngularGazeY"]
].copy()
## 3. velocity
e_velocity = e_movement[
    [
       "id",
       "Timestamp",
       "SpeedX",
       "SpeedY",
       "AccelX",
       "AccelY",
       "VelocityAngleTheta",
       "VelocityModule",
       "VelocityRotationAlpha",
   ]
].copy()
## 4. direction change
e_direction_change = e_movement[
    ["id", "Timestamp", "FlipX", "FlipY", "FlipXY", "FlipXYFreq"
       ↪ ]
].copy()

# Add Tasks to calculate Autoregs
## 1. gaze positions
e_gaze_positions = e_gaze_positions.merge(
    e_task, on=["id", "Timestamp"], how="left"
)
## 2. visual angular gaze positions
e_visual_angular_gaze_positions =
    ↪ e_visual_angular_gaze_positions.merge(
    e_task, on=["id", "Timestamp"], how="left"
)
## 3. velocity
e_velocity = e_velocity.merge(e_task, on=["id", "Timestamp"],
    ↪ how="left")
## 4. direction change
e_direction_change = e_direction_change.merge(
    e_task, on=["id", "Timestamp"], how="left"
)

# Calculate Autoregs
```

```
## 1. gaze positions
# Calculate Autoregs
e_gaze_positions = autoregressor.make_autoregressive(
    df=e_gaze_positions,
    grouping_col="id",
    columns=["GazeX", "GazeY"],
    operations=["mean", "median", "std"],
    window_sizes_seconds=[2, 5, 10],
    window_lags_seconds=[0, 5],
    task_window=True,
)

# recover all spine rows
e_gaze_positions = e_spine.merge(
    e_gaze_positions, on=["id", "Timestamp"], how="left"
)
# drop tasks
e_gaze_positions = e_gaze_positions.drop(columns=["Task", "
    ↪ TaskCount"])

## 3. velocity
# display(e_velocity.info())
# display(e_velocity)
e_velocity = autoregressor.make_autoregressive(
    df=e_velocity,
    grouping_col="id",
    columns=[
        "SpeedX",
        "SpeedY",
        "AccelX",
        "AccelY",
        "VelocityRotationAlpha",
        "VelocityAngleTheta",
        "VelocityModule",
    ],
    operations=["mean", "std", "quantile_90"],
    window_sizes_seconds=[2],
    window_lags_seconds=[0],
    task_window=True,
)

# recover all spine rows
e_velocity = e_spine.merge(e_velocity, on=["id", "Timestamp"],
    ↪ how="left")
# drop tasks
e_velocity = e_velocity.drop(columns=["Task", "TaskCount"])

## 4. direction change
autoreg_cols = list(
    set(e_direction_change.columns) - set(["id", "Timestamp", "
       ↪ Task", "TaskCount"])
)
e_direction_change = autoregressor.make_autoregressive(
    df=e_direction_change,
    grouping_col="id",
    columns=autoreg_cols,
    operations=["mean", "std"],
    window_sizes_seconds=[5],
    window_lags_seconds=[0],
    task_window=True,
)

# recover all spine rows
e_direction_change = e_spine.merge(
    e_direction_change, on=["id", "Timestamp"], how="left"
)
# drop tasks
e_direction_change = e_direction_change.drop(columns=["Task", "
    ↪ TaskCount"])

return e_gaze_positions, e_velocity, e_direction_change
```

```
## AUX FUNCTIONS

# conversion to visual angle
def _px_to_visual_angle(size_in_px: int):

    # ISSUE: find GARMIN size (1216/764)

    h = 25 # Monitor height in cm
    d = 60 # Distance between monitor and participant in cm
    r = 768 # Vertical resolution of the monitor

    # Calculate the number of degrees that correspond to a single
       ↪ pixel
    # generally, a very small value, something like 0.03.
    deg_per_px = np.rad2deg(np.arctan2(0.5 * h, d)) / (0.5 * r)

    # Calculate the size of the stimulus in degrees
    size_in_deg = size_in_px * deg_per_px

    return size_in_deg


def _calculate_windowed_freq(df, var_col: str, window_size_seconds:
    ↪  int):

    window_freq = str(window_size_seconds) + "s"

    # set Timestamp as index
    df["TimeIndex"] = pd.to_timedelta(df["Timestamp"], unit="ms")
    df = df.set_index(["TimeIndex"])

    # calculate vars and time operations
    var_rolling_sum = (
        df.groupby("id")[var_col].rolling(min_periods=0, window=
            ↪ window_freq).sum()
    )
    time_rolling_sum = (
        df.groupby("id")["TimeDelta"].rolling(min_periods=0, window=
            ↪ window_freq).sum()
    )

    # set id, Timestamp as index
    df = df.set_index(["id"], append=True)
    df = df.reorder_levels(["id", "TimeIndex"])

    # save operations to columns
    df[var_col + "Freq"] = var_rolling_sum / (time_rolling_sum /
        ↪ 1000)

    # reset index and drop aux cols
    df = df.reset_index()
    return df[var_col + "Freq"]
```

Listing F.9: Target Features Calculator

```
import sys

sys.path.append("../../src")

import numpy as np
import pandas as pd

# from IPython.core.display import display
# from loguru import logger
import auxiliary as aux



def calculate_target(
    e_task_with_checks, back_prop_seconds_list: list,
        ↪ seconds_per_row: float
):
   # change taxonomy
   e_target = e_task_with_checks

   # flag check tasks and create Target
   e_target["Target"] = (e_target.Task == "CHECK").astype(int)

   # back propagate Target
   for back_prop_seconds in back_prop_seconds_list:

       col_name = "TargetBackProp_" + str(back_prop_seconds) + "s"

       # change taxonomy and create column
       e_target[col_name] = e_target["Target"]

       ## replace zeros with NaNs
       e_target[col_name] = e_target[col_name].replace(0, np.nan)

       ## fill with limit equal to number of rows to back-propagate
           ↪  for
       back_prop_rows = int(back_prop_seconds / aux.get_configs()["
           ↪ seconds_per_row"])
       e_target[col_name] = e_target[col_name].fillna(
           method="bfill", limit=back_prop_rows
       )

       # replace remaining NaNs with zeros
       e_target[col_name] = e_target[col_name].fillna(0)

   # drop aux cols
   e_target = e_target.drop(columns=["Task", "TaskCount"])

   return e_target
```

F.2.3 Data Processing

Listing F.10: Gaze Table Constructor

```
import pandas as pd
from loguru import logger
import numpy as np

## Objectives

# - NaN values in Gaze X, Gaze Y: interpolate given conditions
# - Eye Movement: remove unclassified, eyes not found?
# - Automap scores: remove based on percentile
# - Eye Rec_name: associate with pilot name => create ID


def calculate_clean_gaze(id_info, r_tobii_gaze, tasks_table,
    ↪ result_matches):

    # change taxonomy
    # dirty dataset: includes NaN values and all raw data
    e_dirty_gaze = r_tobii_gaze

    # rename columns
    e_dirty_gaze.columns = [
        "Rec_name",
        "Rec_dur",
        "Timestamp",
        "Eye_movement",
        "Automap_score",
        "Gaze_X",
        "Gaze_Y",
    ]

    # fix data types

    ## floats
    # Gaze X is a float cause it contains NaN values for now
    # Gaze Y is a float cause it contains NaN values for now
    # replace commas by dots
```

```
e_dirty_gaze["Rec_dur"] = (
    e_dirty_gaze["Rec_dur"].str.replace(",", ".").astype(float)
)
e_dirty_gaze["Automap_score"] = (
    e_dirty_gaze["Automap_score"].str.replace(",", ".").astype(
        ↪ float)
)

## strings
e_dirty_gaze["Eye_movement"] = e_dirty_gaze["Eye_movement"].
    ↪ astype("string")
e_dirty_gaze["Rec_name"] = e_dirty_gaze["Rec_name"].astype("
    ↪ string")

# format columns
e_dirty_gaze["Rec_num"] = (
    e_dirty_gaze["Rec_name"].apply(lambda x: x.strip("Recording"
        ↪ )).astype(int)
)
tasks_table["Task"] = tasks_table["Task"].astype(str)

# pre-process tasks
tasks_table = tasks_table.merge(
    id_info[["ID", "NewId"]], left_on="id", right_on="ID", how="
        ↪ left"
)
tasks_table = tasks_table.drop(columns=["id", "ID"])
tasks_table.columns = ["Timestamp", "Task", "TaskCount", "id"]

# filter by known recordings
known_ids = result_matches["NewId"]
tasks_table = tasks_table[tasks_table["id"].isin(known_ids)]

# Pre-processing

## Removing Duplicates
```

```
e_dirty_gaze = e_dirty_gaze.drop_duplicates(subset=["Rec_name",
    ↪ "Timestamp"])

## Rows to Drop due to Low Automap Score (LAMS)

# logger.info("Before cleaning by Automap Score")
# display(e_dirty_gaze["Eye_movement"].value_counts())

# remove gaze points with score lower than 10 percentile of all
    ↪ scores
automap_score_cut_percentile = 0.25

# find percentile
minimum_automap_score = e_dirty_gaze["Automap_score"].quantile(
    automap_score_cut_percentile
)

# store droppabble columns in column
## create column
e_dirty_gaze["Drop_for_LAMS"] = 0
e_dirty_gaze.loc[
    e_dirty_gaze["Automap_score"] < minimum_automap_score, "
        ↪ Drop_for_LAMS"
] = 1

# drop aux columns
# e_dirty_gaze = e_dirty_gaze.drop(columns="Automap_score")

logger.info("Found Droppable by LAMS")

## Othter Null Rows: Interpolate if Gap size is Small, otherwise
    ↪  Drop

# calculating for dataset without lams
## mind to not reset index
```

```
edg_no_lams = e_dirty_gaze[e_dirty_gaze["Drop_for_LAMS"] == 0]

# add mask indicating where to interpolate
# do not inerpolate if interval without gaze point is bigger
    ↪ than max gap ms
max_gap_ms = 50
interp_mask, drop_mask = _calculate_interpolate_or_drop_masks(
    orig_df=edg_no_lams, max_gap_ms=max_gap_ms
)

## the NaN values at this point should either: be dropped,
# or be interpolated, so interp_mask and drop_mask are
    ↪ complimentary
## (added they amount for all NaN points)

# store droppabble rows in column
edg_no_lams["Drop_for_Null&LGS"] = drop_mask.astype(int)

# create column to indicate values to be interpolated
edg_no_lams["Interpolated"] = interp_mask.astype(int)

# display amount of gazes
logger.info("Found Droppable by being Null and having LGS (Gap
    ↪ too Big)")

# add to e_dirty gaze
e_dirty_gaze["Drop_for_Null&LGS"] = edg_no_lams["Drop_for_Null&
    ↪ LGS"]
e_dirty_gaze["Drop_for_Null&LGS"] = (
    e_dirty_gaze["Drop_for_Null&LGS"].fillna(0).astype(int)
)

e_dirty_gaze["Interpolated"] = edg_no_lams["Interpolated"]
e_dirty_gaze["Interpolated"] = e_dirty_gaze["Interpolated"].
    ↪ fillna(0).astype(int)
```

```
## Flag all Dropabble Rows

e_dirty_gaze["Drop_for_All"] = (
    e_dirty_gaze["Drop_for_LAMS"] | e_dirty_gaze["Drop_for_Null&
        ↪ LGS"]
)

# Processing

## Interpolation of missing Values

# interpolate
e_dirty_gaze["Gaze_X"] = e_dirty_gaze["Gaze_X"].interpolate()
e_dirty_gaze["Gaze_Y"] = e_dirty_gaze["Gaze_Y"].interpolate()

# display amount of gazes
logger.info("Interpolated Gaze X and Gaze Y")

# forward fill eye movement types when adequate (UNC, ENF)
e_dirty_gaze["Eye_movement"] = (
    e_dirty_gaze["Eye_movement"]
    .replace("Unclassified", np.nan)
    .fillna(method="ffill")
)
e_dirty_gaze["Eye_movement"] = (
    e_dirty_gaze["Eye_movement"]
    .replace("EyesNotFound", np.nan)
    .fillna(method="ffill")
)
e_dirty_gaze["Automap_score"] = e_dirty_gaze["Automap_score"].
    ↪ fillna(0)

## Filtering Known Recordings

# Recording038, Recording039, Recording42, Recording43,
# Recording44 exist only on Tobii (not even on Matlab), so were
    ↪ removed
# Recording083 (Anyl) does not exist for Boris

known_recs = result_matches["Rec"]

# create column indicating known recordings
e_dirty_gaze["Known_rec"] = e_dirty_gaze.Rec_num.isin(known_recs
    ↪ ).astype(int)

# filter by known recordings
e_dirty_gaze = e_dirty_gaze[e_dirty_gaze.Known_rec == 1].
    ↪ reset_index(drop=True)

# reset index
e_dirty_gaze = e_dirty_gaze.reset_index(drop=True)

# drop aux columns
e_dirty_gaze = e_dirty_gaze.drop(columns="Known_rec")

# add new id
e_dirty_gaze = e_dirty_gaze.merge(
    id_info[["NewId", "Rec"]].drop_duplicates(),
    left_on="Rec_num",
    right_on="Rec",
    how="left",
)
e_dirty_gaze = e_dirty_gaze.rename(columns={"NewId": "id"})

e_dirty_gaze[
    ~(e_dirty_gaze["Drop_for_All"] == 1) & (e_dirty_gaze["
        ↪ Interpolated"] == 1)
]

# Drop Rows given Previous Criteria
```

```
prev_size = e_dirty_gaze.shape[0]

# add tasks so values are at bottom
e_dirty_gaze = e_dirty_gaze.append(tasks_table)

# sort values to intertwine at correct positions
e_dirty_gaze = e_dirty_gaze.sort_values(["id", "Timestamp"]).
    ↪ reset_index(drop=True)

# flag rows rows inputted in previous step
e_dirty_gaze["InputtedRow"] = (~e_dirty_gaze["TaskCount"].isna()
    ↪ ).astype(int)

# foward fill inserted task and TaskCount values
# Task and Task Count remain constant until change happens
e_dirty_gaze["Task"] = e_dirty_gaze.groupby("id")["Task"].ffill
    ↪ ()
e_dirty_gaze["TaskCount"] = e_dirty_gaze.groupby("id")["
    ↪ TaskCount"].ffill()

# fill empty initial tasks
e_dirty_gaze["Task"] = e_dirty_gaze["Task"].fillna("T0")
e_dirty_gaze["TaskCount"] = e_dirty_gaze["TaskCount"].fillna(0)

# drop inputted rows
e_dirty_gaze = e_dirty_gaze.drop(e_dirty_gaze[e_dirty_gaze.
    ↪ InputtedRow == 1].index)
assert e_dirty_gaze.shape[0] == prev_size

# drop aux columns
e_dirty_gaze = e_dirty_gaze.drop(columns="InputtedRow")

# Effectively Dropping Rows

e_dirty_gaze = e_dirty_gaze[(e_dirty_gaze["Drop_for_All"] == 0)]
e_dirty_gaze = e_dirty_gaze.reset_index(drop=True)
```

```
e_dirty_gaze["Eye_movement"].value_counts()

# Remove Tasks too Small and with too Little Rows

### Tasks too Small

# calculate task size
e_dirty_gaze["TaskSize"] = e_dirty_gaze.groupby(["id", "
    ↪ TaskCount"])[
    "Timestamp"
].transform(lambda x: x.max() - x.min())

# define threshold from inspection
min_task_size = 3000 # in milliseconds

protected_small_tasks = ["CHECK", "T0"]

# flag small tasks and protected tasks (tasks that can be small,
    ↪  without problem)
unproteced_mask = ~e_dirty_gaze["Task"].isin(
    ↪ protected_small_tasks)
small_task_mask = (e_dirty_gaze["TaskSize"] < min_task_size).
    ↪ astype(int)
drop_mask = (unproteced_mask) & (small_task_mask)

# dropping values
e_dirty_gaze["Drop_for_TTS"] = drop_mask.astype(int)

### Tasks with too Few Points

# calculate task size
e_dirty_gaze["TaskCount"] = e_dirty_gaze.groupby(["id", "
    ↪ TaskCount"])[
    "Timestamp"
].transform("count")
```

```
# define threshold from inspection
min_task_count = 1 # in milliseconds

# dropping values
e_dirty_gaze["Drop_for_TFP"] = (e_dirty_gaze["TaskCount"] <=
    ↪ min_task_count).astype(
    int
)

# drop aux columns
e_dirty_gaze = e_dirty_gaze.drop(columns=["TaskCount", "TaskSize
    ↪ "])

# Remove Gaps too Big

e_dirty_gaze["DeltaTimestamp"] = e_dirty_gaze.groupby("id")["
    ↪ Timestamp"].apply(
    lambda x: x - x.shift(1)
)

# cutoff at gaps above 3000ms
e_new_gaze = e_dirty_gaze.copy()

e_new_gaze = e_dirty_gaze[e_dirty_gaze["DeltaTimestamp"] < 3000]

e_new_gaze["DeltaTimestamp"] = e_new_gaze.groupby("id")["
    ↪ Timestamp"].apply(
    lambda x: x - x.shift(1)
)

# Effectively Dropping Rows

e_clean_gaze = e_dirty_gaze[
    (e_dirty_gaze["Drop_for_All"] == 0)
    & (e_dirty_gaze["Drop_for_TTS"] == 0)
```

```
        & (e_dirty_gaze["Drop_for_TFP"] == 0)
    ]
    e_clean_gaze = e_clean_gaze.reset_index(drop=True)

    GCOR = e_clean_gaze[["id", "Timestamp", "Gaze_X", "Gaze_Y"]].
        ↪ copy()
    GCOR.columns = ["id", "Timestamp", "GazeX", "GazeY"]

    return e_clean_gaze, GCOR



### AUX FUNCTIONS



def _get_gaze_gaps(g):
    gaze_gaps = g.cumsum() - g.cumsum().where(g == 0).ffill().fillna
        ↪ (0).astype(int)
    return gaze_gaps



def _get_max_gaze_gap(df):
    df["not_zero"] = df["GazeGap"] > 0
    df["was_zero"] = df.groupby("Rec_name")["not_zero"].shift(1) ==
        ↪ 0
    df["start_max"] = df["not_zero"] & df["was_zero"]
    df["period"] = df.groupby("Rec_name")["start_max"].cumsum()
    df["MaxGazeGap"] = df.groupby(["Rec_name", "period"])["GazeGap"
        ↪ ].transform(max)
    df["MaxGazeGap"] = df[df["GazeGap"] > 0]["MaxGazeGap"]
    max_gaze_gap = df["MaxGazeGap"].fillna(0)
    return max_gaze_gap



def _calculate_interpolate_or_drop_masks(orig_df, max_gap_ms):

    df = orig_df.copy()
```

```
df["FlagNull"] = (df["Gaze_X"].isna()).astype(int)

# calculate time delta between gaze points, by pilot
df["Timestamp_next"] = df.groupby("Rec_name")["Timestamp"].shift
    ↪ (-1)
df["DeltaTime"] = df["Timestamp_next"] - df["Timestamp"]

# delta time for first timestamp in dataframe
df["DeltaTime"] = df["DeltaTime"].fillna(0)

# drop auxiliar column
df = df.drop(columns="Timestamp_next")

# time difference between current and previous gaze point when
    ↪ current gaze point is null
df["DeltaTimeNulled"] = df["FlagNull"] * df["DeltaTime"]

# calculate cumulative, zero resetting gaze gap
df["GazeGap"] = df.groupby("Rec_name")["DeltaTimeNulled"].apply(
    ↪ _get_gaze_gaps)

# calculate cumulative, zero resetting max gaze gap
df["MaxGazeGap"] = _get_max_gaze_gap(df=df)

# get interpolation mask as places where gaze does respects max
    ↪ gap
df["ShouldInterpolate"] = (df["FlagNull"] > 0) & (df["MaxGazeGap
    ↪ "] < max_gap_ms)

# mask of interpolation
interp_mask = df["ShouldInterpolate"].astype(int)
null_mask = df["FlagNull"].astype(int)

# mask of null points with gaps that are too large
drop_mask = (~interp_mask) & (null_mask)
```

```
    return interp_mask, drop_mask
```

Listing F.11: Gaze Table Smoothing

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler



def calculate_smooth_gaze(gcor):

    # apply smoothing
    gcor["SmoothGazeX"] = (
        gcor.groupby(["id"])["GazeX"]
        .apply(_scaled_moving_average, window=10, n_smooths=2)
        .reset_index()["GazeX"]
    )
    gcor["SmoothGazeY"] = (
        gcor.groupby(["id"])["GazeY"]
        .apply(_scaled_moving_average, window=10, n_smooths=2)
        .reset_index()["GazeY"]
    )

    return gcor


## AUX FUNCTIONS

# Moving Average Smothing
def _scaled_moving_average(
    raw_series, window: int, n_smooths: int, scale: bool = True,
):

    # change taxonomy
    smooth_series = raw_series.copy()

    # smooth over n_smooths times
    for i in range(n_smooths):
        # print("smoothing!")
        smooth_series = smooth_series.rolling(window=window).mean()

    # scale to original series values
    if scale:
        # print("scaling!")
        # convert to numpy
        smooth_series = np.array(smooth_series)
        # instantiate scaler class
        scaler = MinMaxScaler(
            feature_range=(np.nanmin(raw_series), np.nanmax(
               ↪ raw_series))
        )
        # scale array
        smooth_series = scaler.fit_transform(smooth_series.reshape
            ↪ (-1, 1))
        # unstack array to original shape
        smooth_series = smooth_series.reshape(len(smooth_series))
        # convert back to series
        smooth_series = pd.Series(smooth_series)

    return smooth_series
```

Listing F.12: Raw Data Aggregator

```
import pandas as pd
from loguru import logger
from tqdm.notebook import tqdm
import numpy as np



def calculate_rdt(clean_gaze, smooth, tasks_table):

    # change taxonomy
    rdt = clean_gaze

    # add other columns
```

```
rdt = rdt.drop(columns="Task")
rdt = rdt.merge(smooth, on=["id", "Timestamp"], how="left")

# Remove Raw Gaze, store only Smooth Gaze
# select and re-order columns
rdt = rdt[
    [
        "id",
        "Timestamp",
        "Eye_movement",
        "Automap_score",
        "SmoothGazeX",
        "SmoothGazeY",
        "Interpolated",
        "Rec_name",
        "Rec_dur",
    ]
]

# rename columns
rdt = rdt.rename(columns={"SmoothGazeX": "GazeX", "SmoothGazeY":
    ↪  "GazeY"})

# remove NaNs and rest index
rdt = rdt.dropna()
rdt = rdt.reset_index(drop=True)

# Add Tasks

logger.info("Adding Tasks")

prev_size = rdt.shape[0]

# add tasks so values are at bottom
rdt = rdt.append(tasks_table)
```

```
# sort values to intertwine at correct positions
rdt = rdt.sort_values(["id", "Timestamp"]).reset_index(drop=True
    ↪ )

# flag rows rows inputted in previous step
rdt["InputtedRow"] = (~rdt["TaskCount"].isna()).astype(int)

# foward fill inserted task and TaskCount values
# Task and Task Count remain constant until change happens
rdt["Task"] = rdt.groupby("id")["Task"].ffill()
rdt["TaskCount"] = rdt.groupby("id")["TaskCount"].ffill()

# fill empty initial tasks
rdt["Task"] = rdt["Task"].fillna("T0")
rdt["TaskCount"] = rdt["TaskCount"].fillna(0)

# drop inputted rows
rdt = rdt.drop(rdt[rdt.InputtedRow == 1].index)
assert rdt.shape[0] == prev_size

# drop aux columns
rdt = rdt.drop(columns="InputtedRow")

logger.info("Resampling with Constant Freq")
# Constant sample period: 20ms

logger.info("Breaking Pilots into Parts")
## Chunk pilots in parts with gaps in between them
# - When gap bigger than 1000ms
# - Only keep reasonable sizes chunks (bigger than 10.000ms)!

# gap size: won't interpolate for more than...
max_gap_size = 1000 # ms

# min timespan of part
min_part_time = 10000 # ms
```

```
rdt["Gap"] = rdt.groupby("id")["Timestamp"].shift(-1) - rdt["
    ↪ Timestamp"]
gap_threshold = max_gap_size
rdt["Split"] = (rdt["Gap"] > gap_threshold).astype(int)
# pure_ids = rdt["id"]
# e_old_id = rdt["id"].copy()

while rdt["Split"].any():

    # dropping first rows with split already (4 cases)
    rdt["CumCount"] = rdt.groupby("id")["Split"].cumcount() + 1
    rdt = rdt[rdt["Split"] != rdt["CumCount"]]
    rdt = rdt.drop(columns="CumCount")

    # creating new ids with "part"
    rdt["CumSplit"] = rdt.groupby("id")["Split"].cumsum()
    rdt["new_id"] = rdt["id"] + "_p" + rdt["CumSplit"].apply(str
        ↪ )

    # determining acceptable created parts
    sizes = rdt.groupby("new_id")["Timestamp"].apply(lambda x: x
        ↪ .max() - x.min())
    good_sizes = sizes[sizes >= min_part_time]

    # filtering by acceptable parts
    rdt = rdt[rdt["new_id"].isin(good_sizes.index)]

    # retrieve part numbers without gaps
    rdt["part"] = rdt.groupby("id")["Split"].cumsum()
    rdt["new_id"] = rdt["id"] + "_p" + rdt["part"].apply(str)

    # save new id as id
    rdt["id"] = rdt["new_id"]

    # find gaps
    rdt["Gap"] = rdt.groupby("id")["Timestamp"].shift(-1) - rdt[
        ↪ "Timestamp"]
    gap_threshold = max_gap_size
    rdt["Split"] = (rdt["Gap"] > gap_threshold).astype(int)

# create uniform id (# tirando popo)
new_id2 = rdt.id.drop_duplicates().reset_index()
new_id2["pid2"] = new_id2["id"].apply(lambda x: x.split("_p")
    ↪ [0])
new_id2["new_part"] = 1
new_id2["new_part"] = new_id2.groupby("pid2")["new_part"].cumsum
    ↪ () - 1
new_id2["new_id2"] = new_id2["pid2"] + new_id2["new_part"].apply
    ↪ (
    lambda x: "_p%02d" % x
)
new_id2 = new_id2[["id", "new_id2"]]

rdt = rdt.merge(new_id2, on=["id"], how="left")
rdt["id"] = rdt["new_id2"]

# reset index and drop aux cols
aux_cols = ["new_id", "part", "Gap", "Split", "CumSplit", "
    ↪ new_id2"]
rdt = rdt.drop(columns=aux_cols)
rdt = rdt.reset_index(drop=True)

logger.info("Creating Gaze Movement Count Column")
## Creating Gaze Movement Count

rdt["GazeMoveCount"] = rdt.groupby("id")["Timestamp"].cumcount()
    ↪  + 1

logger.info("Start every timestamp at zero")
## Start Every id at Timestamp zero
```

```
rdt["first_timestamp"] = rdt.groupby("id")["Timestamp"].
    ↪ transform("min")
rdt["NewTimestamp"] = rdt["Timestamp"] - rdt["first_timestamp"]
rdt["Timestamp"] = rdt["NewTimestamp"]

aux_cols = ["NewTimestamp", "first_timestamp"]
rdt = rdt.drop(columns=aux_cols)

## Interpolation
# - Smaller than 1000ms

logger.info("Creating new Spine")
### Add all intermediate timestamps (every 20ms)

# convert timestamp to timedelta
rdt["TimeIndex"] = pd.to_timedelta(rdt["Timestamp"], unit="ms")

# round original spine to nearest 20ms timestamp
rdt["TimeIndex"] = rdt["TimeIndex"].dt.round("20ms")

# create new spine
new_spine = pd.DataFrame(columns=["id", "TimeIndex"])
spine_reference = rdt.groupby("id")["TimeIndex"].max().
    ↪ reset_index()

for pid, end_time in tqdm(
    zip(spine_reference["id"], spine_reference["TimeIndex"]),
    total=spine_reference.shape[0],
):

    small_df = pd.DataFrame(columns=["id", "TimeIndex"])
    small_df["TimeIndex"] = pd.timedelta_range(start=0, end=
        ↪ end_time, freq="20ms")
    small_df["id"] = pid

    new_spine = new_spine.append(small_df)
```

```
# merge with existing spine
rdt = new_spine.merge(rdt, on=["id", "TimeIndex"], how="left")
rdt = rdt.drop_duplicates(subset=["id", "TimeIndex"])
rdt = rdt.reset_index(drop=True)

rdt["Timestamp"] = rdt["TimeIndex"].apply(lambda x: round(x.
    ↪ total_seconds() * 1000))
rdt["Timestamp"] = rdt["Timestamp"].astype(int)

logger.info("Fill missing values")
### Fill Missing Values

# Forward fill constant values
const_cols = [
    "Eye_movement",
    "Rec_name",
    "Interpolated",
    "Rec_dur",
    "GazeMoveCount",
    "Task",
    "TaskCount",
]

for col in const_cols:

    rdt[col] = rdt.groupby("id")[col].ffill()

# rdt_old = rdt.copy()
sizes = rdt.groupby("id").count()["Timestamp"]
sizes = sizes[sizes > 2500]

rdt = rdt[rdt.id.isin(sizes.index)]

# Interpolate varying values
logger.info("Interpolating internal points")
```

```
import scipy.interpolate as interp

new_rdt = pd.DataFrame(columns=[*list(rdt.columns), "NewGazeX",
    ↪ "NewGazeY"])

for pilot_id, pilot_df in tqdm(rdt.groupby("id")):

    timestamp_orig = pilot_df["Timestamp"]

    new_df = pilot_df.copy()
    new_df = new_df.dropna()

    timestamp = new_df["Timestamp"]
    gaze_x = new_df["GazeX"]
    gaze_y = new_df["GazeY"]

    new_x = interp.InterpolatedUnivariateSpline(timestamp,
        ↪ gaze_x, k=1)
    new_y = interp.InterpolatedUnivariateSpline(timestamp,
        ↪ gaze_y, k=1)

    pilot_df["NewGazeX"] = new_x(timestamp_orig)
    pilot_df["NewGazeY"] = new_y(timestamp_orig)

    new_rdt = new_rdt.append(pilot_df)

logger.info("Tyding Results")
# flag new interpolations
new_rdt["ResampleInterpolated"] = 0
new_rdt.loc[new_rdt["Automap_score"].isna(), "
    ↪ ResampleInterpolated"] = 1

# set variables as final
new_rdt["GazeX"] = new_rdt["NewGazeX"]
new_rdt["GazeY"] = new_rdt["NewGazeY"]
```

```
# fill new automap scores
new_rdt["Automap_score"] = new_rdt["Automap_score"].fillna(0)

# drop aux columns
aux_cols = ["NewGazeX", "NewGazeY", "TimeIndex"]
new_rdt = new_rdt.drop(columns=aux_cols)

# reset index
new_rdt = new_rdt.reset_index(drop=True)

# reorder columns
new_rdt = new_rdt[
    [
        "id",
        "Timestamp",
        "Eye_movement",
        "GazeMoveCount",
        "Automap_score",
        "GazeX",
        "GazeY",
        "Interpolated",
        "ResampleInterpolated",
        "Rec_name",
        "Rec_dur",
        "Task",
        "TaskCount",
    ]
]

# new_rdt = new_rdt.reset_index(drop=True)

new_rdt.columns = [
    "id",
    "Timestamp",
    "GazeMoveType",
    "GazeMoveCount",
    "AutomapScore",
    "GazeX",
    "GazeY",
    "Interpolated",
    "ResampleInterpolated",
    "RecName",
    "RecDur",
    "Task",
    "TaskCount",
]

e_spine = new_rdt[["id", "Timestamp"]].copy()
e_gaze_metadata = new_rdt[
    [
        *e_spine.columns,
        "GazeMoveType",
        "GazeMoveCount",
        "Interpolated",
        "ResampleInterpolated",
        "AutomapScore",
    ]
]
e_rec_metadata = new_rdt[[*e_spine.columns, "RecDur", "RecName"
    ↪ ]]
gcor = new_rdt[[*e_spine.columns, "GazeX", "GazeY"]].copy()

# Save e_task and e_task_with_checks
e_task_with_checks = new_rdt[[*e_spine.columns, "Task", "
    ↪ TaskCount"]]
e_task = e_task_with_checks.replace("CHECK", np.nan).fillna(
    ↪ method="ffill")

# making new task count
e_task["pid"] = e_task["id"].apply(lambda x: x.split("_p")[0])
task_numbering = (
    e_task[["pid", "Task", "TaskCount"]].drop_duplicates().
```

```
            ↪ reset_index(drop=True)
    )
    task_numbering["NewCount"] = task_numbering.groupby(["pid"])["
        ↪ Task"].cumcount()
    e_task = e_task.merge(task_numbering, on=["pid", "Task", "
        ↪ TaskCount"])
    e_task["TaskCount"] = e_task["NewCount"]

    # drop aux cols
    e_task = e_task.drop(columns=["NewCount", "pid"])

    return e_spine, e_gaze_metadata, e_rec_metadata, e_task,
        ↪ e_task_with_checks, gcor
```

Supporting Files

Listing F.13: Autoregressor Class

```
import sys
sys.path.append("../../")
import src.auxiliary as aux
import pandas as pd
from tqdm.notebook import tqdm
from loguru import logger

# ISSUE: verify name - should include forward/back indication?
# ISSUE: does not work Lags
# EXPAND: include aggregation per Task

def make_rolling_operation(x, win_size_r, win_lag_r, operation,
    ↪ min_periods):

    if operation.split("_")[0]=="quantile":
        quant_num = int(operation.split("_")[1])/100
        result = getattr(x.shift(win_lag_r).rolling(window=
            ↪ win_size_r, min_periods=min_periods),"quantile")(
            ↪ quant_num)
    else:
        result = getattr(x.shift(win_lag_r).rolling(window=
            ↪ win_size_r, min_periods=min_periods),operation)()

    return result

def make_expanding_operation(x, operation, min_periods):

    if min_periods==None:
        min_periods=1

    if operation.split("_")[0]=="quantile":
        quant_num = int(operation.split("_")[1])/100
        result = getattr(x.expanding(min_periods=min_periods),"
            ↪ quantile")(quant_num)

    else:
        result = getattr(x.expanding(min_periods=min_periods),
            ↪ operation)()

    return result

def make_autoregressive(df,
                        grouping_col: str,
                        columns: list,
                        operations: list,
                        window_sizes_seconds: list,
                        window_lags_seconds: list,
                        task_window:bool,
                        min_periods:int=None):

    # Uniform Time Hypothesis
    seconds_per_row = aux.get_configs()["seconds_per_row"]

    df_results = pd.DataFrame()
    # calculate autoregressions
```

```
n_rolling = len(window_lags_seconds)*len(window_sizes_seconds)*
    ↪ len(operations)*len(columns)
n_expanding = int(task_window)*len(operations)
n_combinations = n_rolling + n_expanding

logger.info("You are making " + str(n_combinations) + "
    ↪ combinations!")

with tqdm(total=n_combinations) as pbar:
    for col in columns:
        for win_size_s in window_sizes_seconds:
            for win_lag_s in window_lags_seconds:
                for operation in operations:

                    pbar.update()
                    # pbar.set_description("roll" + str(win_size_s
                        ↪ ) + "s_lag" + str(win_lag_s) + "s_" +
                        ↪ str(operation))
                    stored_col = str(col) + "_roll" + str(
                        ↪ win_size_s) + "s_lag" + str(win_lag_s)
                        ↪ + "s_" + str(operation)

                    # rolling windows
                    win_size_r = int(win_size_s/seconds_per_row)
                    win_lag_r = int(win_lag_s/seconds_per_row)

                    df[stored_col] = df.groupby(grouping_col)[col
                        ↪ ].apply(make_rolling_operation,
                                                                win_size_r
                                                                    ↪ ,
                                                                    ↪
                                                                win_lag_r
                                                                    ↪ ,
                                                                    ↪
                                                                operation
                                                                    ↪ ,
                                                                ↪
                                                                min_periods
                                                                ↪ )
                                                                ↪

    # expanding windows operations in task
    if task_window:

        if "TaskCount" not in df.columns:
            logger.info("Please add TaskCount column!")

        else:
            for operation in operations:
                pbar.update()

                stored_col = str(col) + "_task_expanding_" + str(
                    ↪ operation)
                df[stored_col] = df.groupby([grouping_col, "
                    ↪ TaskCount"])[col].apply(
                    ↪ make_expanding_operation,
                                                                      ope

                                                                      min

df = df.reset_index(drop=True)
return df
```

Listing F.14: Auxiliar Functions

```
import pandas as pd
from IPython.display import display
from loguru import logger
import yaml
```

```
def validate(df):
    display(df.describe())
    display(df.info())
    e_spine = pd.read_parquet("../../data/features/e_spine.parquet")
    assert df.shape[0] == e_spine.shape[0], "Feature and spine size
        ↪ mismatch"
    print("Feature and spine size match")
    for column in df.columns:
        print("\nColumn: ", column)
        aux = df[column].count() / df.shape[0]
        print("Percentage of valid entries: ", aux)
        assert aux > 0.8, "Too much NaN entries"
    return ()


def check_save_feature(
    e_df, file_path: str = "../../data/features/new_feature.parquet"
):

    # display(e_df.describe())
    # display(e_df.info())

    e_spine = pd.read_parquet("../../data/features/e_spine.parquet")
    # check spine columns
    t1 = set(e_spine.columns).issubset(e_df.columns)
    assert t1, "Spine columns missing"

    # check spine size
    t2 = e_df.shape[0] == e_spine.shape[0]
    assert t2, "Feature with wrong length"

    # check index
    t3 = set(e_df.index) == set(e_df.index)
    assert t3, "Index mismatch with Spine"

    # check spine items
    t4 = False not in list(e_df["Timestamp"] == e_spine["Timestamp"]
       ↪ ])
    t5 = False not in list(e_df["id"] == e_spine["id"])
    assert t4, "Timestamp values mismatch with Spine"
    assert t5, "id values mismatch with Spine"

    # check NaNs
    t6 = e_df.isnull().values.sum() / e_df.size < 0.05
    assert t6, "BATMAN"

    # save
    if t1 & t2 & t3 & t4 & t5 & t6:
        logger.info("Passed all checks. Saving feature at: " +
           ↪ file_path)
        e_df.to_parquet(file_path)

    return None


def get_units():

    # Read YAML file
    with open("../../data/infos/units.yaml", "r") as stream:
        unit_dict = yaml.safe_load(stream)

    return unit_dict


def get_configs():

    # Read YAML file
    with open("../../data/infos/configs.yaml", "r") as stream:
        config_dict = yaml.safe_load(stream)

    return config_dict
```

Listing F.15: Configuration File

```
# Encoding of instruments in panel
ins2code_dict : {
    "ADI": 1,
    "ALT": 2,
    "BA": 3,
    "HDG": 4,
    "PWR": 5,
    "SPEED": 6,
    "VS": 7,
}
```