POLITECNICO DI TORINO

Master Degree in Mechatronic Engineering

Politecnico di Torino

Master Degree Thesis

Safety Assessment of UAV Systems: Field Data Analysis

Supervisors

Prof. Giorgio GUGLIERI

Prof. Henrique MADEIRA

Prof. Rui Alexandre Matos ARAÚJO

Candidate

Maria Lucia FERRAMOSCA

December 2021

Summary

Unmanned Ariel Vehicles (UAVs) have gained significant importance in diverse sectors, mainly military uses. Recently, we can see a growth in acceptance of drones in civilian spaces as well. Thus, a profound safety risk analysis/assessment to prevent any possible damage to themselves, the environment, and humans is fundamental for building and utilizing UAVs. To achieve that, two fundamental challenges should be addressed: i) identification of types and frequency of the issues and ii) measurement of their impact. In this thesis, we aim to address the first challenge by automatizing the process of data field analysis. To do so, we first performed some statistical analysis of the reported issues of UAV systems (in Github) and manually extracted detailed data from the reports to better understand the type and nature of the issues. Then, to automatize the analysis, we used machine learning algorithms, first, to extract the *keywords* from the reports and then to build classifier models to classify the reports according to the fault category and severity level. The manually analyzed data is used to validate the results obtained from the machine learning algorithms. We used four classifiers, including Naive Bayes, Decision Tree, Support Vector Machine, and k-Nearest Neighbor, among which Decision Tree turns out to be giving more accurate results in this context. The results provided high accuracy suggesting that these analyses can be performed to further understand UAV system issues, and similar techniques can be performed to analyze other systems as well. In the context of UAV systems, we observed that most reported issues were related to defects and updates in Autopilot source code which tend to have a low impact. However, some catastrophic issues were also identified that are mostly related to position and altitude estimator.

Keywords

Software faults classification, Machine Learning, Github Reported Issues, UAVs Safety, Autopilots, PX4

Acknowledgements

Vorrei ringraziare il mio relatore al Politecnico di Torino, prof. Giorgio Guglieri, che si è sempre reso disponibile e ha sempre accolto positivamente le mie proposte. Inoltre vorrei ringraziare i co-relatori all' Università di Coimbra, prof. Henrique Madeira e prof. Nahgmeh Inaki, che mi hanno seguito passo dopo passo in questo percorso e mi hanno introdotto nell'ambiente di ricerca del progetto BUBBLES dove ho avuto la possibilità di conoscere studenti internazionali con esperienze di vita e di studio molto diverse dalle mie e che mi hanno aiutato a crescere come persona e come studente.

Table of Contents

List of Figures

Chapter 1

Introduction

This chapter constitutes a general introduction of the thesis report by presenting an overview of the context of this study, the problem statement, the contributions and the thesis outline.

1.1 Context of the study

Unmanned aerial vehicles (UAVs) are aircrafts without a pilot on board, which are part of a more extensive system, the unmanned aircraft system (UAS). The UAS consists of the UAV itself, sensors, ground control station, and support services. Drones can be controlled by a ground station pilot or being autonomous, having a payload that does not require human surveillance [1]. UAVs, in particular drones, represents a highly promising emergent industry [2] because of their wide range of uses and the application in multiple sectors, such as deliveries, disaster management, rescue operations, geographic mapping, safety inspections, crop monitoring, hurricanes, and tornadoes monitoring, law enforcement and border control or visual inspection of structures [3] [4]. This is mainly due to their flying ability, simplicity in terms of mechanical design, relatively low price, and also their potential to perform tasks that are costly to be performed by humans or that threaten human lives [5].

Thanks to the many applications for which these systems can be used and the advancement of technology and, the UAS has grown and spread in recent decades. This popularity has made these technologies increasingly affordable and accessible to the population. The civilian drone market, nowadays, is dominated by Chinese drone manufacturer DJI, that alone had 74% of civilian market share in 2018 [6], and in 2021 it reaches the 76%. DJI is followed by Chinese company Yuneec, US company 3D Robotics and French company Parrot with a significant gap in market share [7]. The Federal Aviation Administration (FAA) registered, in September 2021, 866,102 UAVs, of which 343,126 classified as commercial drones and 519,441 as recreational drones. In general, it is expected that the global UAV

market will reach US$21.47 billion by the end of 2021.

1.2 Problem statement

Although drones have a high potential for a wide range of uses in civilian airspace, and can bring affordable life-changing applications, the huge complexity of the entire system brings a severe threat of failures that may result in damage to the drones and, more important, to the environment (e.g., damage in buildings, cars, trees) and ultimately may lead to human casualties, specially with the use of fully autonomous drones.

Several scientific approaches, methods and techniques were developed in the early 20th century to assess potential risks, predict the occurrence of failures and attempt to minimize the consequences of catastrophic situations. However, over the years there have been numerous accidents related to UAVs in civilian areas. Some were actually recorded as incidents and many of them were quite close to cause damages to people, environment, or to the drone itself. Some of these incidents were caused due to communication issues, some were caused due to sensors and GPS errors, and some of them specially in the military and terrorist fields were caused due to security attacks. Studies at the University of Toledo show that up to 2007, the number of civilian and military incidents reported was negligible compared to recent years [8]. The main reason behind this was the low popularity of drones in the civilian areas.

Here we view several examples of such incidents. The first example is related to the case of a Skyjet Aviation Beech King Air A100 that collided with a UAV on 12 October 2017 when it was approaching Jean Lesage airport near Quebec City, Canada [9]. The plane landed safely despite being hit on the wing. This happend because the drone was flying at five times higher than the maximum altitude at which UAVs can fly in Canada (1500 feets).Following this event, the Minister of Transport has increased the restric, including the ban on the UAVs flying near airports.

On August 10, 2018 near the Teton County Fairgrounds in Driggs, Idaho, United States, a hot air balloon carrying a pilot and two passengers was hit by a drone. The drone fell to the ground and was destroyed while the balloon suffered no significant damage and landed safely without injury to the pilot or passengers [10]. The hobbyist drone operator reportedly lost sight of the balloon in the plane's monitor and was operating near the airport without notifying it to the air traffic control, violating federal aviation regulations. This was the first ever recorded mid-air collision between a UAV and a hot air balloon.

On August 10, 2021, a Canadian Flyers International Inc. Cessna 172, registered as C-GKWL, collided with a drone operated by York Regional Police as it approached Buttonville Municipal Airport. The Cessna landed without incident but suffered severe damage, including a bent airbox, a damaged engine hood and a blow to the propeller [9].

December 2015 in Italy, A UAV filming a slalom race in Madonna di Campiglio almost overwhelmed Marcel Hirscher, Austrian former World Cup alpine ski racer [11]. The

International Ski Federation had consented to the use of the UAV, although it was not allowed to fly directly over the competition field. Camera UAVs have since been banned from federations' World Cup races.

In military field, in December 2011, Iran managed to capture a US RQ-170 drone that flew over Iranian territory. It is suspected that Iranian have falsified the GPS signal used by the UAV to hijack it and land it on an Iranian runway.

In August 2014, Iran again claimed to have shot down and captured an Israeli stealth and evasive radar type drone called the Heron [12] which could be further decoded and used against the United States and its allies.

From these few examples it is possible to understand that drones and UAV systems can be critical as their failure can contribute to accidents that involve human lives.

To prevent these incidents, it is required to, in addition to the definition of laws and regulations, perform a more precise study and analysis of the UAV systems, helping to improve the their safe operations. Regarding the regulations and laws, we are witnessing some progresses in different countries in particular in Europe. In many cases, the regulations are focused on the definition of maximum heights, minimum distance between the UAVs, and restrictions on some private areas. These rules are defined to not allow the UAVs to interfere with air traffic, to enter into prohibited areas, and to avoid close contact with the population.

Despite the importance of laws and regulations, they are not enough to guarantee the safety of UAV systems as these systems may fail due to internal software or hardware issues, commutation issues and security attacks. Thus, it is important to study about the most common issues in UAV systems and their impact, helping to improve the systems by eliminating bugs and vulnerabilities and defining appropriate mitigation strategies. A lot of money invested in development of secure and safe systems has been wasted due to an unclear understanding of the root causes of system failures and security attacks. Thus the field data analysis in the context also helps to spent the resources (human resources, money, and time) more effectively.

1.3 Research objectives

As UAV technologies grow and their cost decreases, drones become interesting subjects to undertake and test various applications. Although autonomous navigation still has some challenges, the improvement that has been made in this area makes it a practical method to prevent a human operator from checking every UAV that flies.

Considering that an increase in the applications and quantity of drones involved will also increase the number of possible failure scenarios, guarantee vehicles' safety is extremely important. All the events that have put human life at risk mentioned before only add to concerns about their possible use in the everyday life. A significant question that could be asked is how to ensure the safety and security of drones.

This thesis therefore, is focused on **analyzing and evaluating the severity of the failures that can occur in drones and classify their origin in order to help software and hardware designers to develop products that guarantee the safety of the global system, environment and human**.

In general, two main approaches can be used to increase these systems' safety (and security): fault/failure avoidance and fault/failure tolerance [13]. Unfortunately, due to growing complexity of computer-driven systems, fault/failure avoidance processes are usually not enough [14]. Thus, to maintain drones safe and secure, it is also necessary to use fault/failure tolerance techniques. In either approach, a deep and precise understanding of the system's architecture, functioning, environment, possible faults/failure scenarios, and impact is required. In other words, an accurate risk analysis and assessment is inevitable for building and utilizing drones safely.

In this thesis, we proposed **an approach towards a quantitative risk assessment focusing on automatic field data analysis of issues that have been reported for the UAV systems using machine learning algorithms.** To do this, field data of issues in Autopilot reported by community on GitHub, is used. It is performed a manual analysis to identify the types, frequency and severity of the issues reported, which then leads to the training and building of machine learning-based classification models that are able to automatically identify the category and severity of reported issues. The good results provided by the analysis suggest that these analyses can be performed to better understand UAV system issues, and that a similar techniques can be performed to analyze further systems. In the context of UAV systems, it was observed that most reported issues were related to defects and updates in Autopilot source code which tend to have a low impact. However, some catastrophic issues were also identified that are mostly related to position and altitude estimator which is the most critical component of the Autopilot.

1.4 Thesis contribution

Although drones have high potential for a wide range of uses in civil airspace and can offer life-changing applications at affordable prices, the complexity of the entire drone system also carries a serious threat of failures that can cause damage to the drones, to the environment (damage to buildings, cars, trees, etc.) and, can lead to human victims.

This is why this thesis aims to automate the field data categorization process, in order to produce a security risk analysis/assessment to prevent any harm to oneself, the environment and people.

The main contributions of this work can be summarized as:

- Building a dataset of issues reported for UAV systems' Autopilot in Github by manually identifying the category and severity of the issues.
- Training and Building Classification models using machine learning algorithms (over

the manually built dataset including keywords, fault category and severity level) which are able to automatically identify the category and severity of the reported issues.

- Validating and Generalizing the approach by testing it over a larger number of reports, which are not included in the previously built dataset.

Furthermore, the work presented in this thesis has been accepted for publication as a paper in the *5th International Conference on System Reliability and Safety*, ICSRS 2021, and will be presented at the end of November in Palermo, Italy. The document, entitled *Classifying Fault Category and Severity of UAV Flight Controllers' Reported Issues*, presents the approach and methodology for automating the classification of PX4-related issues extracted from GitHub with relative results.

1.5 Thesis structure

This section presents a brief discussion on the structure of the thesis. The first chapter represents a general introduction that describes the context of the research, the problem statement and the contributions of the dissertation proposed to solve the problem. The rest of the thesis is organized as shows in the Figure 1.1.

Chapter 2

This chapter gives some background on UAV technology, an overview on the purpose of BUBBLES project and an explanation of the U-space services. Subsequently, different algorithms are presented for training a Classifier in the context for text data. Moreover a related work section is presented where are described all the previous work in the context of the main topic that this thesis deal with (i.e. GitHub issues analysis, UAV failures, machine learning for text classification, and more).

Chapter 3

Chapter 3 addresses the problem of the definition of an automatic classification for reported UAV failures. This section reports the approach underlying the study presented in this document and the various methodologies used. The chapter begins with the statistical description of the dataset extracted from GitHut. In sequence, after the definition of a smaller and significantly more reliable data set, the procedure to follow to manually classify the data set is explained. Then, four machine learning algorithms are trained for classifuing Severity and Category. The best of the four is identified evaluating the performance indices. Finally, the chosen best model is applied to a different dataset to assess the reliability of the results obtained.

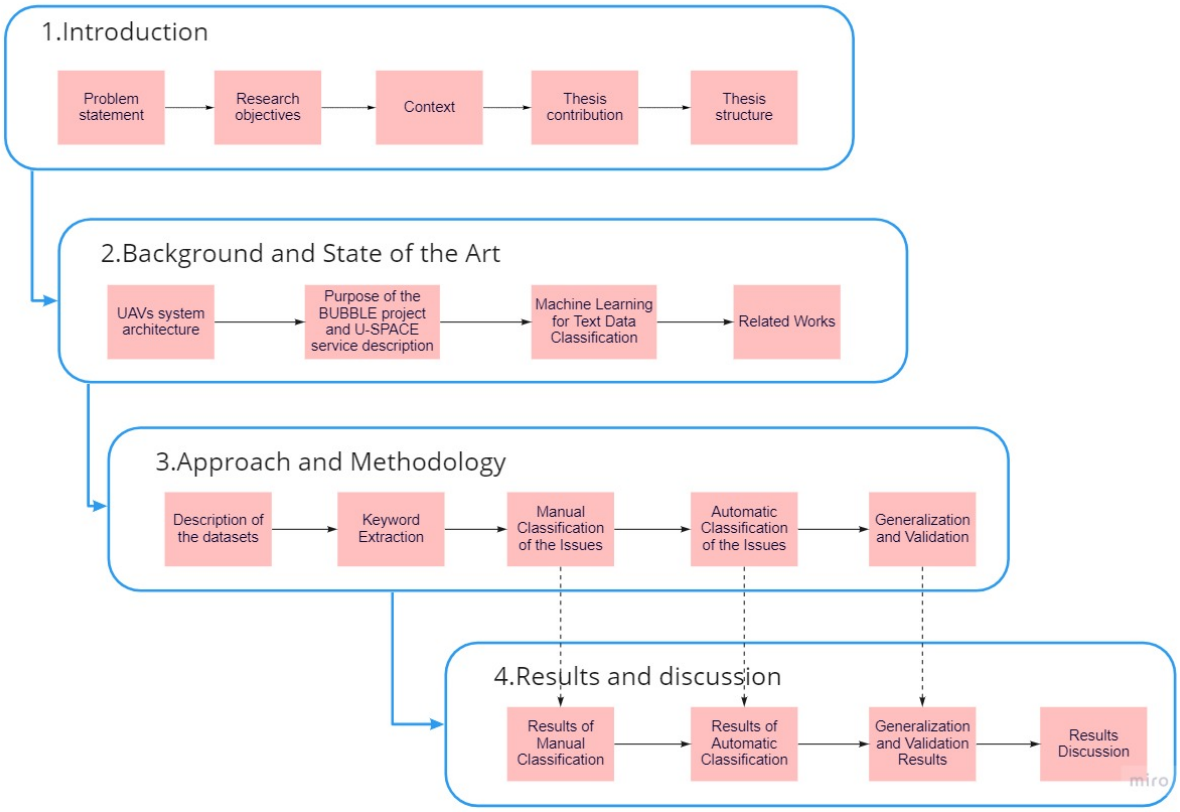

Figure 1.1: Thesis Structure.

Chapter 4

In this chapter all the results provided by each of the step previously defined in the Approach and Methodology part are reported. From the manual and the automatic analysis, to the generalization of the results and the final evaluation of the study.

Chapter 5

Chapter 5 presents a description of some of the most frequent critical failure modes found by experts during manual classification and an in-depth look at automatic analysis, divided by Category. The end of the chapter provides the conclusion of the works. The threat to the validity of the work are presented with the respective possible improvements, and the future works.

Chapter 2

Background and State of the Art

2.1 Context of UAVs

2.1.1 UAV Application

Drone systems have proved to be an emerging sector in industry in recent years [2] thank to their wide range of uses and the application in different sectors, such as deliveries, disaster management, rescue operations, geographic mapping, safety inspections, crop monitoring, hurricanes, and tornadoes monitoring, law enforcement and border control or visual inspection of structures [3] [4]. This is mainly due to their relatively low price, simplicity in terms of mechanical design, quick positioning, safe access to hazardous areas, flexible and scalable deployment to perform many tasks that are costly to be performed by humans or that threaten human lives [5].

UAVs were originally developed during the twentieth century for military use. The missions that they were performing were generally characterized by the three Ds: dull, dirty and dangerous. Dull means long-endurance missions which may become as long as several days due to the desired range and loiter time of the system. Dirty refers to operating in an environment that is contaminated by chemical, biological or nuclear agents. Dangerous missions include suppression of enemy air defense or other missions that require operation in high-risk areas [15].

By the twenty-first they had become essential assets for the military and later, as control technologies improved and costs decreased, their use extended to numerous non-military applications such as in search and rescue, in the agriculture, monitoring of large infrastructures, shipping and deliveries sectors [5].

For example, many manufacturers in Japan started since late 1990's to develop drones for spraying chemicals over the crops, leveraging the ability of the drones of carry more

payloads. Furthermore, with the help of sophisticated sensing technologies that drones can have, it is also possible to minimize the use of chemicals and improve yields [16].

UAVs are also used in many types of meteorological, geological, ecological disasters thanks to the ability to fly over disaster areas and scan the affected area with the aid of cameras and model deployed on the drone itself to identify the exact places where help is required [17]. Some of the examples where drones have been used in case of disasters are the Hurricane Katrina (USA, 2005), the L'Aquila earthquake (Italy, 2009), typhoon Morakot (Taiwan, 2009), the Tohoku earthquake (Japan,2011) and the Emilia earthquake (Italy, 2012) [18].

In addition to the above applications, UAVs can also be used to monitor and track long structures or to map stretches of rivers [19] using visual feedback and GPS to determine the boundaries of the river or structures.

A further civil application involves the use of several UAVs to carry out border or perimeter patrols. The idea is to invest multiple UAVs on the same job so that they can divide up the tasks to be done. For example, one UAV may follow a known target to gather information, while another may provide general border region monitoring. Collaborative UAVs can provide effective, fast and flexible border monitoring compared to other existing systems.

Furthermore, UAVs can be used to inspect and monitor infrastructure such as oil or gas pipelines, roads, bridges and power grids to ensure reliable timely maintenance and thus to extend the life of these civil systems.

2.1.2 UAV Classification

It is hard to achieve a unique classification for UAVs since it differs between countries [20], and it depends on several parameters such as flight altitude, payloads, the weight and size of the drones, flight range, endurance, speed, wings, etc. .

One classification for civil, scientific and military uses, is based upon characteristics, such as size, flight endurance, and capabilities [21] and it categorize them as MAVs (Micro or Miniature Air Vehicles), NAVs (Nano Air Vehicles), VTOL (Vertical Take-Off & Landing), LASE (Low Altitude, Short-Endurance), LALE (Low Altitude, Long Endurance), MALE (Medium Altitude, Long Endurance), and HALE (High Altitude, Long Endurance).

Another classification mainly based on weight, range and endurance, wing loading, maximum altitude, and engine type [22], includes HALE UAVs (High Altitude Long Endurance), MALE UAVs (Medium Altitude Long Endurance), short and medium range UAVs, Mini UAVs and Micro UAVs (MAV).

They can also be distinguished according to their functions in tactical drones (usually are low altitude, short range aircraft), strategic drones (used for long-range reconnaissance over hostile territory) and combat drones (Unmanned Combat Air Vehicle UCAV, can be used for both reconnaissance and attack purposes) [23]. Moreover, the type of gear can also differentiate them in fixed-wing (energy efficient in cruising, better suited for long

endurance missions), rotary wings (energy inefficient, remarkable capability of hovering, and taking-off and landing vertically) and hybrid systems.

The **HALE UAVs** fly with an attitude over 20,000 m with an autonomy of several days. HALEs are considered to be the heaviest UAVs, weighing up to 12,000 kilograms. This type of UAV can fly without being in fleet formation as it alone is sufficient for the type of missions they normally conduct, such as reconnaissance. The Hale are strategic UAVs and can be used for example for in-flight refueling missions, where the Hale plays the role of a tanker. They can be used for reconnaissance and surveillance in military missions.

The **MALE UAVs** fly within an altitude range of 5,000 - 15,000 m with an autonomy of 24 hours. They are similar to HALEs in their applications, but are more used especially in short range missions.

The **tactical drones (TUAV)**, which are used for medium range missions, have a range between 100 and 300 km, and fly at an altitude of less than 5,000 m with a range of ten hours. These vehicles are typically used to support military applications. TUAVs act as a communication network as they are not usually used as part of the training fleet flight, but can work as a cooperative team.

In the end, the **Mini drones (MUAV)** are characterized by an endurance of a few hours, a mass less than 20 kg and a range of up to 30 km. They can be hand-launched and used for different civilian purposes. Differently **Micro UAV (MAV)** are UAVs that have wingspan of 150 mm. They have an endurance of about thirty minutes, a weight less than 500 grams and they can only be launched by hand.

2.2 UAVs system architecture

An unmanned aerial vehicle is defined as an aircraft without a human pilot aboard. UAVs can fly under the remote control of a human operator, such as remote piloting vehicles (RPAs), or with varying degrees of autonomy ranging from autopilot assistance to fully autonomous vehicles without any human intervention [1]. These types of vehicles are characterized by sensors and payload such as a camera, a thermal sensor, etc, which vary according to the type of mission to be carried out. In addition, they are equipped with GPS to determine location information indicating the mission path.

Drones are part of a more extensive system, the unmanned aircraft system (UAS), which is composed of several subsystems [24]:

- A Ground Control Station (GCS) which sends commands to the aircraft.
- An aircraft, UAV, which also transport various types of payloads.
- Communication system that transmits commands and control inputs from the GCS to the aircraft, and sensitive data from the aircraft to the GCS.

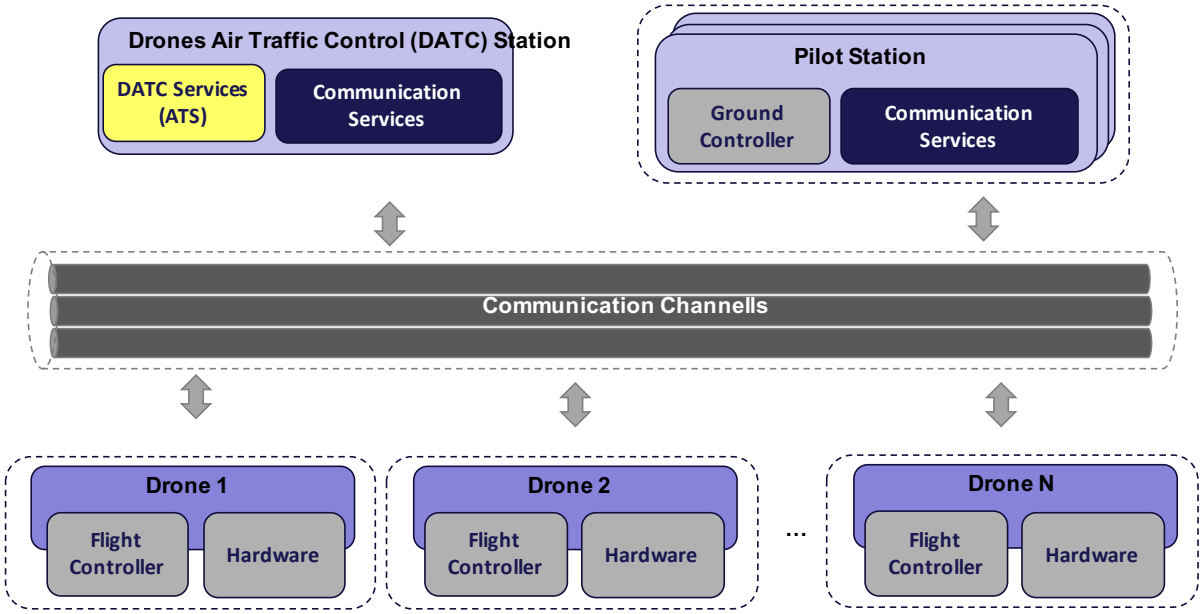

Figure 2.1: General View of Drone Infrastructure.

- Support equipment for maintenance purposes.

All these elements combined enable the flight of the drone without any human intervention and also allow the aircraft to operate in a wide range of missions and emergencies controllable from a base station on the ground, thanks to the ability to communicate with the controller sending payload data, fuel status as well as component status such as engine temperature.

The terminology used to define this system has evolved over the years. The unmanned vehicle was originally known as the remotely piloted aircraft system (RPAS), while today the UAV is used to identify the UAS aircraft. The main difference between the two terms concerns the way in which the system is piloted: the presence of an active autopilot on board characterizes UAV, while an active pilot on the ground characterizes RPAS.

This section presents the main components in the architecture of a UAV, including hardware and software components that are capable of completely or partially replacing human to guide drones in the fly, such as propulsion system, sensors, communication system, flight controller and the open source software platforms.

2.2.1 Propulsion system

To have flight stability and to ensure a more precise flight of a UAV, a good propulsion system is needed, which usually includes motors, electronic speed controllers (ESCs) and propellers. These components are all essential within a drone's propulsion system, as they

ensure the safe flight of the drone.

Electric motor

The electric motor is divided into stator, motor bell housing, windings and bearings.

- The stator of the motor is the fixed part of a rotating system and has its function to generate the rotating magnetic field.
- The motor bell is the part of the drone's motor that rotates, and can rotate clockwise (CW) or counterclockwise (CCW) depending on its configuration. The propellers of the drone are rotating parts too, and they also rotate clockwise (CW) or counterclockwise (CCW), combined with the motors with the respective directions of rotation.

The main characteristics of the engines and propulsion systems are lightness, sturdiness, and efficiency in terms of power referring to the consumption of the batteries (use the lowest possible battery power). Ensuring greater energy efficiency can lead to increased flight time or the use of a heavier payload.

Since drones are designed to be able to fly even outdoors, they must be resistant to all possible weather conditions, so the engine and its components must be weatherproof, and protected from dust, debris and corrosion.

Electronic Speed Control and Propellers

An electronic speed control or ESC is an electronic circuit that controls and adjusts the speed, acceleration and deceleration of drone motors. In many cases ESC also provides motor reversal and dynamic braking. The data is sent to the drone's flight controller from the ground station, which then forwards the received data to the ESC which then sends it back to the drone's engine which is activated, accelerated or decelerated.

The propellers as mentioned above are rotating parts that move following the principle which is based on pushing air in the opposite direction to that of the motion. For example, to move the drone forward, a mass of air is sent behind it. The same happens when the drone is climbing vertically and the propellers push air under the drone. Whichever direction the quadcopter wants to fly, the air must be drawn in and pushed through the propellers in the opposite direction. Propeller blades are designed to have a specific angle to the shaft called the pitch (or pitch) of a propeller. The angle of the propeller blade together with its size and shape define the overall thrust that is imposed on the incident air mass, and therefore also defines the speed of the drone's engine.

Different type of propulsion system for different type of UAV

The choice of the propulsion system varies for each type of drone, as it strongly depends on its size and weight, and also on the mission to be carried out.

For multi-rotor UAVs, such as quadcopters, ESC-controlled Brushless DC (BLDC) motors are usually used to drive the propellers. In the event that the weight is less than 250 g, brushed motors are used, saving the cost of ESCs. Fixed-wing UAVs also use a BLDC for the propeller and servo motors for the flaps. Another typical variant of fixed-wing drones concerns a mixed propulsion, i.e. it uses the propeller as a multi-rotor drone for positioning critical flights, take-off and landing, but flies as a fixed-wing drone once in the air [25].

Electric propulsion is not the only one available, but the most common. There are also engines powered by chemical or nuclear energy on the market, such as the heat engine, the external combustion engine and the internal combustion engine (piston and jet engines) [26]. UAVs weighing 50 kg or more typically use the same propulsion systems for helicopters with two- or four-stroke piston engines. Instead, UAVs in the weight range from 100 to 500 kg are equipped with a rotary piston engine, and UAVs whose weight exceeds 500 kg are equipped with Rotax engines. The heavy fuels that power engines last longer, but their main drawback is that they have a large unit mass. For this reason it is preferable to use electric motors powered by fuel cells, rather than having a large load on board.

2.2.2 Sensors

The control and navigation sensors are responsible for acquiring the data and sending it to the control unit. The main sensors used for the flight of a drone are: gyroscopes, accelerometers, magnetrometer, barometer and GPS receiver. UAVs use sensors to detect all kinds of changes in the surrounding environment, thereby collecting critical data that could hinder flight. Once this information is obtained, the system is able to make appropriate changes to the route by better maneuvering the system. Among the various types of sensors that are used in a UAV to record and collect a variety of information and send it to the ECU, the common ones are:

- Inertial Measurement Unit (IMU): it is the main component of inertial navigation and maneuvering systems in the UAV system. It consists of accelerometers and gyroscopes to accurately estimate the attitude of UAVs, including pitch and roll. It also contains magnetometers to measure yaw and report a drone's specific force, angular velocity and magnetic field surrounding the drone.

- Barometer: it is used to measure the atmospheric pressure and calculate the altitude of the drone.

- Global Navigation Satellite System (GNSS) [27]: measures the position of the drone by calculating the distance from predefined satellites such as Global Positioning System (GPS), GLONASS, Galileo and BeiDuo. Position accuracy depends on the

number of satellites fixed and the Dilution of Precision (DOP). Typically, a low-cost GNSS module has an accuracy of 5 meters.

IMUs and barometers are combined with GNSS to improve the drone's accuracy. The GNSS system is slower than local sensors but is able to provide an absolute position rather than relative as local sensors would.

Additional sensors may be present in the drone system depending on the mission to be carried out but those presented in this section are among those essential to ensure a safe flight.

2.2.3 Communication system

The communication system is a fundamental part of the life of a UAV and allows interaction between the various components of the drone, between the drones themselves and between the drone and the control station. Two different types of communication are used and they are as follows:

- Ground Control Station (GCS): is a software application that is used to communicate wirelessly with the drone. The main functions concern flight monitoring, setting waypoints and sending new commands. To send and receive such data, a telemetry hardware radio unit must be connected to the computing unit to perform the operations. The MAVlink protocol [28] is responsible for the serial connection. QGroundControl, Mission Planner, APM Planner are some examples of GCS applications.

- Radio Control Transmitter (R/C): Used to control the movement, performed by the throttle, and pitch, roll and yaw of the drone. The control commands are mapped into PWM (multichannel) or Pulse Position Modulation (PPM) signals and transmitted to the flight controller, which uses this data to control the drone's motors.

2.2.4 Flight controller

The main component of UAVs and drones is the flight controller, a software system that receives all external data from sensors and GPS and sends commands to the drone to perform a mission. The flight controller is used both in autopilot mode (the drone reads the mission and follows the way-points) and in normal mode (the drone receives remote commands from the control system of the ground station which is controlled by a pilot).

It is fundamental for the processing of UAV system operations, as it controls the engines, reads data from internal and external sensors, implements the attitude estimation and control law (e.g. Kalman filter), and communicates with the control on the ground and also with nearby UAVs. It interacts with other units via previously mentioned communication

systems such as Controller Area Network (CAN), Pulse Width Modulation (PWM) or Universal Asynchronous Receiver/Transmitter (UART), and others.

This section provides a comparison of two of the best known open source flight controllers, called PX4 and ArduPilot. Open-source software (OSS) means that the source code of computer applications may be available through a license. The license gives the copyright holder the rights to study, modify and distribute the software to anyone and for any purpose.

ArduPilot

ArduPilot is an open source autopilot software system. It was created in 2009 as an autopilot for drones and is now also used to control various systems such as conventional airplanes, multirotors and helicopters, boats and even submarines. The software was originally developed for 8-bit ARM-based MCUs to run on their own ArduPilot board which has been replaced by ArduPilot Mega (APM) and has evolved to be optimized for use with 32-bit ARM-based MCUs. ArduPilot has a Ground Control Station (GCS) for mission planning, calibration and vehicle configuration for operating systems such as Windows, Linux and Mac OSX.

It has been installed in over 1 million vehicles around the world, and is used by well-known organizations such as NASA, Intel and Boeing for test and development purposes, as well as being used in universities around the world.

In terms of simulation, it offers both Software in the Loop (SITL) and Hardware in the Loop (HITL). When simulations are run in SITL, the drone, environment and sensor data are based on software running on the user's computer. When simulations are run in HITL, the environment and sensors are still software-based, but the drone's instructions are run on the hardware of a real drone, connected to the user's computer. So the drone itself does not fly, but it has the advantage of testing the program on real hardware.

ArduPilot is compatible with any simulator that uses MAVLink as a protocol to communicate, such as JSBSim. The autopilot receives information about position, speed and more from the sensor from the simulator sends it actions necessary to execute a mission or simply keep the drone model stable in the simulated environment. The ground control station is used to send these commands to the drones, which can be take-off commands, flight to waypoints and mission plans.

PX4

PX4 is another open source autopilot software system. Created in 2009, it was the result of a Micro Air Vehicle competition in Europe. Like ArduPilot, it has matured over the years and now supports multiple vehicles such as rovers, boats, and submarines. PX4

belongs to the DroneCode Foundation[1], which aims to set standards with open source in the drone industry. In addition to the PX4 flight stack and autopilot, both the Ground Control Station (GCS), hardware platforms and simulation are part of the DroneCode project. Many organizations are members of this foundation, including Microsoft.

PX4 works with different vehicle types including multirotor, fixed wing, gliders, helicopters and VTOL technology. It is compatible with the QGroundControl GCS, from which the parameters are set, the sensors read and the mission configured. The use of the license and the high level of completeness make this flight control attractive to commercial airlines.

In terms of drone simulations, PX4 supports, like ArduPilot, SITL and HITL. In addition, it features a particular type of quadcopter-specific simulation called Simulation-In-Hardware (SIH). Unlike the other simulations, in this one everything is running on the hardware and the computer is used only to visualize the virtual vehicle. The autopilot communicates with a simulator via MAVLink, receiving sensor data from the simulator and sending commands to the simulated drone. The recommended ground stations is QGroundControl and it communicates with PX4 takes place via MAVLink, while the recommended simulator is Gazebo. An additional feature, not present in ArduPilot, is external control from an associated computer, usually connected via serial port or Wi-Fi, still using MAVLink as a communication system.

2.3 Purpose of the BUBBLE project and U-SPACE service description

2.3.1 SESAR

The Single European Sky Air Traffic Management Research (SESAR) project started in 2004 as the main technology study of the Single European Sky (SES). Its role is to define, develop and implement what is needed to increase the performance of air traffic management (ATM) and thus build the intelligent air transport system in Europe [29].

The important ambition of this regulation was to integrate and coordinate research and development activities that were previously undertaken in a dispersed and uncoordinated way in the EU. In Europe, both at European and national level, there was no common plan on how to develop and possibly implement promising research results in the field of air traffic. Furthermore, it was even less clear how to prioritize solutions aimed at improving overall performance.

Therefore, with the aim of coordinating all European ATM research towards a common goal, the regulation described three phases of the SESAR project [30]:

[1] `https://www.dronecode.org/`

- Definition phase: carried out by the SESAR Consortium (2005-2008) aims to provide a European ATM MasterPlan (MP), with the aim of defining a research, development and implementation plan for ATM solutions in Europe, and to achieve SES performance objectives. This phase was expected by 2008.

- Development phase: initiated by SESARJU, involves the development and validation of the necessary ATM solutions in a public-private partnership established by this regulation. This phase was originally planned for the period 2008-2013.

- Implementation phase: validated ATM solutions must be produced and implemented on a large scale; this phase was scheduled from 2014 to 2020.

In 2007 the SESAR Joint Undertaking (SESAR JU) was established as a public-private partnership responsible for the modernization of the European Air Traffic Management (ATM) system. The research, development and implementation plan turned out to be too optimistic, and by 2013, only a part of the solutions proposed had been validated and were ready for the transition to the development phase. The SESAR IC combines public and private funds for ATM research and development: the European Commission (EC) and EUROCONTROL are the founding members and provide around two thirds of the research and development budget, while the remaining third is covered by the members of SESAR JU.

The first SESAR JU programme, known as SESAR 1, ran from 2008 to 2016. In that time, SESAR members ran over 400 projects, conducted some 350 validations, 30,000 flight trials and invested 20 million hours to ensure that the results of the programme would meet the operational needs of those who must implement them afterwards. Thanks to this intensive work, the SESAR JU partnership delivered more than 90 industrial prototypes as well as over 60 new or improved operational or technical solutions. The current SESAR JU has 19 members (representing 37 individual companies), as a result of a call for members in 2014 [29].

2.3.2 U-Space services

Drones are a growing market in Europe that provides applications in all environments, including urban areas.

Unfortunately, the failure of the UAS could cause damage to infrastructure or, worse, to humans. For this reason it is considered a critical system and, like any project involving critical systems, it has many associated risks. For example, risk of accidents or risk of security breaches, with attackers using vulnerabilities to perform cyber attacks, jamming, spoofing or terrorism.

Despite the various risks associated with drones crowding our skies, being able to create a true European market would allow drones to exploit their potential to be applied in

various services, and this can help them grow in the economic sector and increase their social acceptance.

U-space is a set of new services that rely on a high level of digitization and automation of specific functions and procedures designed to support safe, efficient and secure access to airspace for a large number of drones. It is designed to facilitate any type of mission, in all classes of airspace and in all types of environments, by addressing an appropriate interface with aviation, manned and air traffic control.

In support of this initiative, in 2017 SESAR drafted the U-space project, which proposes the implementation of four sets of services to support the EU aviation strategy and the regulatory framework on drones[2]:

- **U1**: U-space foundation services covering electronic registration, electronic identification and geofencing. Already available.
- **U2**: U-space initial services for drone operations management, including flight planning, flight approval, tracking and interfacing with conventional air traffic control. With the U1 services, they are the main block of services. Not currently available.
- **U3**: U-space advanced services that support more complex operations in dense areas such as conflict detection assistance and automatic detection and avoidance capabilities.
- **U4**: complete U-space services, which offer very high levels of automation, connectivity and digitization for both the drone and the U-space system.

Figure 2.2 shows the work done, in progress and in the future of the U-Space project. It can be seen that, U1 is in the final start-up phase, while U2 is currently working on various demonstrations, such as flight planning and approval, tracking and procedural interfacing with ATC, and more. Finally, U3 and U4 are still in the research and development phase.

2.3.3 BUBBLES project

In 2017, the SESAR JU launched a series of Horizon 2020 (H2020) exploratory research projects in the context of U-space. The projects have since been completed and other research projects have been launched in 2019. One of the winning proposals of the last open calls was the Building Basic Blocks for a U-Space Separation Management Service (BUBBLES) project.

BUBBLES[3] is a response to the H2020 SESAR JU U-Space program, as part of the exploratory research SESAR 2020 4 (ER4) U-Space (SESAR-ER4-31-2019- U-space). The

[2] `https://www.sesarju.eu/U-space`

[3] `https://bubbles-project.eu/`

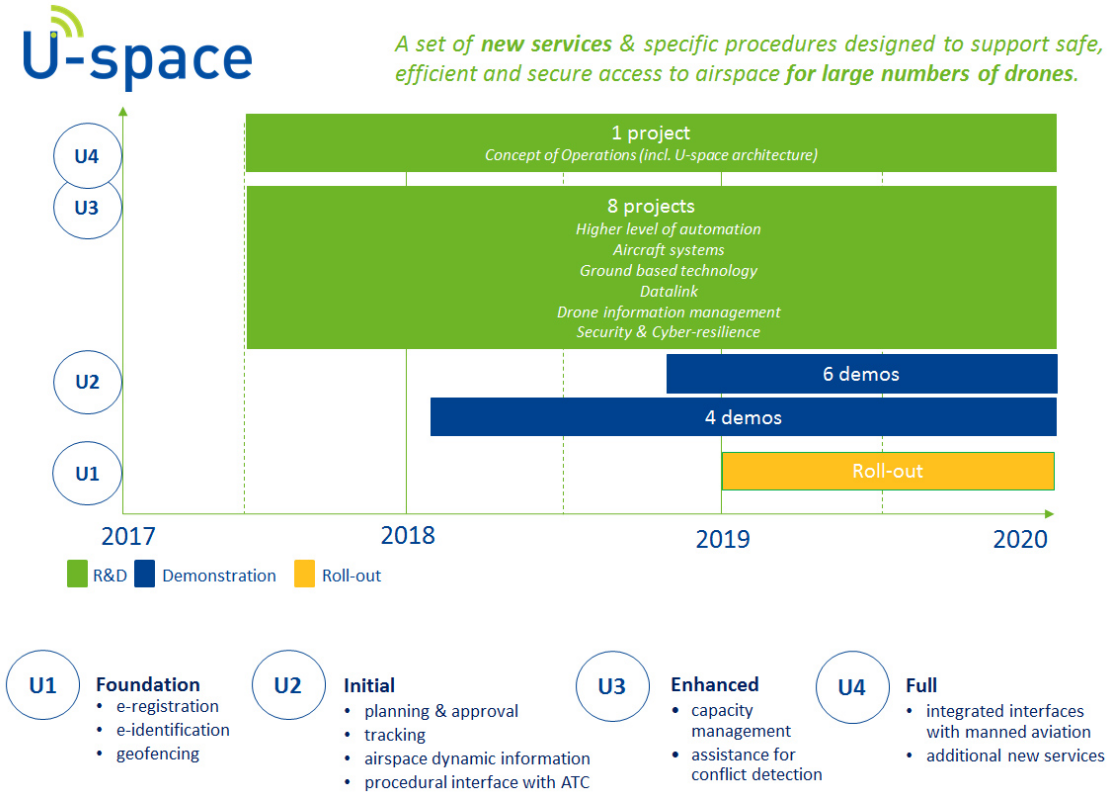

Figure 2.2: U-Space timeline.

EU-funded project brings together five partners from academia, industry and pan-European institutions to contribute to the safe and efficient use of UAS in the European sky.

The development of the technology has led to a low accident rate in manned aviation (less than 3 per million departures), but the contribution of UAS to events where safety is compromised is greater. This difference is due to the fact that air traffic services (ATS) and the systems on which they are based are very mature and safe. On the contrary, U-space services, in particular advanced ones (U3), are still being defined. Therefore, the improvement of the safety and efficiency of UAS operations through the introduction of new advanced U-space services is necessary above all taking into account the high number of UAS flying in the Very Low Level (VLL) foreseen within the Single European Sky - SES (more than 450,000 according to the SESAR Outlook drone).

BUBBLES proposes a step forward in the definition of U-space, as it combines new blocks together with others already defined in order to define a concept for a U-space separation management service that helps to increase safety and efficiency in the VLL (below 150m).

BUBBLES therefore aims to formulate and validate the concept of a new U3 advanced

U-space Separation Management service in order to define and maintain the minimum separation between UAS and between them and aircraft with crew or fixed obstacles. The project seeks to extend the concept of Performance Based Navigation, Communication and Surveillance to UAS, allowing drones to fly closer to high-risk areas, within areas shared with manned aircraft, adopting adequate safety measures. This will demonstrate to authorities how the U-Space Separation Management service concept leads to levels of operational safety comparable to those achieved by manned aviation, contributing to the public acceptance of UAS.

2.4 Machine Learning for Text Data Classification

2.4.1 Text mining

With texts it is meant documents, such as emails, chat conversations, websites and social media, with which we deal every day in the work and private sphere. However, extracting value from this data is difficult and time-consuming. Data Science deals with the extraction and management of insights and knowledge that emerge from recorded data (structured/unstructured) for research and analysis purposes. But extracting useful information from unstructured data such as text is often a huge and complicated task and, very often, a part of the data is often omitted from the analysis. Text mining is therefore important as it helps to retrieve information from unstructured or semi-structured text stored in natural language using advanced analytics and statistical algorithms, in order to elaborate patterns and trends [31].

There are various techniques that can be used in performing text mining, which includes categorization data, entity extraction, information retrieval, clustering, visualization, machine learning, and sentiment analysis.

Since the area of text mining is closely related to that of information retrieval, the ordering of words in a document provides an important semantic meaning. Therefore the representation of words for the extraction of information cannot be based solely on the frequency of the words in the document. There are two main representations of the words in the document that are used in the minimg applications:

- Text as bag-of-words: the set of words in a document is converted into a sparse multidimensional representation. It describes the occurrence of words within a document, and it involves a vocabulary of words and a way to score the presence of the words. Therefore, the number of words defines the dimensions in the representation.

- Text as a set of sequences: the document is treated as a set of smaller independent units, and therefore the individual sentences are extracted as a string or sequences. In this case, the word order is important, although it is only located within sentences or paragraphs.

Text mining traditionally has always used the first type of representation, but in recent years there has been a growing attention towards the second representation, given the growing importance of the application of artificial intelligence to the study of texts. The representation of the text through bag-of-words, in which the words are treated as dimensions with values corresponding to the frequencies of the words, turns out to be isodimensional, sparse and not negative.

2.4.2 Text classification

Text classification is one text mining technique used for categorizing text into organized groups, labels or tags, giving an easy procedure to get information from data and thus automate analysis processes. Some of the most common examples and use cases for automatic text classification include Sentiment Analysis (the process of understanding whether a given text speaks positively or negatively about a particular topic), Topic Detection (the process of identifying the topic or topic of a piece of text) and Language Detection (the procedure for detecting the language of a given text) [31].

The classification is built on the basis of training data that associate the texts with the labels that indicate their belonging to the class. So, for untagged test data, the goal is to associate one of these categories using a supervised model created using the training data.

The reason why the classification is called supervised learning is that the training dataset plays the role of guiding the model towards learning, so that it is then able to assign a particular type of grouping to each new data. This type of guided grouping is used in many applications such as in the case of news portals (they organize incoming documents according to a specific topic such as politics, sports, entertainment and so on, process called news filtering), email filtering and spam, opinion mining, sentiment analysis (the basic idea is to use the text of reviews, blogs or social posts to express opinions on the opinions and feelings of users), and others.

For multiple classification, the set of labels is assumed to be denoted by $L = \{1...k\}$, where the values $1...k$ represent only discrete identifiers with no ordering between them, such as $\{dog, cat, horse\}$. The only case where one could impose an arbitrary ordering between labels, and use them as numerical quantities, is the binary case where the value of k is 2. Many binary classification algorithms use the convention $L = \{0, 1\}$, or they work with the convention $L = \{-1, +1\}$. Binary classification is particularly common and some classification models are naturally designed to solve only the binary case. However, these classifiers can also be used for multiple classification with some algorithmic tricks.

2.4.3 Challenges in text classifiers

Working in machine learning with textual data presents several challenges. Since text are extremely sparse and high-dimensional, standard multidimensional models of classifiers behave in unexpected ways. The frequency of a single term often contains little predictive

power, so only by using it in combination with other characteristics can help building a robust classifier [31].

Another consequence of sparsity is that the presence of a particular term in a document is a more informative feature than its absence, since the presence of a term is statistically rare and therefore more informative. The classifiers that place more focus on absent terms usually have poor performance due to the presence of overfitting. Even the precise frequency of a term contains less incremental information than simply the presence of the term in the document. This asymmetry in the relative importance of terms and their frequencies is sometimes a problem when working with classifiers of the traditional multidimensional domain, which tend to treat all values symmetrically.

2.4.4 Training Validation and Testing

In machine learning classifiers, typically multiple models are evaluated before choosing the final one, which will be used to make predictions on unlabeled data. In this process, models are typically trained on a subset of the original dataset, that is the dataset used by the models to learn from the data. After, the candidate models are evaluate, using the remain part of the original dataset, by comparing them among specific metrics. Finally, once the final model is selected, it is tested the capability of the model of generalization, i.e. how good it performs with new unlabeled data. In order to train the models, perform model selection, and finally evaluate the final model to see if it can generalize well, the original already tagged dataset is split into training and validation sets [32].

The training set is typically the largest set in size, and is used to find the model. In the context of supervised learning, all training data includes the data to be predicted and its outputs (i.e. the label). The validation dataset, on the other hand, still has the data and related outputs, but it is the dataset that will be used to find the performance indices of the model in question, and then compare the metrics of different models in order to choose the model among them. Once the model is defined, the test set (that is the untagged dataset) is used to evaluate the performance of this model and ensure that it can generalize well to new and unseen data. In this case in which a test set is not available, a training and a validation set is just used, which usually has a split ratio of around 75:25.

To evaluate the precision of the classifier the accuracy of the test and of the training is compared. When the accuracy of training significantly exceeds the accuracy of the test, there is a possibility that overfitting (high variance and low bias) has occurred. To prevent it, it is common to perform k-fold cross-validation. K-Fold cross-validation will not reduce overfitting, but using it typically gives a better view of your model, which can ultimately help you avoid or reduce overfitting. The practice of cross-validation involves dividing the data into k blocks (folds) and training k times the model, using a selected fold as the validation set and the rest of the blocks as the training set [33]. At the end, the error on the prediction is evaluated as the average of the errors of each model at each iteration.

Therefore when cross-validation is performed, the model can give good value of accuracy

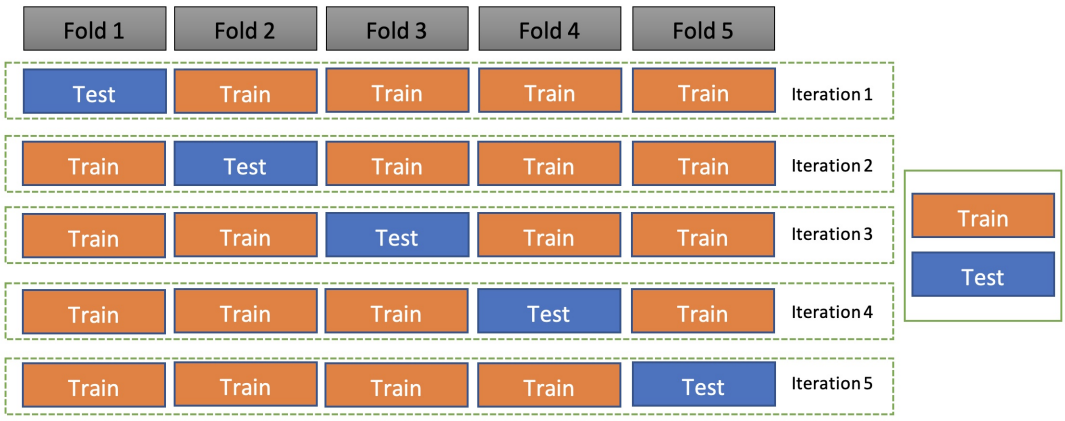

Figure 2.3: Example of k-fold cross-validation with 5 folds.

on some iterations, but it can also perform poorly on others. In general, it is sometimes better to built a model that has slightly worse training accuracy but similar validation accuracy, as this would indicate that the model generalized well to unlabeled instances. Therefore, in the case of text classification with uneven data distribution, when performing cross-validation and being asked to select one of the generated models, it is better not to choose the one with the highest accuracy but to check other metrics such as recall, precision and F1 and choose the one that provides the best generalization capability.

2.4.5 Models

This section discuss four basic models for text classification, which are the Naıve Bayes classifier, Nearest-neighbor classifier, Decision Trees, and Support Vector Machine classifiers. These four classifiers are selected because they are among the oldest methods in the literature, and are among the common used model for text classification.

Naive Bayes

Naive Bayes is a probabilistic classifier based o the Naive Bayes rule:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$$

where D is a set of data and θ a vector of parameter used to build the model. $p(\theta)$ is the a-priori knowledge about θ, before observing the data. $p(D|\theta)$ is the conditional distribution or likelihood that relates a set of observed data D to θ. $p(\theta|D)$ is the posterior distribution, representing the update state of the knowledge about θ, after seen the data. $p(D)$ acts as a normalization constant

This classifier is called naive because it makes some simplifying assumptions about the independence of the attribute value in test instances. The model assumes that the text data is generated from a mixture of different classes, i.e. the labeled data (training and validation set) are the results of a generative process and the purpose of the model is to estimate the parameters of the generative process [31]. Typically, only training data is used to estimate the parameters, the other part of the set is used to validate the reliability of the parameter selected.

Subsequently, these parameters are used to estimate the label to each unlabeled test document based on the probability of generation from each class. This results in a probabilistic classification of unlabeled documents. Naive Bayes classifier has Bernoulli and Multinomial models.

In Bernoulli's model, it is assumed that only the presence or absence of each term in the document is observed. The Bernoulli model assumes that the $j-th$ term is present in a document generated by the $r-th$ class (Cr), with probability $p_j(r)$. Then, the probability $p(Z|Cr)$ of the generation of the document Z from the component Cr is given by the product of the different Bernoulli probabilities corresponding to the presence or absence of other terms:

$$p(Z|Cr) = \prod_{t_j \in Z} p_j(r) \prod_{t_j \notin Z} (1 - p_j(r))$$

According to the Bayes rule of posterior probabilities, the posterior probability of Z being generated by the class component C_r of the $r-th$ class can be estimated from the following formula:

$$p(C_r|Z) = \frac{p(Z|C_r)p(C_r)}{p(Z)}$$

The main task in the training phase of the Bayes classifier is to estimate the values of the a priori probabilities and of the generative probabilities specific to the class. These parameters are estimated so that the observed data has the highest probability of being generated by the model and are then used to predict the labels of unseen test instances.

An assumption made in this algorithm is that the presence or absence of the various terms are conditionally independent of the choice of class. Therefore, the joint probability can be expressed as the product of the corresponding values on the individual attributes. This assumption is the reason why the method is referred to as naive classifier, because this assumption of conditional independence is generally not true in real contexts, but despite this, the actual predictions are surprisingly robust.

While the Bernoulli model uses only the presence or absence of terms in documents, the multinomial model explicitly uses the frequencies of their terms. Just as the parameter $p_j(r)$ in the Bernoulli model denotes the probability that a term is presence in a particular component, the parameter $q_j r$ in the multinomial model denotes the fractional present of the term $j-th$, including its repetitions, for a particular class component r.

To decide when it is preferable to use the Bernoulli model or the multinomial model, two main factors must be considered:

- The typical length of each document.
- The size of the lexicon from which the terms are taken.

For short documents that have a not poor representation compared to a small lexicon, it makes sense to use the Bernoulli model, as in short documents, there is a limited number of repetitions of terms, which is why the information about the frequencies made by multinomial model is not significance in this context. In the case in which the representation of the document is poor, the information about the presence or absence of terms is noisy and the inaccuracy of the Bernoulli model will increase. Therefore, multinomial models is used in this context.

K-Nearest Neighbors

Nearest Neighbor Classifiers are based on the principle that similar instances have similar labels. The basic idea is to identify the k-neighbors closest to the data to be labeled and calculate the number of data belonging to each class. The class with the highest number of instances is reported as the relevant one and therefore to be assigned to the data under examination. Cosine similarity is used to calculate nearest neighbors, although advanced methods such as the substring kernel can be used to incorporate sequence information into the classification process. The nearest neighbor classification can be used for both binary and multiple classes as long as the class with the highest grade is used.

Closest Neighbor Classifiers are also referred to as lazy learners, memory-based learners, and instance-based learners, since these methods store all training examples and use the ones that best match the instance in question. Despite other model-based methods, KNN perform less generalization and learning in advance, but most of the classification work is left to the classification phase.

However, there are many variations of the closer classifiers where some of the learning work is instead brought to the fore. Such classifiers are called closest neighbor adaptive classifiers, which require no training but a greater number of similarity calculations to classify each test instance.

Most of the literature uses the notation of k to denote the number of closest neighbors. This number is a parameter for the algorithm, and its value can be set by trying different values of on the training data. The value for which the maximum accuracy is obtained on the training data, is the optimal value of k.

Decision Tree

As the name suggests, a decision tree is a hierarchical or tree-like partitioning of data obtained with a series of division conditions. During the training phase the data space

is subdivided into regions that are heavily biased towards a particular class. During the test phase, the relevant partition for each test data is identified and the partition label is returned. Once the decision tree is defined, each node corresponds to a region of the data space defined by the split conditions in its predecessor nodes, as shown in Figure 2.4 and the root node corresponds to the entire data space.

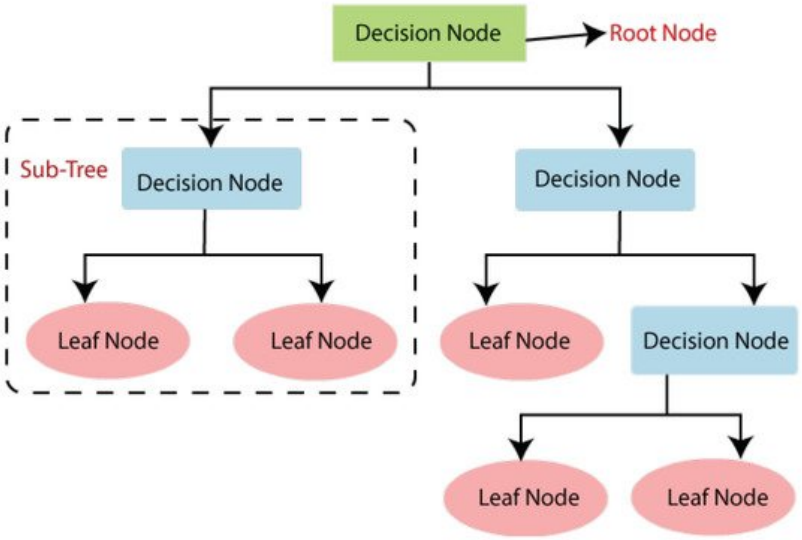

Figure 2.4: Decision Tree classifier graph structure.

As already mentioned, decision trees subdivide the data space recursively using conditions to divide the branches. The basic idea is to choose the division conditions in such a way that in the terminated nodes called leaf nodes (Figure 2.4) the classifier identifies a unique class for the data to be classified. The subdivision criteria typically correspond to constraints on the frequencies of one or more words. A split that uses a single attribute as decision is called a univariate split, while a split that uses multiple attributes as decision is called a multivariate split [31]. An important characteristic of the graph is that during the divisions that are applied recursively in a top-down fashion, each node can have only two children and the division going on until each node in the tree contains a single class.

Trees built on different training samples belonging to the same dataset sometimes have very significant variations, generating little confidence in the classifier's decision-making power, resulting in poor performance on that tree's test data. This problem is solved by trimming the nodes at the lower levels of the tree that do not contribute positively to the generalization power of the model. As a result, the leaf nodes of the pruned tree may no longer contain a single class and then another decision algorithm is applied to define the class to which the data reaches that end of the tree, such as the nearest k-neighbor classification. Pruning is performed by considering each node within the tree, and verifying whether or not the accuracy improves on the data contained by removing the subtree

rooted in that node. If accuracy does not improve, pruning is performed. This procedure is done by selecting the internal nodes from bottom to top, until all have been tested once.

Due to the sparse and highly dimensional nature of the text, the standard implementation of decision trees does not always provide good results. Decision trees have the ability to approximate the boundaries of arbitrary decisions, given an infinite amount of data. However, with a finite amount of data, the predictions of a decision tree are strongly biased in favor of specific classes. This bias is caused by the split criteria at the higher levels of the tree, which have a disproportionately large effect on the final prediction.

Univariate splits are those that cause the creation of unbalanced decision trees, in which paths are decided based on the presence or absence of a term. The paths dominated by the absence of terms are therefore much longer than the paths dominated by the presence of terms. Univariate subdivisions are generally best for classifying short texts or text documents drawn from a smaller lexicon, in case the tree pays more attention to the absence of data. When working with long documents, it is particularly important to use templates with many terms at the decision points of the learning process to focus more attention on the presence of the terms rather than the absence.

Multi-way classification, on the other hand, tends to work poorly in the text, and usually are performed by building single tree for all classes. This is because the terms relating to the various classes are largely disjoint. Therefore, a multi-way classification problem is usually decomposed into multiple one-against-all binary classification problems and the accuracies of the different classifiers are integrated giving a more confident prediction.

Support Vector Machine

The Support Vector Machine classifier is a special case of a linear model for classification. In the linear model the dependent variable y_i, which is a binary value that usually belongs to $L = \{1, +1\}$, and is related by a linear function to the document dependent variable $i-th$, $X_i = (x_{i1}...x_{id})$. X_i are characteristic variables which, in the case of text, correspond to the frequencies of the terms of a lexicon with d terms. Hence, linear models for classification assume that the dependence between y_i and X_i is expressed in terms of d linear coefficients $W = (w_1...w_d)$ and a bias term b such as follows:

$$y_i = sign\{\sum_{j=1}^{d} w_j x_{ij}\} = sign\{WX_i + b\}$$

To construct the model, therefore, we need to find a vector of coefficients W and a bias b, so that the conditions are satisfied. But satisfying the conditions for all training data points is a difficult challenge, so this preliminary modeling hypothesis is considered an approximation to the true function of X_i and y_i. Therefore, since the set of coefficients $w_1...w_d$ and bias b that satisfy the previous equation usually does not exist, it is better to find W and b, so that the equations are satisfied with as few cumulative errors as possible.

Considering numerical dependent variables, the optimization objective function is in the following form:

$$Minimize\ J = \frac{1}{2}\sum_{i=1}^{n}(y_i - WX_i - b)^2$$

In addition to this type of penalty function, there are many other versions in which errors can be penalized, thus varying the properties of the model.

Linear classification has a geometric interpretation of equations in terms of linear hyperplanes. The hyperplane $WX + b = 0$ can be seen as a hyperplane that separates the feature space into two classes, as shown in Figure 2.5. Each datum corresponds to a point in the space of the characteristics that can belong to one of the two hyperspaces defined by the hyperplane. Points are penalized depending on how far they are from the separation hyperplane when they belong to the wrong side. The bias b is proportional to the distance of the hyperplane from the origin, while W defines the inclination of the plane in space.

All forms of supervised learning, as explained earlier, rely on the ability to generalize a model learned from training data by testing its performance on a test data. If the size of the training data is small, the coefficients learned in the training phase may not always be well generalizable. In general, the greater the amount of data available, the better the coefficients that can be learned and the better the ability to predict unknown test data.

As mentioned above, the y_i class label is often considered binary in the case of linear models. When considering multiple classifications, it is common to use meta-algorithms that use a binary classification algorithm as a basis and then make multi-label predictions. There are several strategies for converting ensembles to multi-label classifiers.

- The first approach is defined one-against-all. In this strategy k different binary classification problems are created, so that each class corresponds to a problem. In the i-th problem, the i-th class is considered as positive values while the remaining data are considered negative. At the end of the strategy a total of k models are created. The ensemble is applied to each of these training datasets, creating a total of k models.

- The second approach is called individual. In this strategy, a training dataset is built and 2 pairs of classes are evaluated at a time, and for each model generated there is a vote for the winner. This results in a total of k(k-1)/2 models if k classes are considered. In the end, the class label with the highest number of votes obtained is declared the winner in the order.

A support vector machine (SVM) has a special geometric interpretation of its regularizer, which leads to the notion of separation by margin. The basic idea here is that an SVM creates two parallel hyperplanes symmetrically on each side of the decision boundary, so that most of the points are on either side of these two margin hyperplanes on the correct side.

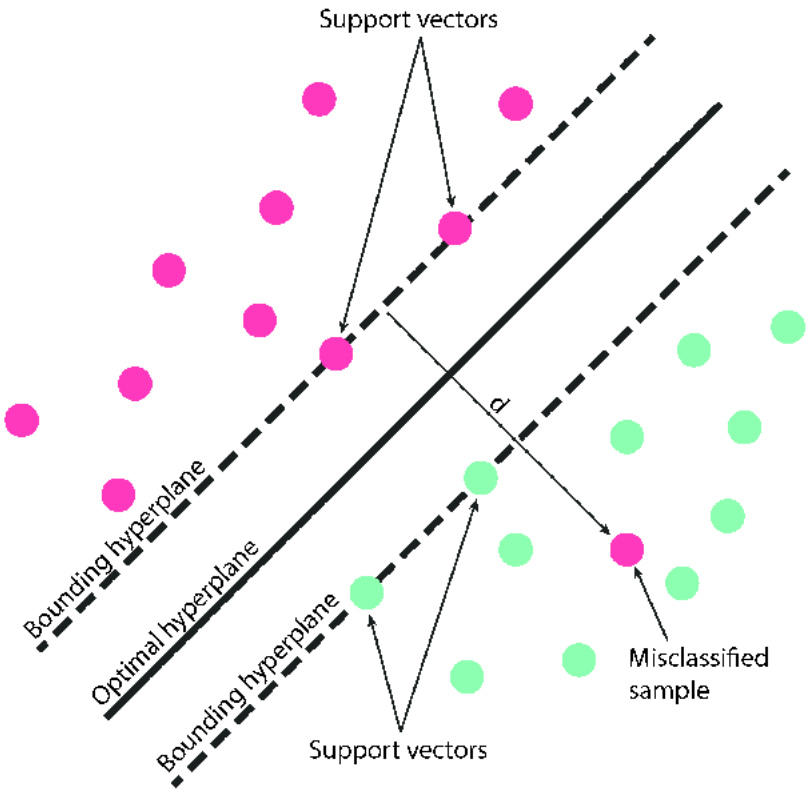

Figure 2.5: Support Vector Machine hyperplane.

SVM generally uses hinge loss rather than quadratic loss within the minimization function:

$$min\ c\sum_{i=1}^{N}[1-y_i(w_0+w^Tx_i)]_+ + \frac{1}{2}||w||_2^2$$

where $[1-y_i(w_0+w^Tx_i)]_+$ is the hinge loss function and $\frac{1}{2}||w||_2^2$ is the regularization term.

A key concept in SVM optimization is the notion of support vectors, from which SVM derives its name. Each of the two separation hyperplanes parallel to the decision boundary could touch one or more training points. Such training data points are referred to as support vectors, that is, points that "support" the hyperplanes on either side of the decision boundary.

There are cases where the data is far from being linearly separable, and therefore it is not possible to define a hyperplane to separate the two classes. To solve this problem, a non-linear feature map is introduced.

$$\phi(x) = (\phi_1(x), ..., \phi_d(x))$$

where $\phi_j(x) : R^n \to R^d, j = 1, ..., d$ are given basis function. The feature map ϕ maps the n-dimensional input space into a d-dimensional feature space. With this mapping, point classes that were not linearly separable in the original input space may become linearly separable in the higher dimensional feature space. The decision boundary in the original input space is now nonlinear, but the problem remains the very same structure as in the linear case, by just replacing the x variables with the Φ, and hence pose the learning problem on the new feature data.

Solving this problem means solving a dual:

$$max_\alpha \quad g(\alpha) = -\frac{1}{2}(\alpha \odot y)K(\alpha \odot y) + 1^T\alpha$$

where $K(x, z) = \phi^T(x)\phi(z)$ is the kernel function.

The advantage of solving the dual equation stay in the fact that this doesn't require knowing the feature map explicitly, but only the kernel matrix $K = \Phi^T\Phi$. Then instead of explicitly know which feature map to use, it is possible to just decide what kernel function to use. The kernal function, cannot be totally arbitrary, but it must satisfy certain properties, it has to be positive semidefinite function. The standard choice for the function are:

- Linear kernel: $K(x, z) = x^T z$
- m-degree polynomial kernel: $K(x, z) = (1 + x^T z)^m$
- Gaussian kernel: $K(x, z) = exp(-\gamma||x - z||_2^2)$
- Sigmoid kernel: $K(x, z) = tanh(k_1 x^T z + k_2)$

This operation of changing the variables allowed to solve the problem in which the problem cannot be linearize without knowing explicitly the feature map but just deciding the kernel function to use.

2.5 Related Works

Currently, many works are being carried out as part of overall UAV systems and making them overall safer, for example, the U-Space services mentioned before which provide online services to support safe and secure missions when there are large numbers of UAVs in the airspace.

Also risk assessment and failure modes and effects analysis are essential analysis for these critical systems and technologies to gain high acceptance. It helps in: i) creating awareness,

ii) identifying who might be at risk, iii) identifying hazards, iv) defining adequate controls and measures to be taken or developed, v) prioritizing hazards, and vi) definition of standards/regulations [34].

This section provides background on UAVs' safety issues and presents related works on safety risk analysis and assessment, analysis of bugs, analysis of Github issues and machine learning for text classification.

2.5.1 Safety of UAVs

To provide better automated services and operations by UAVs, ensuring the safety of the in-flight mission is of the utmost importance. The greater the number of UAVs present in a given application scenario, the greater the risk of failure and the potential collision and damage caused. The smooth integration of UAVs into civilian airspace is based on the demonstration that their mission would present at least a level of safety comparable to that of human-powered aircraft.

The work of Clothier and Walker [35] introduces an empirical analysis on safety objectives based on human-pilot aircraft activity. This analysis considers hazards present in manned and unmanned aircrafts, and historical data of onboard and ground fatality risks. Regarding onboard fatality risks, the work states that, within the analyzed data, 56% of midair collisions by human-piloted aircrafts were fatal, but only 1% resulted in fatalities on the ground. They present a theory that states that although UAVs could exceed the midair collision accident rate by a significant margin, the number of fatalities would still be within an acceptable safety standard due to the lack of an onboard pilot. Regarding ground fatality, the article deduces that it is difficult to set an objective comparison between manned and unmanned vehicles. This is mostly due to the existence of UAVs that are physically incapable of causing fatalities due to their size (e.g., micro air vehicles). However, fatality as a metric does not fully reflect the occurrence of a hazard but only one of its consequences. Similarly safety concerns are shown in the survay of Altawy and Youssef [36] and some possible improvement are discussed in more recent work [37].

2.5.2 UAVs Safety Risk Analysis

Several works can be found regarding safety in the context of UAVs. The work presented by Wackwitz and Boedecker, [38] proposes a four-phase model for a UAS safety risk assessment. These four phases are: i) Safety Hazard Identification; ii) Safety Risk Assessment; iii) Safety Risk Mitigation; and Safety Documentation. They present three methodologies for Hazard Identification: i) Reactive, which involves analysis of past events; ii) Proactive, which involves analysis of existing or real-time situations during drone operation; and iii) Predictive, which involves monitoring and data gathering to predict possible future hazards. Regarding risk assessment, the risks are categorized based on their probability and level of severity. Finally, the risk mitigation methodologies can be categorized in: i)

Corrective actions with an immediate reaction to fix the causes of the safety hazards; and ii) Preventive actions that have a long-term effect on the safety hazard and will, eventually, mitigate the risk to an acceptable level.

In a similar line, a work on small UAVs [39] is focused on the development of two preliminary risk analysis approaches. These two approaches - a Standard Safety Risk Assessment and a Probabilistic Model-Based Risk Assessment - also work at qualitative and quantitative levels. The hazards are subjected to a qualitative risk analysis based on operational complexity, population density, vehicle weight and configuration.

A combination of qualitative and quantitative risk analysis of drones over the internet, is presented in a recent study about drones' safety [40]. The methodology followed is based on two studies. One is a qualitative analysis which is based on a combination of ISO 12100 (i.e., focused on risk analysis and assessment of any machine) and ISO 13849 (i.e., internationally registered standard for performance level of machines and concerns all levels of manufacturing) standards. In the second study, a quantitative analysis is done by applying a Bayesian network-based technique over data collected from previous studies with the objective of analyzing the relationships between the risk of drone crashes and their causes.

Although there are risk assessment and safety analysis models that are becoming widely used, like SORA [41] and MEDUSA [42], there is still a lack of an accepted model throughout the research community in this area. This is mainly due to the difficulty in providing a general model, given the high number of different scenarios a UAV can go through and the risks that can originate from such scenarios.

Two different methods for risk assessment are presented in [43], which are mainly based on the damages caused by the drones' failure. A new method is presented in [44] which derives from two already existing approaches for analysis of drones security threats during conceptualization phases. Other work from Hartmann and Steup [45] demonstrates a scheme for risk assessment based on provided services and communication infrastructures. They aim to evaluate how the risks associated with environment, communication links, sensors, data storage and fault handling mechanisms affect drones in terms of integrity, confidentiality and availability.

These methods are analytical and need realistic parameters to produce reliable results. The analysis of field data, such as the one presented in this paper, is essential to help practitioners and researchers in the difficult task of estimating the parameters required by different analytical modeling techniques.

2.5.3 Failure Modes and Effects Analysis

Failure Mode and Effects Analysis or FMEA is an analysis aimed at the systematic identification of possible Root Causes (i.e. the causes of the error, which are the mechanisms that lead to the occurrence of the error) and Failure Modes, in order to then estimate the related risks to limit or avoid the internal risk of the system in question. Failure Mode

and Effects Analysis is therefore a systematic method of analyzing and classifying the risks associated with various products or processes. It begins by assigning priorities for corrective actions, then acts on the items ranked with the highest score, re-evaluating those items and then returning to the prioritization phase, in a continuous cycle until marginal returns are established [46].

The FMEA procedure for the classification and therefore the identification of the risk is based on the assignment of a numerical value called the risk priority number (RPN), which uses Severity, Event and Detection as metrics. RPN is obtained by simply multiplying the severity of faults (S), the portability of events (O) and the possibility of detection (D) [47].

$$RPN = S \cdot O \cdot D$$

Severity refers to the possible effect of a system failure. The severity is greater the more serious the consequence of the fault is. The portability of event refers to the frequency with which the event is likely to occur in qualitative terms. Detection refers to the likelihood of detecting a root cause before an error occurs. The factors of severity, occurrence and detection are usually assessed individually using a numerical scale between 1 and 10, although these scales may vary depending on the applied FMEA standard. In general, it is true for all standards that a high value represents a low score.

The way of proceeding in the analysis foresees that the system to be studied must be decomposed into its sets. Once all the sub-sets have been identified, for each of them all possible failure modes and root cause must be determined, and each failure mode is assigned a severity level and each root cause must be assigned a level of occurrence and detection. FMEA is a procedure that requires training and study by a cross-functional team of experts from various departments. The team analyzes each component and subsystem of the product for failure modes and then determines the potential causes and effects. The analysis requires a deep understanding of the engineering system, which is usually composed of subsystems which are themselves made up of smaller subsystems. Each phase of the analysis begins with known failure modes at a lower level and describes the effect at the higher level. Collecting these failure effects is then used as a failure mode for a device in a larger system [48].

However, while FMEA is a great analysis tool, it has several challenges. As you perform the analysis in smaller subsystems, it becomes increasingly difficult to find information on component risks. Furthermore, implementing a highly manual FMEA system is a difficult task and is sometimes not easy to use and difficult to understand. As a result, many companies only use FMEA to meet the contractual requirements of their customers [49]. Building FMEA is a tedious and time-consuming task, especially when FMEA is used in complex systems with multiple functions. Further analysis showed that a criticality of the FMEA is the inability to influence the design, as the analysis execution time often exceeds the design/development phase, and therefore making changes in later phases will result more expensive [50].

The various limitations of this approach seem to have hitherto limited the adoption of an automated version of this FMEA technology which is still under development. In the work of Papadopoulos, Parker and Grante [51], the authors propose a new generic approach (i.e. potentially applicable to different engineering models) that allows the automation of FMEA, based on the automation of another taste analysis system which is the fault tree. In this approach, the authors construct the FMEAs from engineering diagrams that have been augmented with component failure information. The first step in analyzing such models is to determine the local failure behavior of components in the system and then use the model structure to automatically determine how local failures propagate causing functional failures to the system. In the final step of the process, the fault propagation logic results in a simple table of direct relationships between component and system failures. As in the case of FMEA, the generated table determines for each component of the system and for each failure mode of that component, the effect on the system, such as the main events of the failure trees.

FMEA as mentioned above is used by various industries, including automotive, aeronautics, military, nuclear and electrical engineering [52], therefore specific standards have been developed regarding the assessment of severity, occurrence and detection for each of its applications. Although data rating scales may differ, the processes for definition remain similar.

Some applications of the FMEA are found in the work [53], in which the authors used FMEA in the failure analysis of geothermal turbines. Another application of the FMEA is on Geothermal Power Plants (GPPS) in the work of Arabian-Hoseynabadi, Oraee and Tavner [46], in which all the possible consequences of the failure modes studied to reduce a great variety of possible modes are evaluated failure. Morover many company used the FMEA for research in different areas of application, for example, FLAME used it for the electrical design of automotive systems aim to provide safety on airbag system [54], and also GENMech [55] use it for the mechanical design.

In the context of UAVs, FMEA is also widely used. For example in Michael B. Revill [56] this risk analysis is used to understand the behavior of the UAV swarm and reveal all potential failure scenarios that could impact the desired mission. Through manual inspection or semi-automatic control of drone flight, unwanted behavior and failure modes can be discovered allowing mission planners to prevent these events from occurring or ensure that the drone responds to failures with the necessary safety behaviors.

In the article by Wu, Yang and Cao [57] instead, hardware and functional analysis are adopted to produce an FMEA concerning EMA which is a system that helps the UAV or guided missile to control the flight attitude. Undoubtedly, EMA plays a significant role in ensuring flight safety, therefore an FMEA is an important tool in design, as it allows defining essential safety requirements and highlights issues that require more attention.

2.5.4 Analysis on the bug

A software bug is an error, defect or failure in a computer program or system that produces an incorrect or unexpected result or causes unplanned behavior [58]. A bug report, on the other hand, is a software document describing software bugs submitted by a developer, tester, or end user [59]. Since programmers can hardly write programs without bugs, it is inevitable to find bugs in software development. Also, finding and fixing bugs in software development is expensive and time-consuming. In fact, software testing and debugging has been estimated to consume more than a third of the total development cost of software [60]. Therefore, bug prediction is an important topic in the software engineering cycle of empirical studies, becoming an important measure for software managers and end users of software [61].

Bug repositories are often used in open source software projects to allow both developers and users to post problems encountered with the software, suggest possible improvements, and comment on existing bug reports. Especially in the case of large projects, bugs are easy to find, so it becomes very important to have an appropriate bug tracking tool to make communication between teams more effective and also allow to have all bugs and changes stored in the system web based. Users of the bug tracking system are typically customers, software testers, project team members, management members, and software developers, all of whom provide all users with the ability to monitor and manage multiple projects simultaneously [62].

Therefore a typical scenario in the evolution of the software involves the analysis of a bug report through a bug tracking system. In the work of Torchiano and Ricca [63], the authors dwell on some information that can be extracted from the software repositories such as the source code comments and the version control log. This information is used by a tool to provide the developer with a shortlist of files (suggestion list) that may be affected by the proposed change request.

In the context of smartphone platforms and tools, in parallel with the growth in the distribution of smartphone applications (apps), errors relating to the apps themselves are also growing. Therefore, in the article by Bhattacharya, Ulanova, Neamtiu and Koduru [64], an in-depth study of bugs in Google's Android smartphone platform and 24 widely used open source Android apps is performed. First, several metrics are defined to understand the quality of bug reports and analyze the bug fix process: bug fix time, bug categories, bug priority, status and community engagement. Second, differences in bug life cycles are identified and how they can affect the bug fix process. Finally, a study is conducted on Android security bugs as they are devices that carry security sensitive information.

Unlike other studies that use static code metrics or modify metrics as inputs to predict remaining bugs in a software system [65], the goal of Wu, Zhang, Yang and Wang's work [61] is to monitor and predict the increase in the number of bugs per month in a large software system using historical bug data. To achieve this, the authors interpret the number of bugs per month given as a time series and they used time series analysis

algorithms such as ARIMA and advanced ARIMA X12 to predict the number of bugs.

When dealing with large open source software projects, continuous software development processes are carried out with a regular release cycle, during which new bugs are reported, evaluated and fixed. A well-known open source software is Eclipse which in the second half of 2018 went from an annual to a quarterly release cycle, becoming a relevant object of study to investigate how this change affected Eclipse's recent bug management activity. Work done by Abou Khalil, Constantinou, Mens, Duchien and Quinton [66] presents an analysis of Eclipse's bug handling process over a 15-year period, considering 138K bug reports from Bugzilla, including 16 versions Eclipse annuals and two quarterly releases in 2018. In the end, the study shows that, from bug fixes, fix rate, bug detection time, and other data, it is possible to see how Eclipse's bug handling activity has improved over time, with a sharp decrease in the number of reported bugs and an increase in the bug fix rate.

2.5.5 Git Issues Analysis

Github is a widely accepted version control system. It is used by both the industry and the research community, and the number of works and contributions from Github users, researchers, and developers has increased significantly in recent years.

Many researchers are working on analyzing Git issues with different means and methodologies and pursuing various goals. For example, a study on the relationship between user churn and software issues [67] analyzes user churn using Git problems, while in Caglayan and Bener work [68] the authors analyze the behavior towards ownership of issues, its resolution, and the issues patterns by evaluating the issues of two large projects. A similar analysis of user behavior was published in the bitcoin longitudinal analysis [69], where the authors study the comments and commenters of Git issues. In another work [70], the authors present a structured survey to understand the usage of Git issues and their usefulness.

Some contributions are also providing tools and methodologies to manage Git issues, for example, Dhasade, Venigalla and Chimalakonda [71] proposes an approach to prioritize Git issues to better understand and resolve the reports from users. In another study [72], the authors present a web application to manage Git issues reported by the software managers and developers to better understand the trends and reports.

In Fan, Yu, Yin, T. Wang, and H. Wang work [73], the authors aim to understand if text mining on reported Git issues could provide relevant information to understand potential issues and weaknesses of a software system. The results, based on a large dataset, suggest that text mining could be very beneficial to understand and analyze Git issues. In another analysis [74], partially similar to our initial statistical analysis of the dataset, the authors use the Git issues of the bitcoin system. Their results show how their study can be helpful to understand how Git issues can help highlight the trends and identify the major issues and faults in a system.

2.5.6 Machine Learning in Natural Language text classification

Text classification is a supervised learning technique. It helps speed up the storage and retrieval of information, which would otherwise take a lot of time and effort from manual classification, which can also cause inaccurate categorization due to people's subjectivity, especially in large datasets.

There are several text classification applications such as spam filtering, news classification by topics in online newspapers, knowledge management and support for internet search tools, and more [75].

Especially in the medical field, many studies demonstrate that it is possible the to automatically diagnose a certain disease from medical records or clinic notes. For example, in the work published by BMC Medical Informatics and Decision Making [76] the detection of the medical subdomain of a clinical case, such as cardiology, gastroenterology and neurology, is foreseen by medical reports written mainly in natural language. To accurately classify a note's medical subdomain, the authors built a machine learning-based natural language processing (NLP) pipeline and developed medical subdomain classifiers based on the note's content. In the article, clinical notes are considered a powerful resource for solving various clinical problems as they provide detailed patient conditions, which usually cannot be obtained from the other kind of text data as the electronic health record (EHR) system.

In another work [77] the automatic classification of ischemic stroke is studied. In ischemic stroke (IS), subtype classification is critical for management and prediction of outcomes. The study uses the natural language processing of electronic health records (EHR) which will be the input dataset for trining the classifiers in machine learning. A similar approach was followed in the work of Caccamisia, Jørgensenc, Dalianisb and Rosenlunda [78] which aim to define smoking status of patients from the electronic medical record (EMR).

Concerning the methodology, the work proposed by Dien, Loc and Thai-Nghe [79] follows the fundamental steps in the definition of an automatic classifier for text data. The authors propose an approach for the classification of articles by testing different machine learning algorithms (SVM, Naive Bayes and KNN), vectorizing the data before classification. The texts were pre-processed, and later segmented into words in order to generate TF-IDF term matrix. TF is used to estimate the frequency of occurrence of a term in a given document, to evaluate the weight of the word in the text. The resulting matrix becomes the input of the ranking models.

Differently, in [80] a Semantic Text Similarity approach is followed. In this study, aviation accidents are classified into different Human Factors (HF) class. The Semantic approach is based on comparing a new event with already tagged data, in terms of semantic meaning: if the events are semantically similar enough to the tagged data, there is a high probability that the new data have the same HF. The steps followed in this study are: i) Select a dataset ii) Preprocess the text data iii) Build and train the classifier model

iv) Use the model to represent phrases and tokens from the report and v) calculate the semantic similarity between new phrases and old tagged phrases and obtain the HF with the highest value.

Chapter 3

Approach and Methodology

Considering all the advances on the UAV system and looking at the growth of IT, the arrival of a new world seems to be very close, and for that to happen, people need to work together to produce better software [81]. Version control systems are the best ways to achieve desired collaboration and transparency, and one of the most accepted systems available online is GitHub [82] [83] [84]. It offers many user benefits such as version control, branch workflow, distributed development, community management and many more.

The main advantage of Github is that it has its own online problem reporting service, where developers, software quality assurance (SQA) experts, and users report problems and bugs they are facing or have identified, which can then be fixed via discussion among users through comments under the description of the reported problem.

Since autonomous drones are controlled by the ground station running an autopilot, it is necessary to understand the overall failures that can occur in the pilot system. Understanding means identifying and analyzing the various types of failures, particularly those occurring in the autopilot software since software reliability is of relevant importance in the analysis and the overall efficiency [85].

One of the most used autopilots in the sector that is been selected for this analysis is PX4[1]. This autopilot has the advantage of being open source and being available on Github, which makes it more accessible for analyzing and understanding faults and failures that may occur in the system.

Github was therefore the starting point from which the idea for this study was born: a vast dataset was selected from the platform that reports a wide variety of types of failures ranging from code and update errors, to communication failures, to errors in the estimation by the sensors, which occurred during the real or simulated flight of autonomous drones via the PX4 autopilot.

The initial step was to fully know all the characteristics of the starting dataset. Therefore

[1]https://px4.io/

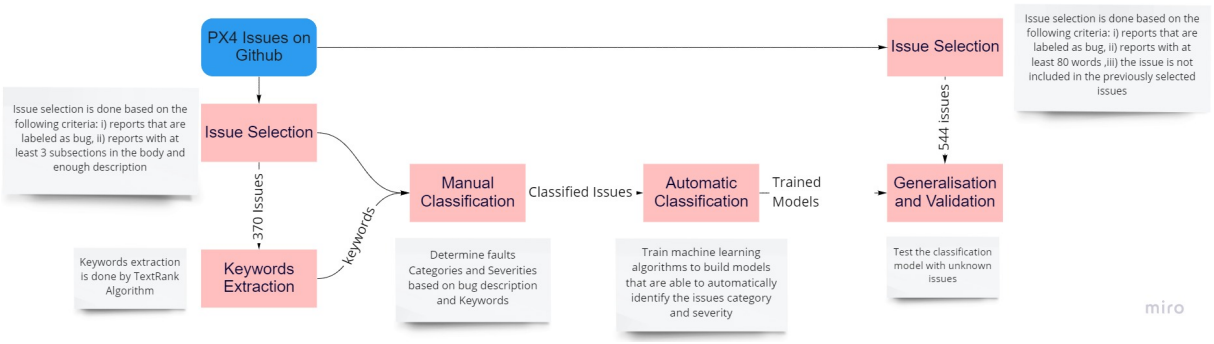

Figure 3.1: Approach and the Methodology to Implement the Approach

a series of statistical and qualitative analyzes were carried out in order to extract as much information as possible and become confident with the preliminary material. It was searched the trend of the number of creating issues reported on the Github platform, the categories of issues most frequent per years and the time required for them to be resolved, and many other information.

Once the potential of the dataset and the possible connections that can unite the issues it presents have been fully understood, a working approach has been planned.

Figure 3.1 presents the general approach and the process followed in this study for analysis of the bug reports of PX4 flight controller. It reports the following phases:

1. **Data Selection**: Definition of a more meaningful dataset among the 17k data collected by GitHub using an inclusion criterion.

2. **Keywords Extraction**: Extraction of keywords from the collected reports using TextRank algorithm[2] implemented in Matlab text Analytics toolbox.

3. **Manual Classification**: Manually identification of Category and Severity of the issues previously selected, using the details of the issues and the keywords during the classification. This manual analysis is done by a group of 5 expert researchers.

4. **Automatic Classification**: Definition of 4 machine learning algorithms to build classification models to automatically identify the Category and Severity of the reported issues. This is done over the dataset built by the manual classification.

5. **Generalization and Validation**: Identification of the best performing classifier from the previous phase and verification the performance applying it to a set of unknown reports selected (by applying new criteria) from the rest of the issues in

[2] `https://it.mathworks.com/help/textanalytics/ref/textrankkeywords.html`

Github (all issues minus the issues used in the previous phase). The classifiers and the results are validated using cross validation, blocking and manual verification.

In this study we used MATLAB[3] as a tool to implement the aforementioned approach. Matlab has been selected for the many advantages it offers such as [86]:

- Huge number of techniques and algorithms in machine learning environment.
- Collecting and managing of data in tables and different types of variables.
- Powerful visual representation of data that can be extracted.

3.1 Description of the datasets

The download of the issues dataset was done on March 18, 2021, and a total of 17,162 issues were collected from Github, considering only issues reported for PX4[4].

While analyzing the issues reported by the GitHub community, several categories were identified. Some examples of these categories are catastrophic bugs such as *"... the platform restarts mid-air which causes the vehicle to crash ..."*, hardware faults *"... while the system was heating up but the sensor reported temperature was constant at 43.18 degC ..."*, logs issues *"... PX4 lists no logs when using QGC to request a log list ..."*, messages or communication issues *"... During flight, the ekf2 module stops publishing all messages, causing a flight termination. It is remarkable that before it's stops publishing, it publishes one more time all messages at the exact same moment and with exact same timestamp ..."* and many more.

Therefore, the datasets based on the problems reported by the natural language user community are highly heterogeneous: it varies from very vague problem descriptions to very detailed ones. This feature of the dataset, together with the large amount of data, makes the task of extracting meaningful statistical information highly challenging [87].

Although Github has the problem reporting service, it does not allow to export problems directly, but to analyze these datasets it is important to export them in parsable formats. For this reason, another open source platform called Github csv tools[5] was used, which is based on Nodejs and it allows to export the reported problems of a repository in a csv file with the following relevant details:

In Table 3.2 two types of problems extracted from Github with the Github cvs tool described above are presented, both examples are reported without the Body section as a more detailed description of this part will be discussed in the next chapters. The first

[3] `https://www.mathworks.com/products/matlab.html`

[4] `https://github.com/PX4/PX4-Autopilot`

[5] `https://github.com/gavinr/github-csv-tools`

ID Title State Labels Created at Closed at Body

Table 3.1: Relevant detail tags extracted for each GitHub issue.

problem is a solved (closed) type of problem, in fact both the creation date and the closing date are reported. The second problem, being still open, is incomplete in the closing date and therefore an empty box is left in the table.

ID	Title	State	Labels	Created at	Closed at
17139	*gyro_fft:...*	closed	bug	2021-03-16T00:17:43	2021-03-16T04:16:15
17007	*RTL_FLT_TIME...*	open	bug	2021-03-02T21:45:19	

Table 3.2: Examples of Issues extracted from GitHub without the Body details.

For the purposes of this statistical study of the collected data, a generalized tool was created on MATLAB in order to be used by researchers or professionals in the sector to perform analyzes on data sets similar to those extracts from the Github issue system.

3.1.1 Statistical analysis: graphs and tables

The analysis proposed in this section starts from a qualitative evaluation of the datasets obtained from GitHub, and then continue with quantitative analyzes referred to the bugs and to the most frequent labels, as well as an analysis of the expected times for the closure of each issue.

The first evaluation is aimed at identifying the most frequent words present in the title and in the description (Body) of each issue. For this purpose, the Word Clouds created with MATLAB are a useful result for obtaining an intuitive view of the most common types of issues reported. Figure 3.2 displays the most common bugs on the set of PX4 data issues. The greatest reported are *firmware* issues, but there are also *gps*, *control*, *sensor* and *mode* issues among the most popular. *Mavlink* issues are also reported on the cloud as one of the most common types of PX4-related problem. This means that it is recurrent to find communication issues within those related to the software.

Investigating the bugs in detail we found that 90% of them were closed (as shown in Figure 3.3) in a time interval ranging from 2012 to 2021. Figure 3.4 illustrates the trend of issues created and closed by years. The growing number of them detected over the years testifies to the high interest in the problem related to the failure on drones piloted by PX4, and also highlights the growing ability to resolve these problems, as the tendency of closed problems grows parallel to the open ones.

Having noticed how much the number of issues created each year has increased, the interest of the study moved on the types of issues most frequent per year. In an attempt

Figure 3.2: Word Cloud of PX4 reported issues

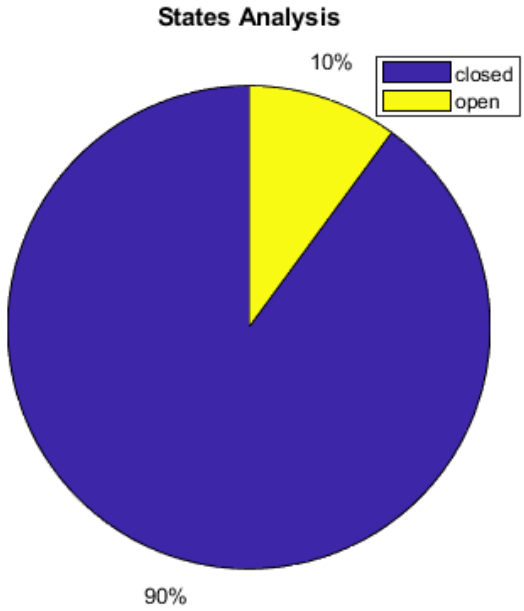

Figure 3.3: State of PX4 reported issues

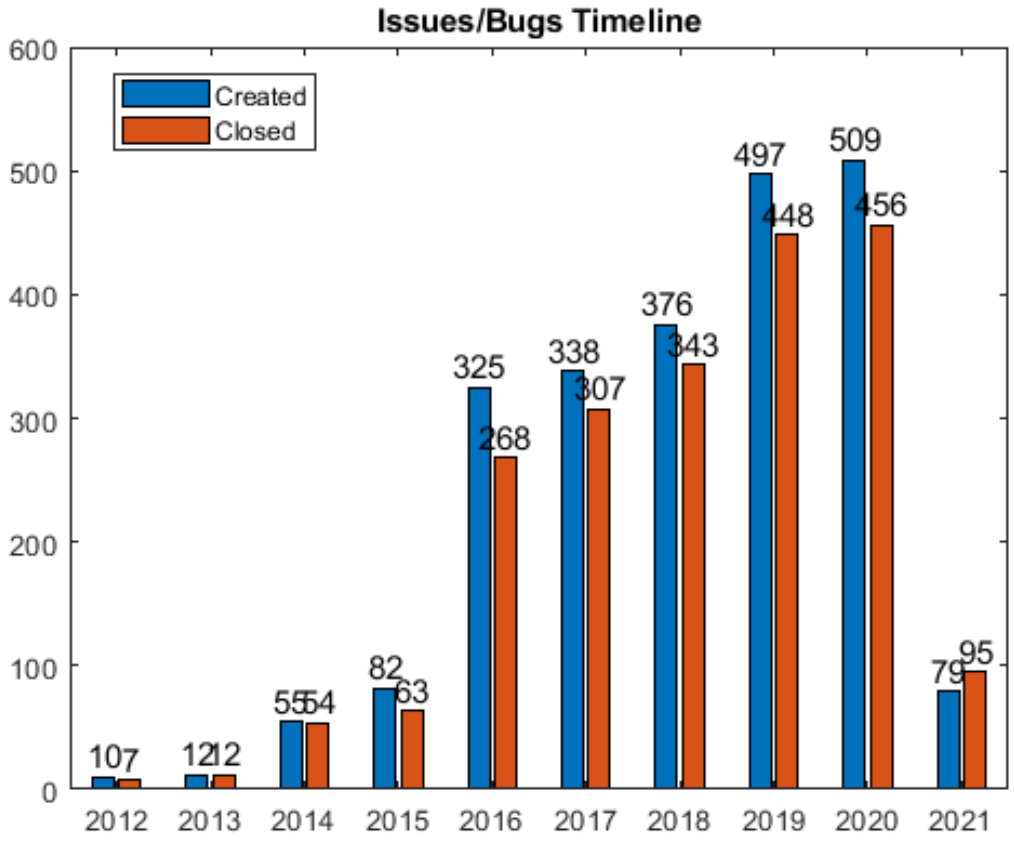

Figure 3.4: Timeline of PX4 reported issues

to understand how the search for errors and resolutions has changed over time, the Figure 3.5 extracted a timeline, which represents the top 5 most frequent labels per year. Between 2012 and 2013, that are the years in which the lowest number of issues is recorded, the labels present are only two and concern *bug* and *enhancement*. After this first period, *bug* always remain an imported part of the total but new types of tags appear, such as *VTOL*, *fixed wing*, *multirotor*, *stale* and *cleanup*. These types of problems concern some specific types of drones and their applications in flight. They would appear to be related to the PX4 flight mode, which is the ways the autopilot responds to remote control input and how it handles vehicle movement during autonomous flight. *Cleanup* usually refers to code cleaning that involves upgrading new versions or deleting old ones or improving some modules. This is justified by the fact that after an initial phase of detecting and fixing bugs, the second phase is more focused on finding new types of problems that are no longer related to software bugs but to the way in which the software works.

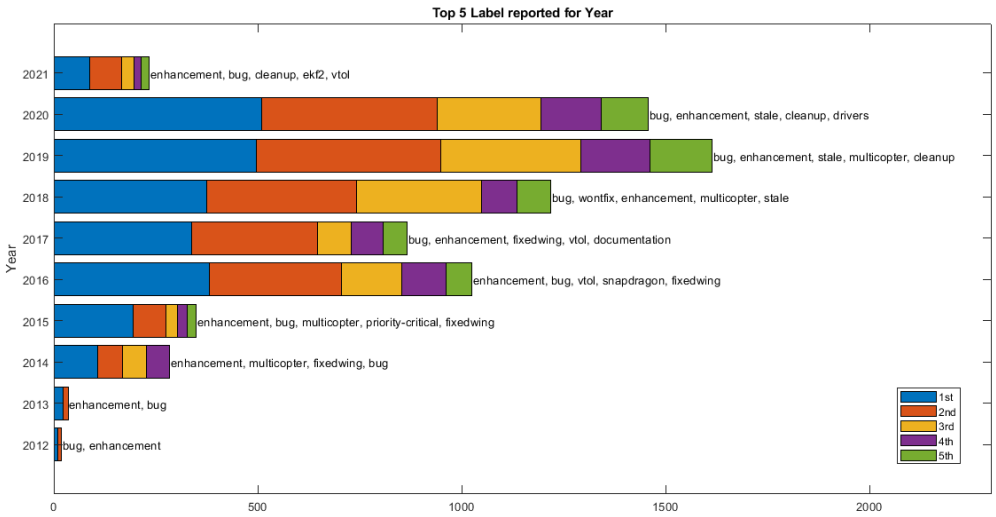

Figure 3.5: Analysis of the 5 top label for year of the Issues reported for PX4

Another graph illustrated in Figure 3.6 shows the top 10 of the labels reported on Git for PX4 issues and the one in Figure 3.7 shows the same analysis but for issues that are closed. Comparing the two diagrams we can see that the percentages are almost equal and it is not surprising to note that in the first and second position of the top 10 there are *bug* and *enhancement* issues respectively, as the previous analyzes already revealed that they are the most common over every year.

Understanding the average resolution times of the issues we came to identify as shown in Figure 3.8, how many days are needed to close an issue per label. The ones that take more time to update is *VTOL* (Vertical Take Off and Landing) label with 42 days while a *bug* takes 34 days to fix, which is still a long period of time even if less then the one before. A VTOL aircraft can fly as either a multicopter or as fixed-wing vehicle. Probably the main problem with this is in switching between modes, which happens automatically from the PX4 when it is set in Auto mode.

Further analysis on the times is made to compare the closing times in days between all the issues and just the bugs. Figure 3.9 displays the two distributions: the mean values are not far between the two curves, but the standard deviation of the total issues is much higher than just bugs one. The bug bell shows well-concentrated values around the mean value.

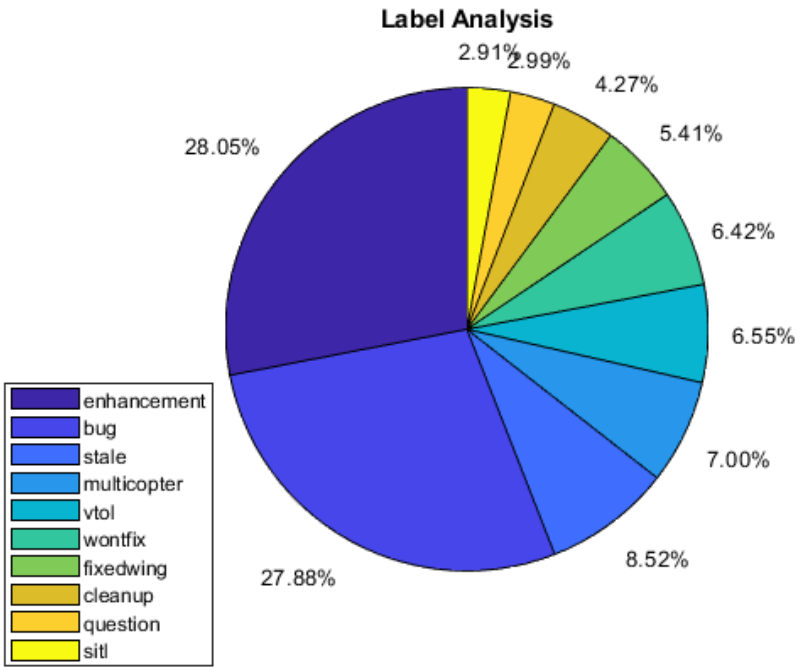

Figure 3.6: Label Analysis of the Issues reported for PX4

3.2 Definition and Analysis of a smaller and more relevant subset

Given the nature of our dataset, which is quite uneven, ranging from very vague to very detailed issues, it is important to define some inclusion criteria to select adequate data from more than 17K issues for our analysis.

First of all, from the entire dataset only the issues labeled as bug have been selected, as we aim to analyse the faults and failures in UAV PX4-piloted systems. As the second inclusion criterion, the most detailed reports have been selected, in order to have enough information for analysis. To this ends, reading numerous descriptions of the issues it was observed that reported issues with subsections (e.g., *Describe the bug*, *To Reproduce*, *Expected behavior*, *Possible Fix*) are more detailed than others.

Thus, the chosen inclusion criteria are:

- Reports that are labeled as bug.
- Reports with at least 3 subsections in the body.

14,879 reports are eliminated by the first criterion and from the remaining reports 1673 number of reports are eliminated by the second criterion, which resulted in giving us 610

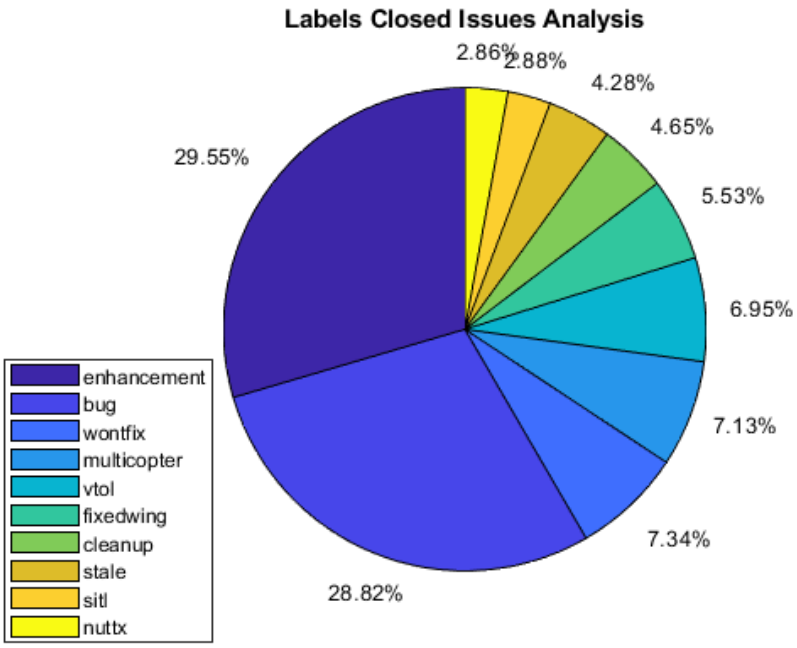

Figure 3.7: Label Analysis of the Closed Issues reported for PX4

issues to work with.

Out of these bugs, some were related to improvements, developers call and pull request, and some had a very short description or only links in subsections. This is the starting point from which manual analysis begins and the automation of this begins.

3.3 Keyword Extraction

Since the issues reported by the community are written in natural language and consequently many parts of the text are not relevant to the category and severity of the issues, we have chosen to use keywords as input to build more effective classifiers. The keywords are words that characterize the topic treated in the text, allowing to understand the context and the main information from the description of the problem, facilitating manual and automatic classification.

The TextRank [88] algorithm proposed by MATLAB is used to extract keywords. We used this algorithm due to its popularity and known good performance, also confirmed on the text dataset used. It is a graph-based algorithm aimed at classifying the graph nodes by deciding the importance of these nodes, in relation to the global information generated during the creation of the graph itself. The way it is defined what is important and what

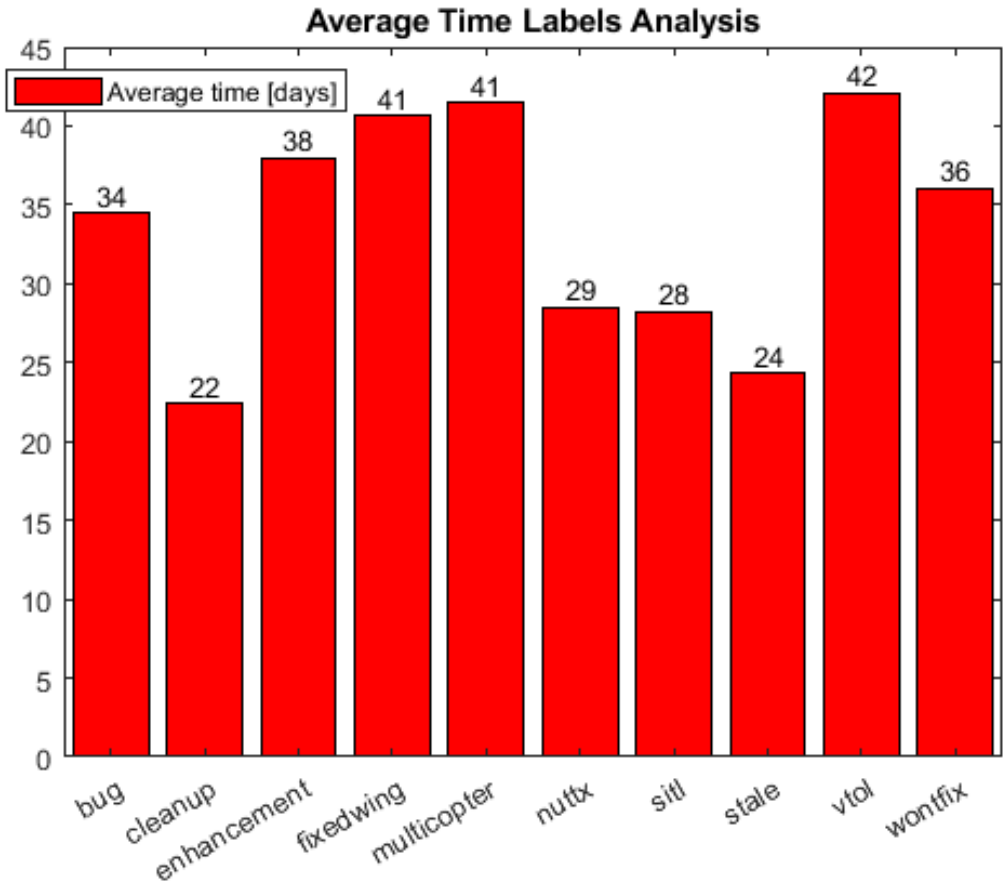

Figure 3.8: Average time for Label required for closing an Issue reported for PX4

is not, is called vote or recommendation. Ultimately the scores associated with each node are based on the votes that are cast for it.

The expected result at the end are the words representative of the natural language text assigned to the algorithm to be processed.

Figure 3.10 shows an outline of the steps followed for the extraction of keywords, starting from the raw text up to the choice of keywords. The main steps are:

1. Pre-Processing: preparation of the text, by text-cleaning.
2. Tokenization: separation of each sentence of the text into words.
3. Definition of the graph: identification of the text units that are candidate keywords (nodes) and identification of the connection between them (edges).
4. Ranking: iterative application of graph-based ranking algorithms until convergence and definition of the nodes with the highest final score.

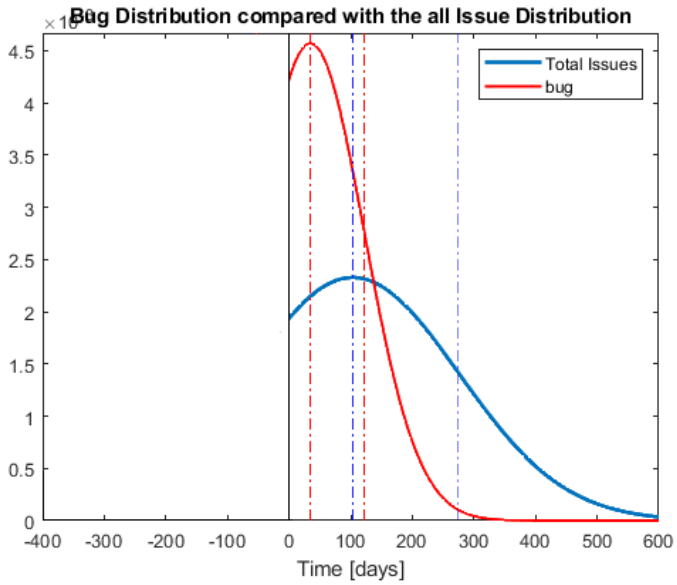

Figure 3.9: Normal distribution of the time (in days) needed to the total issues and of the bug issues reported for PX4 in order to be closed

The algorithm does not store the extracted data after processing, but after cleaning the memory of the keywords previously evaluated as candidates, it repeats all the steps for each new text submitted to it.

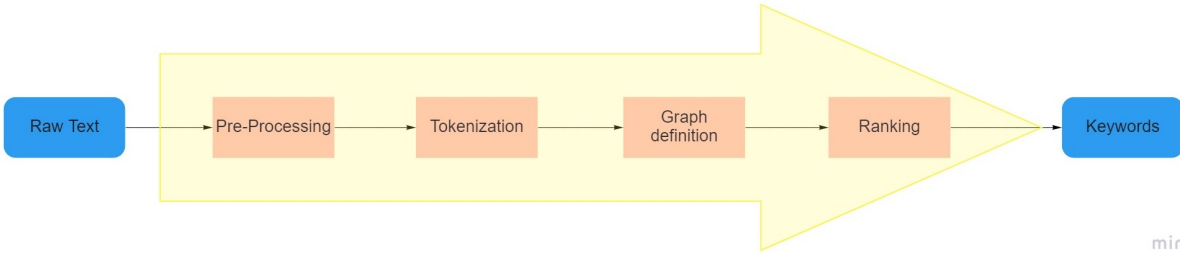

Figure 3.10: Keywords Extraction Method

3.3.1 Pre-Processing

Pre-processing is very important in Natural Language Processing, as it is important to filter the text as much as possible to obtain the best resolution performance. The process therefore varies according to the applications for which this text will be used.

Because it is good practice to make textual data noise-free, a basic text cleanup has been applied to each of the problem descriptions proposed as input for keyword extraction.

Starting from the raw data, the proposed pre-processing performs these main steps:

- Remove Punctuation: remove all unnecessary and insignificant symbols.
- Remove uppercase letters: convert all text to lowercase, to limit the algorithm to only process 26 English alphabets, instead of 52.
- Remove Stop Words: remove all words from a list of commonly used natural language words that carry little useful information. Examples of stop words in are "a", "the", "is", "are" and so on.
- Divide the text in sentence.

3.3.2 Tokenization

Tokenization essentially consists of dividing a sentence, paragraph, or entire text document into units, called tokens. In this case, the text already divided in to sentences, is broken into words. Tokens are non-sensitive data extracted from sensitive document, and they allow to understand the context and help interpret the meaning of the text by analyzing the sequence of the words stored as a vector representing that document.

This process then allows an unstructured string (text document) to be transformed into a numeric structured data suitable for machine learning. Before applying the MATLAB function the division of the text in tokens is therefore performed, as the function requires data consistent with the machine language, that means in the form of numeric data. This process resulted as an essential step when working with text data.

3.3.3 Node and Edge Graph definition

In this step, our starting point is text that has been processed, divided into sentences and tokenized.

These tokenized sentences are concatenated to obtain a large text to which a syntactic filter obtained by part-of-speech task [89] that filters nouns and adjectives is applied. Noun and adjective are the category of word tags selected to define a subset of words that are potential keywords. These are the nodes of the graph.

The concept of co-occurrence windows [90] is used to create the edges. A number n is chosen, that represents the window of words within any links between the words are established, and it should be less than the number of words present in a sentence: for the case in question, 2 has been chosen, which is the minimum acceptable value. Then edge are added from one word in the window to all the nouns and adjectives that appear in the window of 2 words; moving to the next windows of 2 words, additional edges are added

between the words which are nouns and adjectives in their respective windows; and the process continue in this way until the end of the document. This creates a co-occurrence plot based on certain window sizes.

In this way the graph is created, where the nodes are nouns and adjectives and the edges are the relationships that are created based on the context that they share within a window of a certain number of words.

3.3.4 Ranking

PageRank (PR) is a basic Google Search algorithm used to present the ranking of web pages in search engine results [91]. PageRank is based on counting the number of links to a page and the quality of these in order to determine according to a certain scoring function, a rough estimate of how important the website is.

Starting from the graph created following the rules described in the previous subsection, it is defined V as the set of vertices and E as the set of edge. Defining also, $In(V_i)$ the degree for the vertex V_i (number of edges that point to V_i) and $Out(V_i)$ is the out degree of the vertex V_i (number of vertices that vertex V_i point to). Then the final score for each of the vertex V_i is given by:

$$S(V_i) = (1-d) + d \cdot \sum_{j \in In(V_i)} \frac{1}{|Out(V_j)|} S(V_j)$$

where d is the damping factor: the typical value is equal to 0.8, and its role in the formula is to give an initial score to the newly added nodes.

The damping factor therefore comes into play when the algorithm has to deal with the most recent nodes that are created. When the score of the newly arrived node is calculated, a summation of all the weighted scores of the nodes to which the new one is connected must be evaluated but its initial value is zero, since in the start no connection are established. In the end, the damping factor is a kind of initialization to the score of a new node in the graph.

Therefore to calculate the PageRank for a certain node in a graph, it must be sum the initialization value made from the damping factor plus the summation of the scores of the nodes pointing to the node in exam weighted by the out degree of each of these nodes.

Once one of the scores has been performed, the scores of the nodes connected to it need to be updated accordingly, as each score depends on the score of the connected nodes. In this way it would recursively update all the scores until the deviation between the obtained values changes no more than 0.01. When some sort of equilibrium is reached the algorithm stops and the scores obtained at that point are the final set of scores, which are called PageRank score that symbolizes the importance of each node in the graph or each page on the web.

To give an example of the application of this whole approach on the dataset analyzed in this thesis, Table 3.3 presents the top three keywords extracted and selected for the following issues:

- **Issue 1**: *"We are seeing the issue that PX4 lists no logs when using QGC to request a log list or using some other mavlink application..."*.
- **Issue 2**:*"...In the mixer file the steering was controlled by roll, but in PX4 rover steering is controlled by yaw. And this was the reason why the attitude control did not work correctly..."*.
- **Issue 3**:*"... Pixhawk is unable to rotate around the YAW axis. After flying above the relative altitude of zero, the yaw rotation ability comes back. ..."*.

	KW1	**KW2**	**KW3**
Issue 1	*log*	*list*	*mavlink*
Issue 2	*control*	*yaw*	*attitude*
Issue 3	*altitude*	*zero*	*yaw*

Table 3.3: Examples of Keywords Extracted

.

3.4 Manual Classification of the Issues

Before starting the manual and automatic classification of each data of the extracted dataset, it is necessary to define the categories and severity levels according to which the data will be assigned a label. These definitions are crucial for the classification as they were the guidelines followed by the experts to manually classify the data but also the way to verify that the classification made through an automatic machine learning classifier respected the definitions associated with each class.

3.4.1 Severity classes

It should be noted that the severity of the risk is particularly susceptible to the type of the damage and the probability of its occurrence [92]. In the context of UAVs, the safety risk of each problem is highly dependent on the application scenarios. For example, the loss of communication can have different frequencies and levels of severity depending on the mission area. The frequency of communication problems may be higher in rural areas than in urban areas which are generally equipped with better communication services, but the impact of failures would be lower in rural areas. Therefore, the severity level would be

labeled as critical when the quality of communication degrades if the drone is operating on top of highways. When the application context changes and the drone is considered to be on top of the forest or in areas with no people, gravity can be labeled as marginal as it will only lead to damage of the drone.

Therefore, when labeling manually, severity must consider several factors including application context and distribution assumptions to be interpreted accurately.

The level of severity of the risk is estimated based on the level of injury or the harmful impact on people, drone and the environment. The following four severity level definitions have been adopted for classification [40]:

1. **Negligible**: The faults/failures/threats does not cause harm to self, mission, environment or to human.
2. **Marginal**: The faults/failures/threats may cause self harm to the drone or it's internal systems.
3. **Critical**: The faults/failures/threats may cause self and mission harm including some harm to the environment but does not include harm to human.
4. **Catastrophic**: The faults/failures/threats may cause serious harm to self, mission and environment including human injuries or casualties.

3.4.2 Category classes

To define the categories of failure a table that provide the list of potential drone hazards is defined. This list has been identified using reactive methods, which manage problems once they arise or are encountered. Databases of accidents, safety and flight reports, surveys, maintenance reports are used [38].

Such work was performed also in the work of Allouch, Kouaa, Khalgui, and Abbes [40] in which the authors used the following documents:

- Accidents and incidents databases: Federal Aviation Administration preliminary reports of Unmanned Aircraft Systems Accidents and Incidents (FAA UA A&I) database, National Transportation Safety Board (NTSB), FAA's Aviation Safety Information Analysis and Sharing (ASIAS) database, FAA's Accident and Incident Data System (AIDS).
- Safety and flight reports: NASA's Aviation Safety Reporting System (ASRS), Annual Insurance Report17 and ICAO Safety Report.
- Maintenance reports: FAA's Aviation Maintenance Reports.
- Surveys: NTSB's Review of Aircraft Accident, FAA's Summary of Unmanned Aircraft Accident/Incident.

Type	Examples
Communication	Network congestion, network unavailability/delay, Network Jitters.
Algorithmic	Verification error, decision-making error, delayed responses.
Software	Control system failure, flight control system/verification error, autopilot, error, system operation erros, bugs in code, process errors, vision system failure
Hardware	CPU error, avionic hardware, flight sensors.
Navigation	GPS signal loss/error, GPS spoofing, DS-B signal inaccuracy, navigation system error, attitude error, erroneous waypoint
Technical factor	Battery depletion, faulty battery cell, power loss, technical malfuncion, inappropriate charge cycle,loss of control, loss of transmission.
Mechanical	Mechanical fasterner failure, actuation failure, motor.
Thermal	Freeze, explosion.
Electronic	Power loss, propulsion failure, saturation, overflow
Human factor	Lack of safety culture awerness, security attack (on the Ground Control Station, on the DataLink, on UAV), pilot error (inexperience pilot, not familiar with the area, faligue, rush)

Table 3.4: Types and examples of potential drone hazard.

From the list of documents that the work presented are extracted the most relevant damages in the context of the critical components of UAV systems that are linked or are part of the Autopilot software system, and then the Table 3.4 is defined. This Table was used as a preliminary list for defining the groups to which the fault categories belong, and then this categorization is used to perform manual identification and then to check the correctness in the classification made by the experts.

Ultimately, four critical components of UAV systems were identified that are essential to ensure safe flight, including communication, flight mission, position estimator, and the flight controller itself. Although the estimator is part of the flight controller, it was considered as a separate component due to its critical function in UAV systems. Therefore, the flight controller involves all problems except the one related to the estimator. Therefore, the four categories defined in this study are as follows:

1. **Communication**: This category includes issues related to communication with Autopilot.

2. **Mission**: This category includes faults in missions or faults related to the transition from one flight mode to another during the mission.

3. **Estimator**: This category includes issues related to the position and altitude estimator, one the most critical component of Autopilot that uses the sensors and GPS

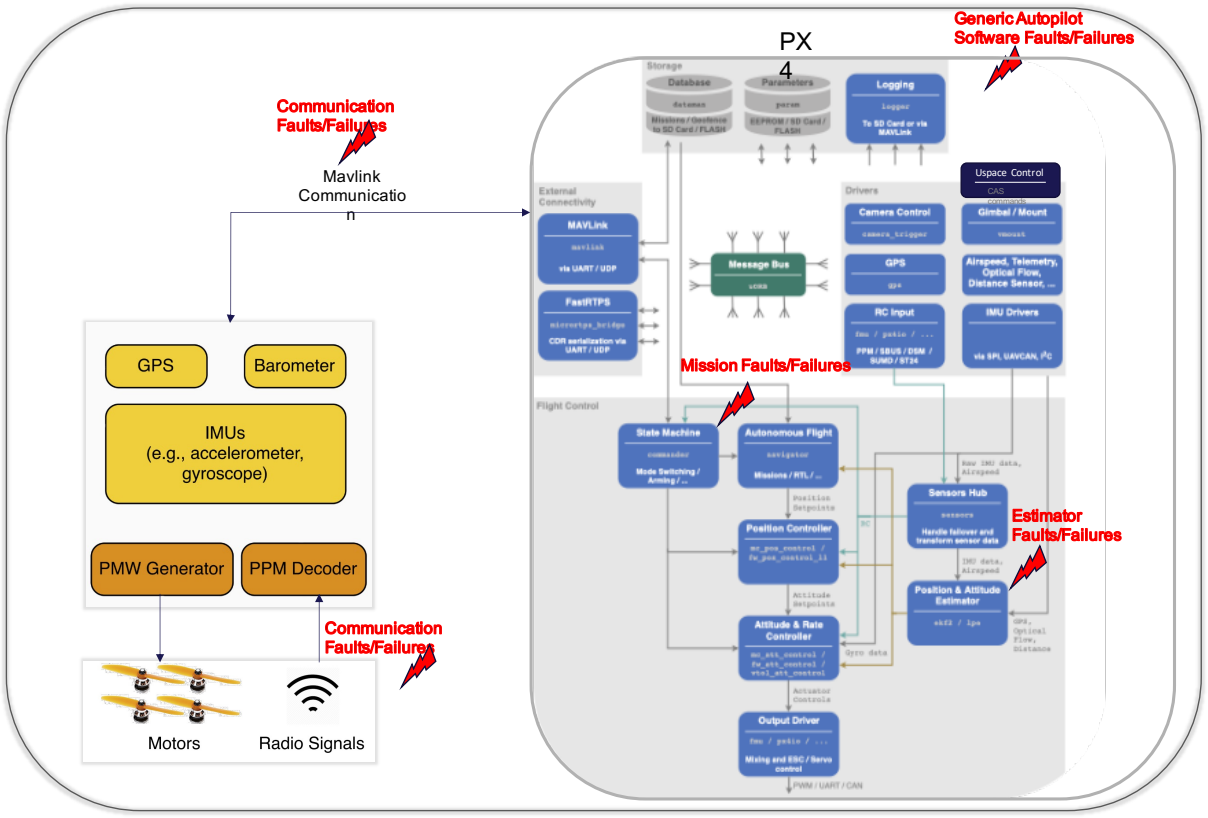

Figure 3.11: Drone Architecture and Possible Faults/Failures

inputs to estimate the drone current position.

4. **Autopilot**: This category includes the software generic faults found in the source code of PX4.

After defining the categories and severity levels, the manual classification process was started. The comprehension of the 610 problems, selected for the analysis, and therefore the assignment of the relative category and severity, was carried out by five researchers expert in UAV areas, including the author of the thesis report. Experts were provided with a subset of the 610 problems with the original description reported by GitHub users, the keywords extracted for each report, and information on categories and severity levels.

3.5 Automatic Classification of the Issues

After manually analyzing and classifying 610 issues, four different classifiers have been trained in order to automate the classification of the issues. The four classifiers selected in

for this study are:

- Naive Bayes[6]: it is a probabilistic classifier, based on Bayes' theorem with an assumption of independence between classes, i.e. that the presence of a particular characteristic in a class is not correlated to the presence of any other characteristic.

- Decision Tree[7]: it is based on a graphical representation of all possible decisions that could be made based on a certain condition. The graph is a tree structure, where the internal nodes represent the characteristics of a dataset, the branches represent the decision rules and each leaf node represents the class it is assigned for classification when the condition on the branch leading to that leaf is met.

- Support Vector Machine[8]: it uses a technique called the kernel trick to transform the data in order to find an optimal boundary between the possible outputs. Depending on which side of the boundary the new data is, this is assigned to the class in question or not.

- k-Nearest Neighbor[9]: K is the number of closest neighbors to a new unknown variable that needs to be classified. The new datum is classified according to the class already assigned to its neighbors, the most common class among its k closest neighbors is the class assigned to the new data.

These classifiers are chosen because they are the commonly used algorithms for building multiclass classification models [93] with high performance. They are also capable of achieving good results in the cases where the data type is natural language-based text, as in the case in question [94].

The classifiers are formed using the 610 manually classified problems. For each of the four previously mentioned classifiers, the model was trained following a 4-fold cross-validation to prevent overfitting.

As mentioned, when developing a machine learning model, it is necessary to evaluate the performance of the model by splitting down the dataset into training data and test data. The training data is used to train the model, and the same model is then tested on the test data to evaluate the performance of the model. Test data must be kept independent of training data so that no data loss occurs. However, dividing the data for training and testing differently also changes the accuracy of the model, so it is not possible to get a fixed accuracy for the model.

[6] `https://it.mathworks.com/help/stats/fitcnb.html`

[7] `https://it.mathworks.com/help/stats/fitctree.html`

[8] `https://it.mathworks.com/help/stats/fitcecoc.html`

[9] `https://it.mathworks.com/help/stats/fitcknn.html`

In 4-fold cross-validation, the original dataset is uniformly partitioned into 4 subparts or folds, and for each iteration, among the 4-folds, one group is selected as validation data and the remaining groups (3) are selected as training data. The process is repeated 4 times until each group is treated as validation. The accuracy of the final model is calculated by taking the mean accuracy of the 4 validated model.

This validation process is popular because it is simple to understand and because it produces a less biased or optimistic estimate of the model's ability to classify unknown data.

3.5.1 Confusion Matrices

To describe the performance of a classifier model on a set of test data for which true values are known, a confusion matrix is constructed. Confusion matrix is a performance measure for a machine learning classification problem where the output can consist of two or more classes. It is a table with four different combinations of predicted and actual values and is useful for measuring Recall, Precision, Specificity, Accuracy and AUC-ROC curves.

Each entry in a confusion matrix denotes the number of predictions made by the model into which it classified the classes correctly or incorrectly. Confusion Matrix has four types of terms which are true positive (TP), true negative (TN), false positive (FP) and false negative (FN). The definition of these values is as follows:

1. **TP**: Number of issues that are correctly classified positive (i.e., the category/severity predicted by the classifier is the actual category/severity to which the issue belongs).

2. **TN**: Number of issues that are correctly classified as negative (i.e., the category/severity that the classifier does not predict for an issue is not the actual category/severity to which the issue belongs).

3. **FP**: Number of issues that are classified as positive but are in fact negative (i.e., the category/severity predicted by the classifier is not the actual category/severity to which the issue belongs).

4. **FN**: Number of issues that are classified as negative but are in fact positive (i.e., the category/severity that the classifier does not predict for an issue can be the actual category/severity to which the issue belongs).

Most of the time the confusion matrix is used for a binary classification problem that has only two classes to classify, usually a positive and a negative class. Unlike binary classification, in multiclass classification cases there are no positive or negative classes. To generate the matrix it is necessary to find TP, TN, FP and FN for each single class and the negative classes are all those that do not belong to the one under examination. Finally, a matrix is defined that has $n\ x\ n$ entries, where n is the number of classes.

3.5.2 Metrics

After having extracted all the above values from the matrices, several performance indices including *Accuracy*, *Precision*, *F1-score* and *Recall* are calculated to make a fair comparison between the classifiers. The following classic formulas for the metrics are used:

1. **Accuracy** = (TP+TN)/(TP+FP+FN+TN). It is a ratio of correctly classified issues to the total number of issues.

2. **Precision** = TP/(TP+FP). It is the ratio of correctly classified issues to the total issues that are classified with the same category/severity. In simple terms, precision shows what percentage of the positive predictions made were actually correct. The precision is the number of correctly identified positive results divided by the number of all positive results, including those not identified correctly.

3. **Recall** = TP/(TP+FN). It is the ratio of correctly classified issues to all issues with the same actual category/severity. In simple terms means, what percentage of actual positive predictions were correctly classified by the classifier. The recall is the number of correctly identified positive results divided by the number of all samples that should have been identified as positive.

4. **F1 Score** = 2*(*Precision***Recall*)/(*Precision*+*Recall*). It is the harmonic mean of precision and recall. It is the harmonic mean of the precision and recall. The highest possible value of an F-score is 1.0, indicating perfect precision and recall, and the lowest possible value is 0, if either the precision or the recall is zero.

3.6 Generalization and Validation

The last phase of this study is focused on demonstrating the generalization and validation of the obtained results. To do so, the best classifier is selected from the previous phases and tested it against a new test set from the remaining dataset different from the 610 issues used in the previous phases.

To build this new test set, the following criteria are applied:

- Reports that are labeled as a bug.

- Reports that are not included in the previously selected reports (610 issues).

- Reports with at least 80 words (80 words are large enough for the keyword extracting function to work with a detailed description of the problems and thus produce the keywords more accurately).

From the first criterion we obtained 2283 issues, applying the second, the number decreased until 1673, and in the end, with the last criterion 490 reports were found, which were used to test the classifier model selected for this analysis.

To verify the correctness of the classifier on this new dataset, 150 of the issues reported in the data were manually analyzed by two experts (one of them is the author of this document), to be compared with those identified automatically.

Chapter 4

Results and discussion

This section provides the results obtained at each stage of the methodology discussed in the previous chapter, from manual classification to generalization and validation of the results. Each phase of the approach provides relevant results in the study of issues related to the PX4-piloted flight of autonomous drones, but also provides input for the next step, which is further analyzed or validated according to the methodology.

4.1 Results of Manual Classification

To summarize the manual analysis, Figure 4.1 and Figure 4.2 show the number of issues classified for each Category and Severity, respectively.

4.1.1 Severity classes

Regarding the Severity, the majority of the issues are classified as *Negligible* (which is not surprising because many of the issues, as can be seen in Figure 3.6, are not all issues but also enhancements, questions, etc), and then the number of problems assigned to each class decreases when the severity increases, going from *Marginal* to *Critical* and at the end *Catastrophic*.

During the severity assessment, it is discovered that most *Catastrophic* cases are related to system crashes (*"...UAV started tilting uncontrollably and then crashed into a wall..."*), caused by failed missions especially in case of transition from one flight state to another or by communication problems with the autopilot and the system in flight. While *Marginal* or *Critical* ones are more originated by the estimator (*"...The following points don't work as they should upon triggering a GPS Failsafe on a VTOL drone in FW..."*) or mission (*"...Takeoff command temporarily rejected..."*).

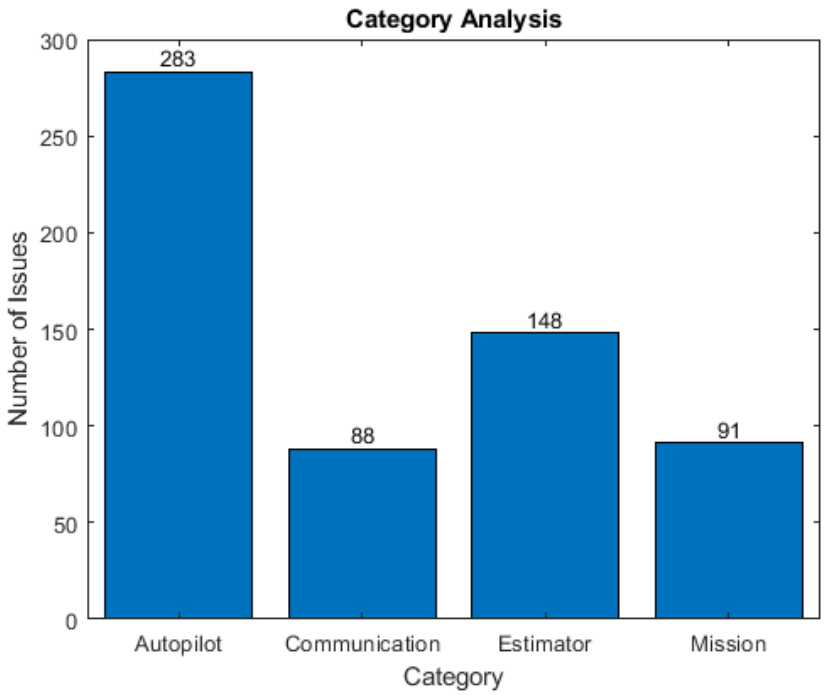

Figure 4.1: Number of Issues for each Categories Classified Manually by Experts

4.1.2 Category classes

The analysis reports that the most common faults/failures are related to the *Autopilot* category following by the *Estimator*, and *Communication* and *Mission* faults categories with similar numbers.

As mentioned, throughout the manual analysis procedure, it was found that the most significant number of faults regard the internal issues of *Autopilot*, many of which concern the change of flight mode (*"...I found that when I choose fw to mc, the drone attitude is unstable!..."*) and others concern configuration errors(*"...User reported worse altitude hold after upgrading from PX4..."*). Other common causes of failure are related to the *Estimator* class. Less frequent issues are *Communication* errors between UAV components or between the UAV and the control station (*"...the uORB messages stopped updating..."*).

To validate the manual classifications made by the different people who participated in the analysis, different problems were chosen at random and discussed among the same experts in order to verify that the way of interpreting and identifying the various issues was similar among all. It turned out that regarding the definition of severity, the classification was not a difficult task as all the experts agreed on the assignment of the severity level for each of the data examined. On the other hand, in evaluating the category, there have been various cases in which the univocal definition of a label has created confusion

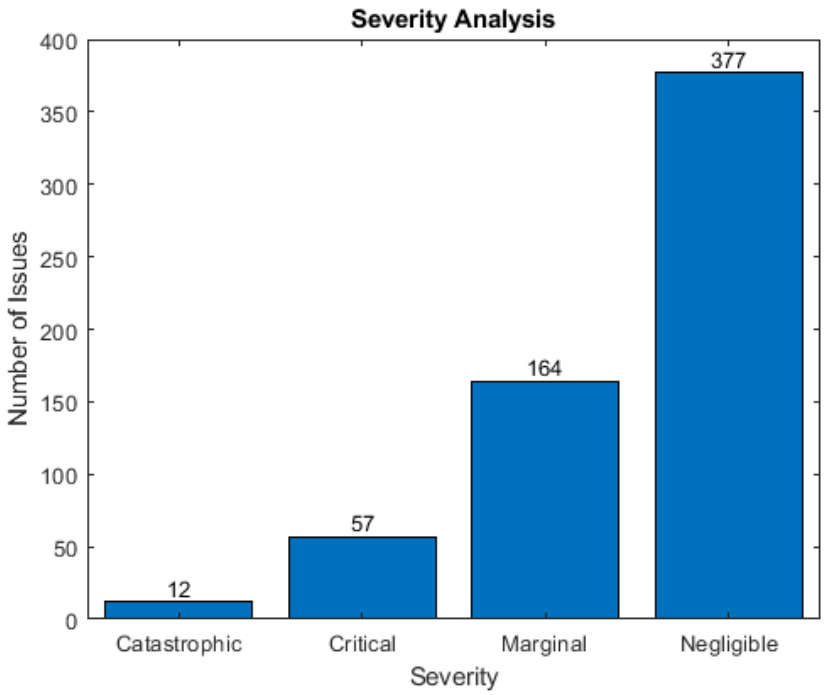

Figure 4.2: Number of Issues for each Severity Classified Manually by Experts

and required the comparison between experts to have a clearer interpretation of the problem. For example for reports like *"...every time I try to calibrate my radio I got the following message from QGC: "detected 0 radio channels. to operate px4 you need at least 5 channels"..."* it was discussed whether to identify it as *Autopilot* or as *Communication*, as the problem concerns the failure in the use of the radio during the autonomous flight of the drone, but carefully reading the description of the issue it is explained that, by testing the system with Ardupilot, it seems that everything works correctly. Therefore the problem lies in the PX4 software and it should be label as *Autopilot* issue.

This confirms that the automation procedure is useful in this context, as manual classification is time-consuming, error-prone, and limits scaling to large datasets.

4.2 Results of Automatic Classification

Four classification models for the classification of fault categories and four for the classification of Severity levels were built, with four classes for each one (i.e., four categories and four severity levels), using the manual validate dataset obtained from the previous section for training the classifiers.

4.2.1 Confusion Matrices

To compare and evaluate the performances of the models in order to find the best algorithm for classifying issues in the context of UAV PX4-related failure, performance indices were calculated from the confusion matrices entry (Figure 4.3 and Figure 4.4).

The row of each matrix represents the occurrences in a true class, while each column represents the occurrences in a predicted class. The correct classifications are shown in blue in the diagonals, while the incorrect classifications are shown in red in the rest of the matrices.

Figure 4.3 and Figure 4.4 show the detailed performance in cofusion matrix form of Decision Tree, K-nearest neighbor, SVM and Naive Bayes classifiers in the case of severity and category respectively.

In the case of severities (Figure 4.3), the **Decision Tree** matrix appears to have a high number of correct classifications, and moreover it is the only classifier model that detects *Catastrophic* problems even though seven issues that are catastrophes were misdetected as *Marginal*. For the rest of the classes the wrong predictions are distributed homogeneously around the diagonal. In **SVM** instead, most of the problems are foreseen as *Negligible*. The issues predicted as *Marginal* are overall more correct, while in predicting the *Critical* class only one identification is correct. Differently **Naive Bayes** predicts most problems as *Negligible* or *Marginal* with few errors around these classes, and the issues identified as *Critical* are more correct. In **K-Nearest Neighbors** instead, there is no class that is correctly predicted, and the distribution of the data is homogeneous between *Negligible*, *Marginal* and *Critical* labels.

As regards the classification of categories (Figure 4.4), the **Decision Tree** matrix has high values in the correct classification on the diagonal and low values of incorrect classifications homogeneously distributed among all categories, except for high data erroneously predicted as *Autopilot*. **K-Nearest Neighbors** has the highest numbers of wrong classifications, while **SVM** performs more correct prediction on *Mission* and *Communication* problems but classifies *Estimator* issues very badly. Finally, **Naive Bayes** has great diagonal values and few erroneous predictions, competing alongside the **Decision Tree** to become the best category classifier model.

4.2.2 Metrics

The accuracy is the first metric extracted from the confusion matrices and the results, shown in Table 4.1 and Table 4.2, present high values for all classifiers, going from 78.77% in the case of K-Nearest Neighbors for the category classification to 93.43% of Decision Tree for severity classification. Although classification accuracy is the most used metric for evaluating classification since it is easy to calculate and interpret, it works well with unbiased distributed class datasets and slightly biased class distributions, but in case the asymmetry in class distributions is severe, accuracy becomes an unreliable measure of

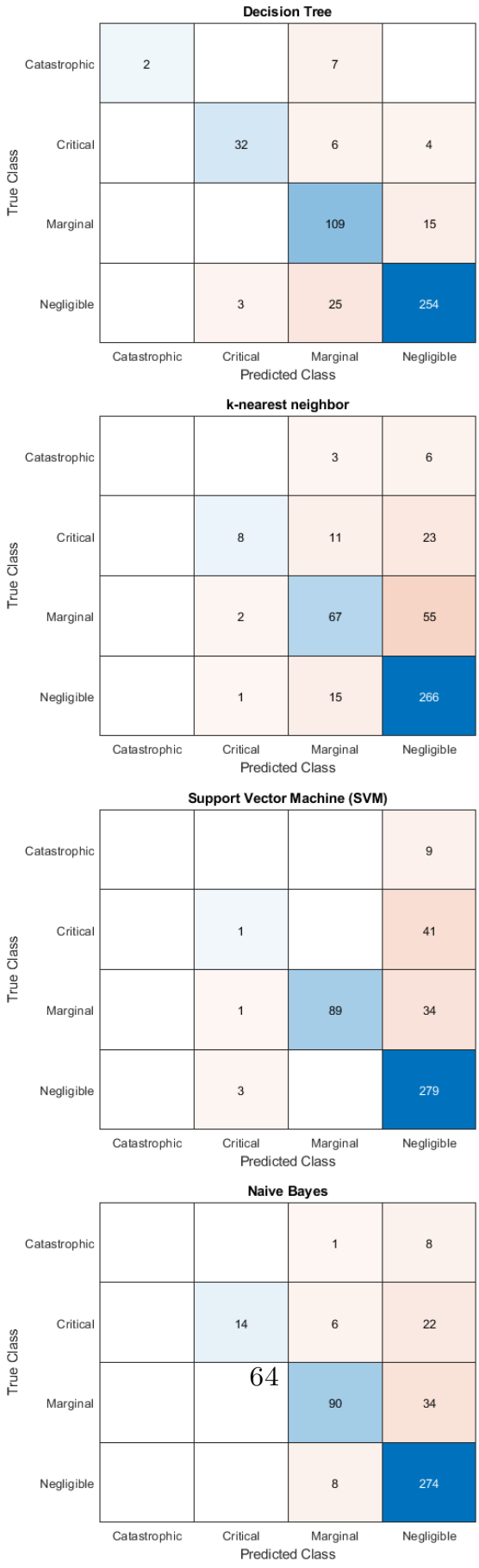

Figure 4.3: Confusion Matrix of Decision Tree (first), K-Nearest Neighbors (second), SVM (third) and Naive Bayes (fourth) classifier for predicted Severity.

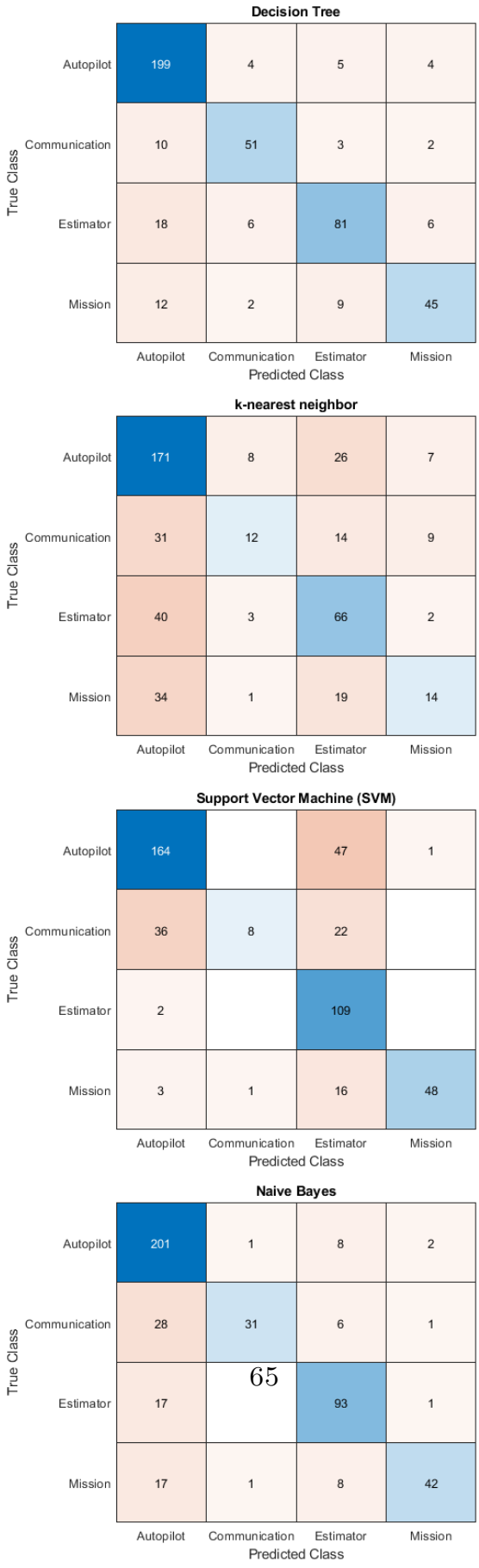

Figure 4.4: Confusion Matrix of Decision Tree (first), K-Nearest Neighbors (second), SVM (third) and Naive Bayes (fourth) classifier for predicted Category.

model performance. Therefore, most of the time it comes to the conclusion that a high accuracy score is good, suggesting that a model is performing well or even excellent when, in fact, it is not.

This is what happens in the case in question, where the distribution of the dataset is unbalanced and therefore, accuracy is no longer an adequate measure, as it does not distinguish between the number of correctly classified examples of different classes. Hence, it can lead to erroneous conclusions [95].

For this reason, following the common practice of using alternative metrics to summarize model performance for unbalanced classification problems, Accuracy, Recall, and F1 Score are calculated and compared.

Table 4.1: Average Performance of Classifiers in Classifying the Fault Severity.

Name	Accuracy	Recall	Precision	F1-score
Naive Bayes	0.9135	0.5077	0.6669	0.5425
Support Vector Machine	0.9037	0.4327	0.4921	0.4358
Decision Tree	0.9343	0.6910	0.8965	0.7286
k-Nearest Neighbor	0.8731	0.4185	0.5463	0.4382

Table 4.2: Average Performance of Classifiers in Classifying the Fault Category.

Name	Accuracy	Recall	Precision	F1-score
Naive Bayes	0.9015	0.7183	0.8563	0.7581
Support Vector Machine	0.8599	0.6357	0.8076	0.6338
Decision Tree	0.9114	0.7757	0.8145	0.7921
k-Nearest Neighbor	0.7877	0.4472	0.5213	0.4517

Table 4.1 and Table 4.2 report *Accuracy*, *Recall*, *Precision* and *F1-score* of the four models for the classification of Fault Category and Severity respectively. Among the four classifiers, it is observed that Naive Bayes and Decision Tree perform well, in terms of *Accuracy*, with the dataset provided to the models, but Decision Tree shows slightly better performance giving us 91.14% and 93.43% *Accuracy* on Fault Category and Severity respectively. In terms of *Recall*, the Decision Tree always shows significantly better results than the other classifiers. In terms of *Precision*, Naive Bayes shows significantly better results than the other classifiers. This all means that the Decision Tree can classify the reports with a relatively good performance, but it generates more false alarms than Naive Bayes. On the other hand, Naive Bayes cannot classify most reports correctly, but it does not generate many false alarms. This implies that each of these two classifiers is suitable for different application scenarios. When dealing with a critical system in which we expect to identify all catastrophic issues related to a critical component of the system

(e.g., estimator in this case), the **Decision Tree** is a better choice as it classifies most of the reports correctly, no matter it generates some false alarms. In a different scenario, when there are limited resources (i.e., human resources, money, and time) to fix the bugs, **Naive Bayes** is the best option as it identifies some of the critical issues and does not generate many false alarms resulting in waste of the resources.

By studying in more detail the model created by the **Decision Tree**, which is the best model in the context of our work, we observed that it generates the tree in the following way. For each node, it checks whether the keywords provided as input are within the group of words that are identified by the algorithm as the most discriminative words for a specific class (e.g., *Catastrophic* in Severity). The most frequent class in the dataset is checked in the first node (e.g., *Negligible* in Severity), and the less frequent classes are checked in the subsequent nodes (*Marginal*, *Critical*, and *Catastrophic*).

Category	Keywords
Autopilot	px4, log, mode, test, error, build, firmware
Estimator	gps, sensor, flight, image
Mission	position, log, set, vehicle, setpoint
Communication	mavlink, info, message, sensor, code

Table 4.3: Frequent Keywords for Fault Category class.

In order to understand how the associations between classes and keywords are made by the algorithm, the frequency of the keywords associated with each category are summarized in Table 4.3. As a result, "px4", "log", "mode", "test", "error", "build" and "firmware" are the most repeated keywords for the issues categorized in *Autopilot* category. While for *Estimator* category, the most frequent words are "gps", "sensor", "flight" and "image". For *Mission* category, we have "position", "log", "set", "vehicle" and "setpoint" as the most frequent keywords. Finally, the most frequent keywords for *Communication* issues are "mavlink", "info", "message", "sensor" and "code". As shown, there are a few overlapping between the most frequent keywords of different categories (e.g., "log" and "sensor"). To achieve better results from automatic classification (and even manual classification), we should ask and help the community (e.g., by providing suggestions) to describe the issues with more detail and precision.

4.3 Generalization and Validation Results

In order to find out whether the proposed approach for classification of reports based on keywords can be generalized, another set of reports from the Github dataset, different from the ones used for training the model (i.e., different criteria are used to select the bugs), were selected, and the Decision Tree classification models built in the study were tested on this

new dataset. Since this new group of issues are not classified manually, it is not possible to check the correctness of classification of all 490 issues. For this reason, we first verified whether there is any consistency between manual and automatic classification. To do so, we counted the number of issues assigned to each class (in terms of fault category and severity), both for the automatically classified dataset (490 issues) and for the manually classified dataset (610 issues) and the results are summarized in tables and graphs.

The graphs in Figure 4.5, Figure 4.6, Figure 4.7 and Figure 4.8 show the occurrences of the category classes for the dataset where the classes are manually identified and for the dataset where the classes are identified by the classifier, and the occurrences of the severity classes for the same dataset respectively. The graphs allow an easier interpretation of the data and also an immediate comparison of the results. From Figure 4.5, Figure 4.6 we can see how the *Autopilot* categorized issues has passed from 46.39% to 64.49% identifications by the automatic classifier in the new dataset. *Mission* and *Communication* are identified with the same percentage in both datasets of reported issues, while *Estimators* are slightly lower in the automatic case. In Figure 4.7 and Figure 4.8 the severities are presented in percentages, and the first important thing to note is that in the automatically classified dataset no data has been identified as *Catastrophic*, although further analyzes have revealed that some problems ending with crushing were detected. Probably because the extraction of the keyword did not select in some cases words that could indicate the real seriousness of the problem. Regarding the other classes, these are ranked with similar percentages among the graphs.

Unlike the figures, in the Table 4.5 and Table 4.4 the information is mixed and the occurrences of the reports of each category are presented for any gravity are presented for the manually extracted data and the automatic one, respectively. Occurrences appear homogeneous between the two cases, with the exception of problems classified as *Negligible* and *Marginal Autopilot*. In correspondence with the increase in emissions classified as *Autopilot* there is therefore a decrease in those defined as *Estimator*, but the other two categories have the same percentage in both datasets. This is not surprising since the first dataset is chosen by selecting the issues that are reported with a longer and more accurate description of the problem, while usually the shorter bodies that occur in greater numbers in the second dataset are related to update or installation problems, which we have identified *Autopilot* problems of *Negligible* severity as *Autopilot*.

	Catastrophic	**Critical**	**Marginal**	**Negligible**
Autopilot	3(0.49%)	12(1.97%)	43(7.05%)	225(36.88%)
Communication	1(0.16%)	12(1.97%)	19(3.11%)	56(9.18%)
Estimator	2(0.33%)	11(1.8%)	65(10.65%)	70(11.47%)
Mission	6(0.98%)	22(3.61%)	37(6.06%)	26(4.26%)

Table 4.4: Occurrence of Classes in the Manually Classified Dataset with 610 issues.

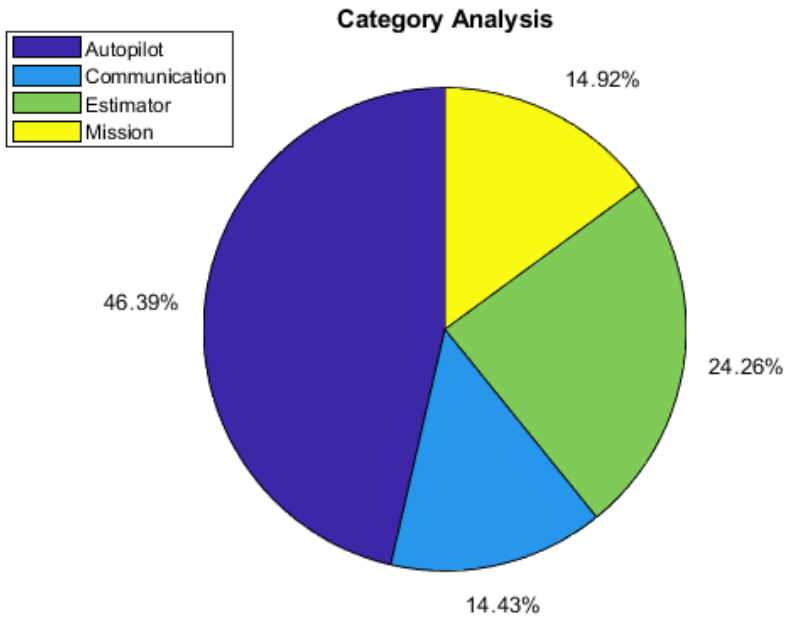

Figure 4.5: Pie chart of the Category classes occurrences manually identified in the dataset of 610 issues.

	Catastrophic	Critical	Marginal	Negligible
Autopilot	0(0%)	13(2.65%)	68(13.88%)	235(47.96%)
Communication	0(0%)	3(0.61%)	13(2.65%)	40(8.16%)
Estimator	0(0%)	8(1.63%)	25(5.10%)	30(6.12%)
Mission	0(0%)	10(2.04%)	30(6.12%)	15(3.06%)

Table 4.5: Occurrence of Classes in the Automatically Classified Dataset with 490 Issues.

In addition to the consistency between the manual classification and automatic classification, a sample of the issues from the new dataset (150 out of 490) is also selected and the classification is manually checked. Table 4.6 shows the results of this manual classification. Decision Tree classifier works reasonable in the classification of fault severity detecting 105 correct class of severity over 150 issues selected (*Accuracy* of 70%), thus even in the case of unknown issues (i.e., issues different from those used for building the model in terms of structure and details) the automatic classifier provide good results. In terms of Fault Category, the result is similar as the previous case with 103 issues correctly identify over 150 selected (*Accuracy* of 68.67%).

These final data, even if they do not give an excellent score, turn out to be good enough

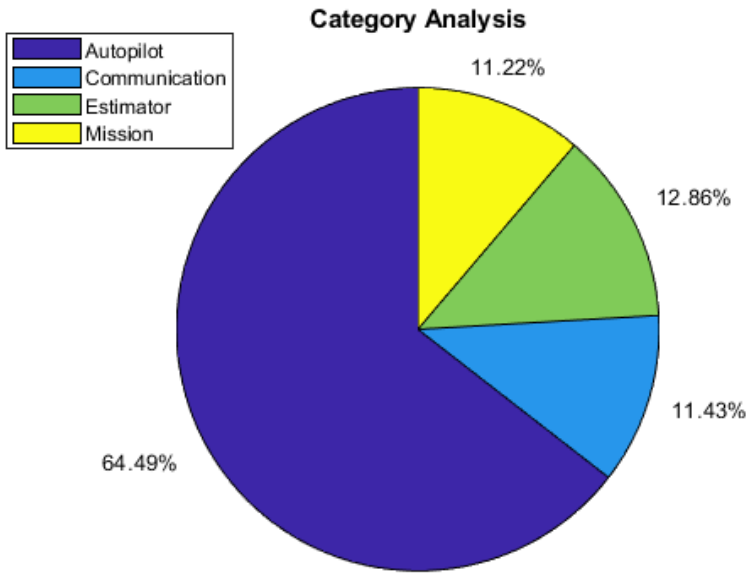

Figure 4.6: Pie chart of the Category classes occurrences identified by the classifier in the dataset of 490 issues.

values to confirm that this work is proper and practicable and therefore, this study on creation of an automatic classifier can be used in the classification of issue extracted from GitHub also for different systems, or to further analyze systems related to drones, to have a more complete perception of the damage that can occur during the flight of autopiloted UAVs.

	Corrected classified	Total Issue	Percentage
Severity	105	150	70%
Category	103	150	68.67%

Table 4.6: Validation Analysis.

4.4 Results Discussion

The analysis carried out in this work confirms that it is possible to automatize the classification of bug reports written in natural languages by using machine learning algorithms trained based on the keywords extracted from the reports. An example of

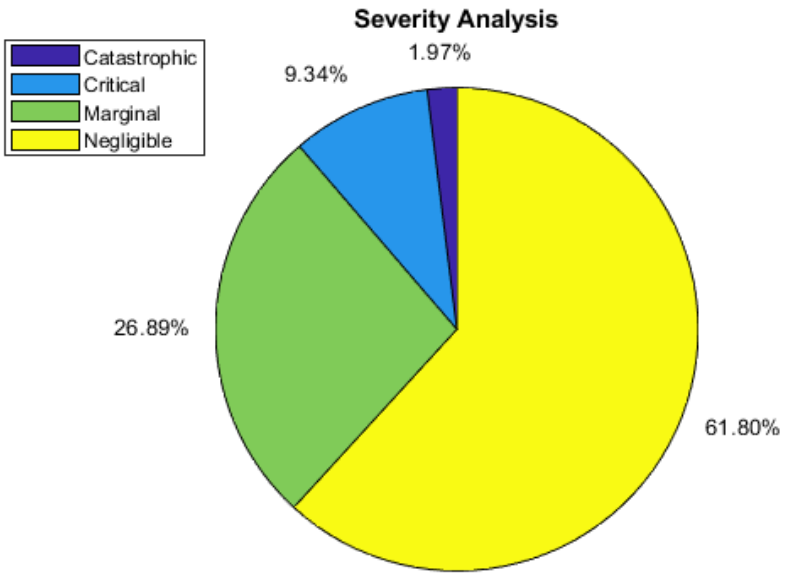

Figure 4.7: Pie chart of the Severity classes occurrences manually identified i the dataset of 610 issues.

the potentiality of this classifier can be seen in the case of the issue that reports *"... an accident while using terrain following and I found that the vertical velocity estimate of the EKF is ignored in the configuration ..."*. The "crash" keyword (extracted from the description of the issues) is associated, by the classifier, to the severity level of *Catastrophic*, and the keywords "estimate" and "EKF" to the fault category of *Estimator*.

In the end, the main insights of this study are highlighted in the following:

- Not all machine learning algorithms are able to achieve a good performance, but it is possible to train some algorithms like Decision Tree and Naive Bayes to classify the bug reports written in natural languages.
- Classifiers perform differently and are suitable for different scenarios. The Decision Tree algorithm is suitable for classifying bug reports of critical systems as it classifies most of the cases correctly, despite producing some false alarms. In contrast, Naive Bayes is more suitable when we need to classify the bug report of a low critical system with limited resources for bug review, as it produces fewer false alarms than the Decision Tree model.
- In order to achieve better results, we need to somehow help the community to generate

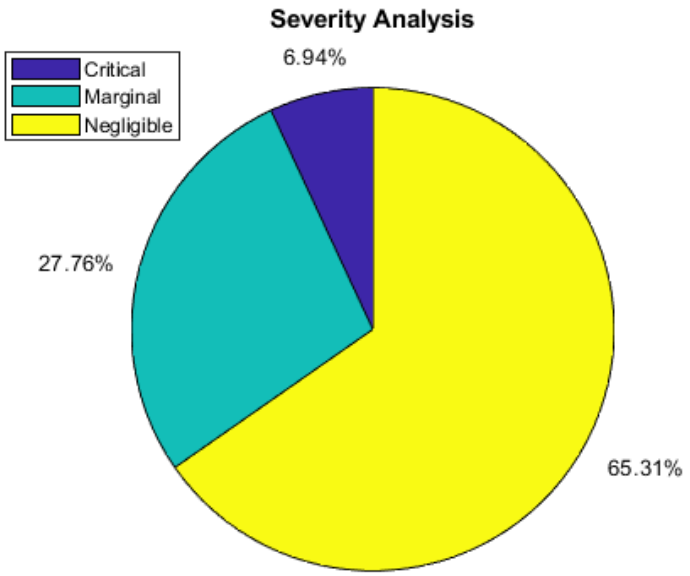

Figure 4.8: Pie chart of the Severity classes occurrences identified by the classifier in the dataset of 490 issues.

more detailed and more precise reports to avoid overlapping keywords. This can be done by making the version control systems equipped with an intelligent suggestion system helping the reporters to select the best keywords while writing the issue.

- Regarding the UAV issues, in particular, we can summarize that a large percentage of mission and estimator issues result in catastrophic and critical issues. Moreover, compared to all internal bugs of Autopilot, the estimator is the origin of a considerable percentage of issues. Interestingly communication issues are the least frequent issues and, in most cases, have a negligible impact.
- All the information collected in this thesis is useful for the evaluation of the safety risk of fault/failures/threats in UAV systems, or for the realization of Failure Mode and Effect Analysis (FMEA) based on the knowledge of categories and severities extracted from automatic classifiers.

Chapter 5

Conclusion and future work

This chapter starts with the description of some of the most frequent failure modes found by experts during manual classification and an in-depth look at automatic analysis. The most critical and catastrophic UAV failures are presented here divided by category, since the most negligible ones are still relevant but do not jeopardize the safety of the system and the environment. Below are listed all the threats to the validity of the work that have been identified during the evolution of the work and analyzing the results obtained. Furthermore, potential future works aimed at improving the proposed work are presented and the potential of this methodology in the scientific field is described, together with the final conclusions.

5.1 Common detected Failure

The descriptions of the problems reported on GitHub (in particular the more detailed ones) also present subsections in which the expected behavior of the drone is explained. The expected behavior is optimal safety behavior that the drone should follow in the event of a breakdown. For each of the following failure modes, the failsafe behavior reported by users is also presented, which in the problems encountered were not always performed by the drone.

5.1.1 Autopilot Failure

Waypoints and Flight Modes

Many PX4 piloted UAV missions involve waypoint navigation, which is pre-programmed or real-time planning and where different points along the trajectory are assigned. In this type of navigation there are also changes in the flight modes that can occur automatically during the flight or also be pre-programmed. An autopilot failure mode is an electronics

failure that occurs if the UAV does not respect the navigation schedule: it does not respect assigned waypoints or expected flight modes. When in flight, an autopilot failure can have devastating impacts. Depending on the proximity of nearby assets, the failed UAV can collide with other UAVs, enter prohibited territories, or create a safety hazard for people in the vicinity.

Therefore, the optimal safety behavior for an autopilot failure scenario is to first attempt to take manual control. If this is not feasible, the UAV should immediately reduce altitude and land.

5.1.2 Estimator Failure

Propulsion system

As explained in the introductory chapter UAVs can be powered by gas or by on-board batteries, and in both cases due to the nature of the propulsion system, UAVs are limited by the inherent autonomy of their fuel source. During the flight the range of the remaining autonomy is always evaluated, but sometimes it happens that the expected value and the one evaluated in flight do not coincide. When the autonomy is less than expected, the point where there is enough fuel to land is estimated and the drone starts the landing phase.

Preventive landing caused by a failure mode of the propulsion system should be evaluated as an anomalous event. Furthermore, the assumption in a preventive landing situation is that the autonomous system still has control over the flight operations. In the event that control is not established, however, various catastrophic events can occur such as the explosion in the air or the crash on the ground of the drone due to the engines that stop working.

Sensors

All drone sensors are sensitive subsystems that must be calibrated and compensated in order to provide a correct estimate of all the information necessary for the drone's flight, such as position and orientation. From the reported problems there are many cases of incorrect estimation, and in most of these cases the wrong estimated attitude causes a drift in the roll angle (also in pitch, but not so much), even if the pose desired by the pilot was in a different state. The safety behavior foresees that the pilot, using the manual controls, compensates the position of the drone by tilting it in the opposite direction. In the event that manual controls are not established, dangerous events such as the crash of the drone can occur.

Global Positioning System (GPS)

GPS provides extremely accurate information on the location of the UAV. When a UAV is unable to use GPS, it loses the ability to know its position in the sky based on global coordinates, resulting in inaccurate positioning which can be critical when flying over dense urban terrain and other densely populated areas. In the event that the speed and direction of the UAV are not accompanied by an accurate position estimate by the GPS, the UAV can quickly falter in high risk territories. If the connection with is re-established, the UAV should turn off the throttle, forcing a crash. Instead, if the GPS connection is re-established, the UAV should immediately go to the return to launch command.

5.1.3 Communication Failure

Ground Control Station (GCS)

Wireless transmissions to transmit commands and navigation information in drones are fragile connections and are the primary connections between drone and GCS. Many issues report that transmissions have been interrupted for a certain period of time or completely lost and when this occurs, the GCS loses the ability to direct the flight of the drone.

As in all the cases already examined there are corrective behaviors that should be performed by the UAV system before considering the fatal error. In the case of a GCS link loss scenario, a recursive check for the leakage link occurs for a predetermined time interval of n minutes. If the time period expires and the GCS link has not re-established the connection, the error is deemed effective and security behaviors such as RTL or others are triggered (if possible).

5.1.4 Mission Failure

Geofence boundary

Among the basic rules of UAV flight there is one that requires missions to be bound by a perimeter of the flight area. This perimeter is known as a geofence, is usually defined by command and control communication limits, and limits the vehicle to fly in a certain area. A geofence breach occurs when a UAV exits the boundary of the predefined geofence, and this can have serious security implications, especially as it could result in the loss of communications. For this reason, a breach of the geofence must be dealt with immediately. Usually to avoid a total loss of communications, the area defined by the geofence is reduced compared to the calculated one, to provide a communication margin to try to redirect the drone within the area allowed for flight.

When a UAV ignores a directive to stay within a defined area, it is assumed that something rather serious is happening within the UAV's autopilot control (such as a hacker attack). Therefore, the default safety behavior for a geofence breach is to take

manual control of the aircraft and return it to the launch position. If manual control is not established, the next behavior is to kill the throttle causing a controlled crash.

5.2 Threats to Validity

Although promising results, working on this study and analyzing the outcomes step by step some limitations that threaten the validity of the results were obtained:

- Selection of dataset: Although it was included all reports with enough description for analysis, the size of the dataset classified manually and used for training the models is small, and the distribution of data is not well balanced.
- Validation of the Manual Classification: Validation of manual classification is limited to verifying the classifications by two experts involved in the manual classification. Although this eliminates some possible errors, it does not guarantee a full validation.
- Keywords extractions: it was noticed that, especially during the manual final validation, the extraction of the keywords did not select the words that give the seriousness of the issue or the context to which it referred.
- Validation of the Automatic Classification: To validate the generalization of the automatic classification model, we verified the consistency and checked the classification of a random sample (150 out of 490 reports) manually by two experts.

5.3 Future Works and Improvements

In parallel to the detection of some threats to the validity of the work, some possible improvements are identified.

- Selection of dataset: changing the selection criteria the dataset to be used to build the classifier models, can give more data and therefore make the classifier more reliable and improve its metrics.
- Validation of the Manual Classification: natural language processing methodologies and techniques for that dataset can be used to eliminate the initial manual analysis, reducing the time spent in it and will also reduce ambiguity in the training data.
- Keywords extraction: Using custom algorithms for keyword extraction can improve model performance.
- Validation of the Automatic Classification: manually verifying the 490 classified issues and not just a portion of these (150 data), the accuracy of the classifier on an external dataset can be more recognized.

The techniques proposed have the advantage of being applied to other datasets. In the context of UAV systems, these analyses can be performed (which is an event streaming platform used by the autonomous drones and drones management systems), FastDDS (is a middleware for embedded architectures) and Mavlink (a messaging protocol used by autonomous drone systems), also for the other open-source autopilots like Ardupilot. This helps to identify new causes of failure and the relative frequency in the various systems related to the UAV, allowing an increasingly in-depth study and guaranteeing a more precise risk assessment, preventing any risks and damage to the environment, to the drone and to humans.

5.4 Conclusion

The work proposes a method to perform the safety risk assessment of unmanned aerial vehicles (UAVs), in particular to identify the types and frequency of problems involving UAVs. an experimental study was presented on the automatic classification of the severity and nature of the reported problems for the PX4 flight controller software present in a dedicated repository on Github. A total of 610 issues were extracted from the 17,162 available on GitHub for this software, with enough information (comments etc.) to be able to manually categorize them into two dimensions: error categories and error severity. This dataset is then used to train different machine learning models, and the relative effectiveness of these models was evaluated to rank other problems chosen from those not used in training. The method involves: i) extraction and manual classification of data from available reports; ii) generation of models trained to classify problems; iii) evaluation of models trained with other data sets.

Four different classifiers are tested and Decision Tree is identified as the best performing algorithm in the context of this data type, providing respectively 93.43 % and 91.14 % *Accuracy* on the error category and gravity. It also shows relatively high performance on unknown data sets giving 70% and 68.67% *Accuracy* on severity and error category respectively.

These results confirm the possibility of machine learning algorithms to automate the classification process of bug reports written in natural languages, thus allowing to speed up the process of identifying the risk category, severity and frequency of occurrence in very large field datasets.

The application of this methodology on the proposed systems would help in the field of risk analysis to increase the search for the most fragile components of UAS and define solutions to avoid any type of damage to the system, to the environment and to humans.

Bibliography

[1] Ram Gopal Lakshmi Narayanan and Oliver C Ibe. «Joint network for disaster relief and search and rescue network operations». In: *Wireless Public Safety Networks 1*. Elsevier, 2015, pp. 163–193 (cit. on pp. 1, 10).

[2] G Pradeep Kumar and B Sridevi. «Development of Efficient Swarm Intelligence Algorithm for Simulating Two-Dimensional Orthomosaic for Terrain Mapping Using Cooperative Unmanned Aerial Vehicles». In: *The Cognitive Approach in Cloud Computing and Internet of Things Technologies for Surveillance Tracking Systems*. Elsevier, 2020, pp. 75–93 (cit. on pp. 1, 8).

[3] Divya Joshi. *Drone technology uses and applications for commercial, industrial and military drones in 2020 and the future*. URL: `https://www.businessinsider.com/drone-technology-uses-applications`. (accessed: 10.01.2021) (cit. on pp. 1, 8).

[4] G Morgenthal and N Hallermann. «Quality assessment of unmanned aerial vehicle (UAV) based visual inspection of structures». In: *Advances in Structural Engineering* 17.3 (2014), pp. 289–302 (cit. on pp. 1, 8).

[5] Imad Jawhar, Nader Mohamed, Jameela Al-Jaroodi, Dharma P Agrawal, and Sheng Zhang. «Communication and networking of UAV-based systems: Classification and associated architectures». In: *Journal of Network and Computer Applications* 84 (2017), pp. 93–108 (cit. on pp. 1, 8).

[6] Joshua Bateman. «China Drone Maker DJI: Alone atop the Unmanned Skies». In: *CNBC, September* 1 (2017) (cit. on p. 1).

[7] S French. *DJI market share: here's exactly how rapidly it has grown in just a few years*. 2018 (cit. on p. 1).

[8] Ahmad Yazdan Javaid. «Cyber security threat analysis and attack simulation for unmanned aerial vehicle network». PhD thesis. University of Toledo, 2015 (cit. on p. 2).

[9] Paul R Nesbit, Thomas E Barchyn, Chris H Hugenholtz, Sterling Cripps, and Maja Kucharczyk. «Reported UAV incidents in Canada: analysis and potential solutions». In: *Journal of unmanned vehicle systems* 5.2 (2017), pp. 51–61 (cit. on p. 2).

[10] Abhishek Sharma, Pankhuri Vanjani, Nikhil Paliwal, Chathuranga M Wijerathna Basnayaka, Dushantha Nalin K Jayakody, Hwang-Cheng Wang, and P Muthuchidambaranathan. «Communication and networking technologies for UAVs: A survey». In: *Journal of Network and Computer Applications* (2020), p. 102739 (cit. on p. 2).

[11] Astrid Gynnild and Turo Uskali. «The first wave of drone journalism: From activist tool to global game changer». In: *Responsible drone journalism*. Routledge, 2018, pp. 15–35 (cit. on p. 2).

[12] Mehrdad Balali and Michelle Moghtader. *Iran says it shoots down Israeli spy drone*. 2014 (cit. on p. 3).

[13] Algirdas Avizienis. «Fault-tolerance: The survival attribute of digital systems». In: *Proceedings of the IEEE* 66.10 (1978), pp. 1109–1125 (cit. on p. 4).

[14] Janick Edinger, Dominik Schäfer, Christian Krupitzer, Vaskar Raychoudhury, and Christian Becker. «Fault-avoidance strategies for context-aware schedulers in pervasive computing systems». In: *2017 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE. 2017, pp. 79–88 (cit. on p. 4).

[15] Brian P Tice. «Unmanned aerial vehicles: The force multiplier of the 1990s». In: *Airpower Journal* 5.1 (1991), pp. 41–55 (cit. on p. 8).

[16] Sven Eriksson and Maths Lundin. «The drone market in Japan». In: *EU-Japan Centre for Industrial Cooperation* (2016) (cit. on p. 9).

[17] Balmukund Mishra, Deepak Garg, Pratik Narang, and Vipul Mishra. «Drone-surveillance for search and rescue in natural disaster». In: *Computer Communications* 156 (2020), pp. 1–10 (cit. on p. 9).

[18] Sudipta Chowdhury, Adindu Emelogu, Mohammad Marufuzzaman, Sarah G Nurre, and Linkan Bian. «Drones for disaster response and relief operations: A continuous approximation model». In: *International Journal of Production Economics* 188 (2017), pp. 167–184 (cit. on p. 9).

[19] G Gabriel, Miguel A Ramallo, and Elizabeth Cervantes. «Workload perception in drone flight training simulators». In: *Computers in Human Behavior* 64 (2016), pp. 449–454 (cit. on p. 9).

[20] Mostafa Hassanalian and Abdessattar Abdelkefi. «Classifications, applications, and design challenges of drones: A review». In: *Progress in Aerospace Sciences* 91 (2017), pp. 99–131 (cit. on p. 9).

[21] Adam C Watts, Vincent G Ambrosia, and Everett A Hinkley. «Unmanned aircraft systems in remote sensing and scientific research: Classification and considerations of use». In: *Remote Sensing* 4.6 (2012), pp. 1671–1692 (cit. on p. 9).

[22] Maziar Arjomandi, Shane Agostino, Matthew Mammone, Matthieu Nelson, and Tong Zhou. «Classification of unmanned aerial vehicles». In: *Report for Mechanical Engineering class, University of Adelaide, Adelaide, Australia* (2006) (cit. on p. 9).

[23] Prem Mahadevan. «The military utility of drones». In: *CSS Analyses in Security Policy* 78 (2010) (cit. on p. 9).

[24] Federal Aviation Administration. *Unmanned Aircraft Systems (UAS)*. URL: `https://www.faa.gov/uas/`. (accessed: 22.02.2021) (cit. on p. 10).

[25] Darren Lance Gabriel, Johan Meyer, and Francois Du Plessis. «Brushless DC motor characterisation and selection for a fixed wing UAV». In: *IEEE Africon'11*. IEEE. 2011, pp. 1–6 (cit. on p. 13).

[26] Miroslaw Adamski. «Analysis of propulsion systems of unmanned aerial vehicles». In: *Journal of Marine Engineering & Technology* 16.4 (2017), pp. 291–297 (cit. on p. 13).

[27] Paul D Groves. «Principles of GNSS, inertial, and multisensor integrated navigation systems, [Book review]». In: *IEEE Aerospace and Electronic Systems Magazine* 30.2 (2015), pp. 26–27 (cit. on p. 13).

[28] Azza Allouch, Omar Cheikhrouhou, Anis Koubâa, Mohamed Khalgui, and Tarek Abbes. «MAVSec: Securing the MAVLink protocol for ardupilot/PX4 unmanned aerial systems». In: *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. IEEE. 2019, pp. 621–628 (cit. on p. 14).

[29] Tatjana Bolić and Paul Ravenhill. «SESAR: The Past, Present, and Future of European Air Traffic Management Research». In: *Engineering* 7.4 (2021), pp. 448–451 (cit. on pp. 16, 17).

[30] Michael Standar. «SESAR and the european ATM Master plan-Cooperation with the US/NextGen». In: *2012 Integrated Communications, Navigation and Surveillance Conference*. IEEE. 2012, pp. 1–34 (cit. on p. 16).

[31] Charu C Aggarwal. *Machine learning for text*. Springer, 2018 (cit. on pp. 20–22, 24, 26).

[32] Andrius Vabalas, Emma Gowen, Ellen Poliakoff, and Alexander J Casson. «Machine learning algorithm validation with a limited sample size». In: *PloS one* 14.11 (2019), e0224365 (cit. on p. 22).

[33] Sylvain Arlot and Alain Celisse. «A survey of cross-validation procedures for model selection». In: *Statistics surveys* 4 (2010), pp. 40–79 (cit. on p. 22).

[34] ccohs.ca. *Risk Assessment*. URL: `https://www.ccohs.ca/oshanswers/hsprograms/risk_assessment.html`. (accessed: 16.02.2021) (cit. on p. 31).

[35] Reece Clothier and Rodney Walker. «Determination and evaluation of UAV safety objectives». In: *Proceedings of the 21st International Conference on Unmanned Air Vehicle Systems*. University of Bristol. 2006, pp. 18–1 (cit. on p. 31).

[36] Riham Altawy and Amr M Youssef. «Security, privacy, and safety aspects of civilian drones: A survey». In: *ACM Transactions on Cyber-Physical Systems* 1.2 (2016), pp. 1–25 (cit. on p. 31).

[37] Jesse Rawlins Paterson, Jiwoong Han, Tom Cheng, Paxtan Huish Laker, David Livingston McPherson, Joseph Menke, and Allen Y Yang. «Improving usability, efficiency, and safety of UAV path planning through a virtual reality interface». In: *Symposium on Spatial User Interaction*. 2019, pp. 1–2 (cit. on p. 31).

[38] Kay Wackwitz and Hendrick Boedecker. «Safety risk assessment for uav operation». In: *Drone Industry Insights, Safe Airspace Integration Project, Part One, Hamburg, Germany* (2015) (cit. on pp. 31, 53).

[39] Lawrence C Barr, Richard Newman, Ersin Ancel, Christine M Belcastro, John V Foster, Joni Evans, and David H Klyde. «Preliminary risk assessment for small unmanned aircraft systems». In: *17th AIAA Aviation Technology, Integration, and Operations Conference*. 2017, p. 3272 (cit. on p. 32).

[40] Azza Allouch, Anis Koubâa, Mohamed Khalgui, and Tarek Abbes. «Qualitative and quantitative risk analysis and safety assessment of unmanned aerial vehicles missions over the internet». In: *IEEE Access* 7 (2019), pp. 53392–53410 (cit. on pp. 32, 53).

[41] jarus-rpas.org. *SORA*. URL: `http://jarus-rpas.org/sites/jarus-rpas.org/files/jar_doc_06_jarus_sora_v2.0.pdf` (cit. on p. 32).

[42] eurocontrol. *MEDUSA*. URL: `https://www.eurocontrol.int/sites/default/files/2019-10/09-safety_0.pdf` (cit. on p. 32).

[43] Giorgio Guglieri, F Quagliotti, and Gianluca Ristorto. «Operational issues and assessment of risk for light UAVs». In: *Journal of Unmanned Vehicle Systems* 2.4 (2014), pp. 119–129 (cit. on p. 32).

[44] Georg Macher, Eric Armengaud, Eugen Brenner, and Christian Kreiner. «Threat and risk assessment methodologies in the automotive domain». In: *Procedia computer science* 83 (2016), pp. 1288–1294 (cit. on p. 32).

[45] Kim Hartmann and Christoph Steup. «The vulnerability of UAVs to cyber attacks-An approach to the risk assessment». In: *2013 5th international conference on cyber conflict (CYCON 2013)*. IEEE. 2013, pp. 1–23 (cit. on p. 32).

[46] Hamid Reza Feili, Navid Akar, Hossein Lotfizadeh, Mohammad Bairampour, and Sina Nasiri. «Risk analysis of geothermal power plants using Failure Modes and Effects Analysis (FMEA) technique». In: *Energy Conversion and Management* 72 (2013), pp. 69–76 (cit. on pp. 33, 34).

[47] Javier Puente, Raúl Pino, Paolo Priore, and David de la Fuente. «A decision support system for applying failure mode and effects analysis». In: *International Journal of Quality & Reliability Management* (2002) (cit. on p. 33).

[48] Ping C Teoh and Keith Case. «Failure modes and effects analysis through knowledge modelling». In: *Journal of Materials Processing Technology* 153 (2004), pp. 253–260 (cit. on p. 33).

[49] BG Dale and Peter Shaw. «Failure mode and effects analysis in the UK motor industry: A state-of-the-art study». In: *Quality and Reliability Engineering International* 6.3 (1990), pp. 179–188 (cit. on p. 33).

[50] P G Hawkins and David J Woollons. «Failure modes and effects analysis of complex engineering systems using functional models». In: *Artificial intelligence in engineering* 12.4 (1998), pp. 375–397 (cit. on p. 33).

[51] Yiannis Papadopoulos, David Parker, and Christian Grante. «Automating the failure modes and effects analysis of safety critical systems». In: *Eighth IEEE International Symposium on High Assurance Systems Engineering, 2004. Proceedings.* IEEE. 2004, pp. 310–311 (cit. on p. 34).

[52] Hooman Arabian-Hoseynabadi, Hashem Oraee, and PJ Tavner. «Failure modes and effects analysis (FMEA) for wind turbines». In: *International Journal of Electrical Power & Energy Systems* 32.7 (2010), pp. 817–824 (cit. on p. 34).

[53] Z Mazur, R Garcıa-Illescas, and J Porcayo-Calderón. «Last stage blades failure analysis of a 28 MW geothermal turbine». In: *Engineering Failure Analysis* 16.4 (2009), pp. 1020–1032 (cit. on p. 34).

[54] Husain Aljazzar, Manuel Fischer, Lars Grunske, Matthias Kuntz, Florian Leitner-Fischer, and Stefan Leue. «Safety analysis of an airbag system using probabilistic FMEA and probabilistic counterexamples». In: *2009 Sixth International Conference on the Quantitative Evaluation of Systems*. IEEE. 2009, pp. 299–308 (cit. on p. 34).

[55] Swapnil B Ambekar, Ajinkya Edlabadkar, and Vivek Shrouty. «A review: implementation of Failure Mode and Effect Analysis». In: *International Journal of Engineering and Innovative Technology (IJEIT)* 2.8 (2013), pp. 37–41 (cit. on p. 34).

[56] Michael B Revill. *UAV swarm behavior modeling for early exposure of failure modes*. Tech. rep. Naval Postgraduate School Monterey United States, 2016 (cit. on p. 34).

[57] Jiufeng Wu, Tao Yang, and Xinlei Cao. «A Typical Failure Mode and Effects Analysis for EMA Used for UAV and Guided Missile». In: *Journal of Physics: Conference Series*. Vol. 1815. 1. IOP Publishing. 2021, p. 012043 (cit. on p. 34).

[58] Jie Zhang, Xiaoyin Wang, Dan Hao, Bing Xie, Lu Zhang, and Hong Mei. «A survey on bug-report analysis». In: *Science China Information Sciences* 58.2 (2015), pp. 1–24 (cit. on p. 35).

[59] Anh Tuan Nguyen, Tung Thanh Nguyen, Jafar Al-Kofahi, Hung Viet Nguyen, and Tien N Nguyen. «A topic-based approach for narrowing the search space of buggy files from a bug report». In: *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE. 2011, pp. 263–272 (cit. on p. 35).

[60] Tao Xie, Lu Zhang, Xusheng Xiao, Ying-Fei Xiong, and Dan Hao. «Cooperative software testing and analysis: Advances and challenges». In: *Journal of Computer Science and Technology* 29.4 (2014), pp. 713–723 (cit. on p. 35).

[61] Wenjin Wu, Wen Zhang, Ye Yang, and Qing Wang. «Time series analysis for bug number prediction». In: *The 2nd International Conference on Software Engineering and Data Mining*. IEEE. 2010, pp. 589–596 (cit. on p. 35).

[62] Sandeep Singh. «Analysis of bug tracking tools». In: *International Journal of Scientific & Engineering Research* 4.7 (2013), pp. 134–140 (cit. on p. 35).

[63] Marco Torchiano and Filippo Ricca. «Impact analysis by means of unstructured knowledge in the context of bug repositories». In: *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. 2010, pp. 1–4 (cit. on p. 35).

[64] Pamela Bhattacharya, Liudmila Ulanova, Iulian Neamtiu, and Sai Charan Koduru. «An empirical analysis of bug reports and bug fixing in open source android apps». In: *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE. 2013, pp. 133–143 (cit. on p. 35).

[65] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. «A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction». In: *Proceedings of the 30th international conference on Software engineering*. 2008, pp. 181–190 (cit. on p. 35).

[66] Zeinab Abou Khalil, Eleni Constantinou, Tom Mens, Laurence Duchien, and Clément Quinton. «A longitudinal analysis of bug handling across Eclipse releases». In: *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2019, pp. 1–12 (cit. on p. 36).

[67] Omar El Zarif, Daniel Alencar Da Costa, Safwat Hassan, and Ying Zou. «On the Relationship between User Churn and Software Issues». In: *Proceedings of the 17th International Conference on Mining Software Repositories*. 2020, pp. 339–349 (cit. on p. 36).

[68] Bora Caglayan and Ayse Bener. «Issue ownership activity in two large software projects». In: *ACM SIGSOFT Software Engineering Notes* 37.6 (2012), pp. 1–7 (cit. on p. 36).

[69] Giuseppe Destefanis, Marco Ortu, David Bowes, Michele Marchesi, and Roberto Tonelli. «On measuring affects of github issues' commenters». In: *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*. 2018, pp. 14–19 (cit. on p. 36).

[70] Casimiro Conde Marco Neto and Márcio de O. Barros. «A structured survey on the usage of the issue tracking system provided by the github platform». In: *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse*. 2017, pp. 1–10 (cit. on p. 36).

[71] Akash Balasaheb Dhasade, Akhila Sri Manasa Venigalla, and Sridhar Chimalakonda. «Towards prioritizing github issues». In: *Proceedings of the 13th Innovations in Software Engineering Conference on Formerly known as India Software Engineering Conference*. 2020, pp. 1–5 (cit. on p. 36).

[72] Imet Ristemi, Marika Apostolova Trpkovska, and Betim Cico. «Mygitissues web application as a solution in dealing with issues on github». In: *2019 8th Mediterranean Conference on Embedded Computing (MECO)*. IEEE. 2019, pp. 1–4 (cit. on p. 36).

[73] Qiang Fan, Yue Yu, Gang Yin, Tao Wang, and Huaimin Wang. «Where is the road for issue reports classification based on text mining?» In: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. 2017, pp. 121–130 (cit. on p. 36).

[74] Chelsea Hinds-Charles, Jenelee Adames, Ye Yang, Yusong Shen, and Yong Wang. «A Longitude Analysis on Bitcoin Issue Repository». In: *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*. IEEE. 2018, pp. 212–217 (cit. on p. 36).

[75] Kabita Thaoroijam. «A study on document classification using machine learning techniques». In: *International Journal of Computer Science Issues (IJCSI)* 11.2 (2014), p. 217 (cit. on p. 37).

[76] Wei-Hung Weng, Kavishwar B Wagholikar, Alexa T McCray, Peter Szolovits, and Henry C Chueh. «Medical subdomain classification of clinical notes using a machine learning-based natural language processing approach». In: *BMC medical informatics and decision making* 17.1 (2017), pp. 1–13 (cit. on p. 37).

[77] Ravi Garg, Elissa Oh, Andrew Naidech, Konrad Kording, and Shyam Prabhakaran. «Automating ischemic stroke subtype classification using machine learning and natural language processing». In: *Journal of Stroke and Cerebrovascular Diseases* 28.7 (2019), pp. 2045–2051 (cit. on p. 37).

[78] Andrea Caccamisi, Leif Jørgensen, Hercules Dalianis, and Mats Rosenlund. «Natural language processing and machine learning to enable automatic extraction and classification of patients' smoking status from electronic medical records». In: *Upsala Journal of Medical Sciences* 125.4 (2020), pp. 316–324 (cit. on p. 37).

[79] Tran Thanh Dien, Bui Huu Loc, and Nguyen Thai-Nghe. «Article classification using natural language processing and machine learning». In: *2019 International Conference on Advanced Computing and Applications (ACOMP)*. IEEE. 2019, pp. 78–84 (cit. on p. 37).

[80] Guido Perboli, Marco Gajetti, Stanislav Fedorov, and Simona Lo Giudice. «Natural Language Processing for the identification of Human factors in aviation accidents causes: An application to the SHEL methodology». In: *Expert Systems with Applications* 186 (2021), p. 115694 (cit. on p. 37).

[81] Filippo Lanubile, Christof Ebert, Rafael Prikladnicki, and Aurora Vizcaíno. «Collaboration Tools for Global Software Engineering». In: *IEEE Software* 27.2 (2010), pp. 52–55. DOI: 10.1109/MS.2010.39 (cit. on p. 39).

[82] softwaretestinghelp.com. *15 BEST Version Control Software (Source Code Management Tools)*. URL: https://www.softwaretestinghelp.com/version-control-software/. (accessed: 29.04.2021) (cit. on p. 39).

[83] fullscale.io. *Top 10 Version Control Systems*. URL: https://fullscale.io/blog/top-10-version-control-systems/. (accessed: 29.04.2021) (cit. on p. 39).

[84] hackernoon.com. *Top 10 Version Control Systems*. URL: https://hackernoon.com/top-10-version-control-systems-4d314cf7adea. (accessed: 29.04.2021) (cit. on p. 39).

[85] Jan Ploski, Matthias Rohr, Peter Schwenkenberg, and Wilhelm Hasselbring. «Research Issues in Software Fault Categorization». In: *SIGSOFT Softw. Eng. Notes* 32.6 (Nov. 2007), 6–es. ISSN: 0163-5948. DOI: 10.1145/1317471.1317478. URL: https://doi.org/10.1145/1317471.1317478 (cit. on p. 39).

[86] Priya Pedamkar. *Advantages of Matlab*. URL: https://www.educba.com/advantages-of-matlab/. (accessed: 29.04.2021) (cit. on p. 41).

[87] Jisheng Liang, Krzysztof Koperski, Thien Nguyen, and Giovanni Marchisio. «Extracting Statistical Data Frames from Text». In: *SIGKDD Explor. Newsl.* 7.1 (June 2005), pp. 67–75. ISSN: 1931-0145. DOI: 10.1145/1089815.1089825. URL: https://doi.org/10.1145/1089815.1089825 (cit. on p. 41).

[88] Rada Mihalcea and Paul Tarau. «Textrank: Bringing order into text». In: *Proceedings of the 2004 conference on empirical methods in natural language processing*. 2004, pp. 404–411 (cit. on p. 47).

[89] Slav Petrov, Dipanjan Das, and Ryan McDonald. «A universal part-of-speech tagset». In: *arXiv preprint arXiv:1104.2086* (2011) (cit. on p. 50).

[90] John A Bullinaria and Joseph P Levy. «Extracting semantic representations from word co-occurrence statistics: A computational study». In: *Behavior research methods* 39.3 (2007), pp. 510–526 (cit. on p. 50).

[91] Fan Chung. «A brief survey of PageRank algorithms». In: *IEEE Transactions on Network Science and Engineering* 1.01 (2014), pp. 38–42 (cit. on p. 51).

[92] Harrison G Wolf. *Drones: Safety risk management for the next evolution of flight*. Taylor & Francis, 2017 (cit. on p. 52).

[93] Dániel Szöllősi, Dénes Lajos Dénes, Ferenc Firtha, Zoltán Kovács, and András Fekete. «Comparison of six multiclass classifiers by the use of different classification performance indicators». In: *Journal of Chemometrics* 26.3-4 (2012), pp. 76–84 (cit. on p. 56).

[94] Sundus Hassan, Muhammad Rafi, and Muhammad Shahid Shaikh. «Comparing SVM and Naıve Bayes classifiers for text categorization with Wikitology as knowledge enrichment». In: *2011 IEEE 14th International Multitopic Conference*. IEEE. 2011, pp. 31–34 (cit. on p. 56).

[95] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. «A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches». In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.4 (2011), pp. 463–484 (cit. on p. 66).