INCIDENT ROUTING: TEXT CLASSIFICATION, FEATURE SELECTION, IMBALANCED DATASETS, AND CONCEPT DRIFT IN INCIDENT TICKET MANAGEMENT

Matheus Correia Ferreira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Geraldo Bonorino Xexéo

Rio de Janeiro

Dezembro de 2017

INCIDENT ROUTING: TEXT CLASSIFICATION, FEATURE SELECTION, IMBALANCED DATASETS, AND CONCEPT DRIFT IN INCIDENT TICKET MANAGEMENT

Matheus Correia Ferreira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Geraldo Bonorino Xexéo, D.Sc.

Prof. Geraldo Zimbrão da Silva, D.Sc.

Prof. Leandro Guimarães Marques Alvim, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
DEZEMBRO DE 2017

Ferreira, Matheus Correia

Incident Routing: Text Classification, Feature Selection, Imbalanced Datasets, and Concept Drift in Incident Ticket Management / Matheus Correia Ferreira. – Rio de Janeiro: UFRJ/COPPE, 2017.

XIII, 144 p.: il.; 29,7 cm.

Orientador: Geraldo Bonorino Xexéo

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2017.

Referências Bibliográficas: p. 127-132.

1. Text Classification. 2. Feature Selection. 3. Imbalanced Datasets. 4. Concept Drift. 5. Incident Management. I. Xexéo, Geraldo Bonorino II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Acknowledgements

To my parents, Gloria and Luis, for raising me and teaching me the value of hard work and dedication; to my sister, Juliana; and to my family as a whole, for the healthy, loving, and supportive environment they have maintained through my life

To Fernanda and Patrícia, great masters in Computer Science and non-official advisors ever since before I started this research, for their ideas, help, and advice.

To my friends, for the moments of leisure they provided me with during these two years, and more, especially to: Andressa, Carina, Duda, Felipe, Isabel, Iuri, Malena, Marcelle, Marina, Mariana, Rafael, Renan, Sara, and William.

To my students and former students, some of which have grown into great friends; through their admiration during all these years, they have done for me more than they could possibly ever know. Thanks to all, and especially to: Ana Eduarda, Jayana, Júlia, Larissa, Letícia, Luana, Maria Eduarda, Maria Julia, Marcela, Natália, and Nathalia. You are all incredibly talented, and whatever your definition of success may be, I will always be rooting for you with total confidence that you will get there. Get out there and rock the world.

To my classmates with whom I worked during this course, especially to Anderson, Pedro, and Rosangela; and to Jones and Max, who collaborated with this research.

To the teachers I came into contact with at UFRJ, the knowledge they transmitted was certainly invaluable to this work. Thanks to Filipe, Jano, Nathália, Pedreira, and Zimbrão. To all the teachers I have had during my life. And a special thanks to my advisor during this research, Geraldo Xexéo, for the ideas, time, patience, and meetings.

To the people from work that made it easier to conciliate my professional and academic lives during this course: Anderson, Erico, Fábio, Lia, Marcos, and Michel.

Finally, to those who, while not essential to this work, certainly made the journey far more pleasant: to Shigeru Miyamoto, for creating some pretty awesome games; and to various musical masters, especially to Bob Dylan, for spitting out punchlines and wisdom at a faster rate than anybody is able to consume; to Bruce Springsteen, for singing about characters who were born to lose but that always feel like they are going to win; to Joe Strummer, for fighting relentlessly; to Neil Young, for doing whatever he wants whenever he pleases in whichever way he sees fit; and to Tom Waits, for being Tom Waits.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ROTEAMENTO DE INCIDENTES: CLASSIFICAÇÃO DE TEXTO, SELEÇÃO DE ATRIBUTOS, DATASETS DESBALANCEADOS, E CONCEPT DRIFT EM GESTÃO DE TICKETS DE INCIDENTES

Matheus Correia Ferreira

Dezembro/2017

Orientadores: Geraldo Bonorino Xexéo

Programa: Engenharia de Sistemas e Computação

Ao mesmo tempo em que a economia mundial entrou em um período onde serviços, não produtos, são o foco dos negócios, a tecnologia também avançou. Serviços de tecnologia da informação existem na interseção destas correntes, e a importância deles é clara dado o número de serviços disponíveis em plataformas eletrônicas. Companhias dependem deles para entregar valor aos seus clientes; governos os usam para prover serviços essenciais para a população; e usuários desfrutam dos mesmos para entretenimento, compras, e outras atividades. Provedores de serviços de TI têm o desafio de manter a infraestrutura tecnológica que suporta estes serviços. Uma vez que o mundo se tornou dependente de sistemas de TI, a queda de qualidade ou indisponibilidade destes não são boas para os negócios. Assim, a solução eficiente dos incidentes que causam estes problemas é um dos aspectos críticos da gestão de serviços de TI. Buscando reduzir a carga de trabalho envolvida no gerenciamento de incidentes, um sistema que designa incidentes registrados para as áreas que têm a expertise para resolvê-los é criado e testado com incidentes reais coletados de uma empresa brasileira provedora de serviços de TI. Para isto, técnicas de classificação automática de texto, e problemas de maldição da dimensionalidade e datasets desbalanceados são analisados. Além disso, concept drift, que ocorre conforme a distribuição dos dados muda com o tempo, também é tratado com um algoritmo tradicional da área e um grupo de ensembles propostos que trazem melhores resultados na redução da carga de trabalho e acurácia de classificação.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

INCIDENT ROUTING: TEXT CLASSIFICATION, FEATURE SELECTION, IMBALANCED DATASETS, AND CONCEPT DRIFT IN INCIDENT TICKET MANAGEMENT

Matheus Correia Ferreira

December/2017

Advisors: Geraldo Bonorino Xexéo

Department: Systems and Computer Engineering

While the world's economy has entered a period where services, not products, are the focus of businesses, technology has also advanced. Information technology services exist in the intersection of these two currents and their importance is clear given the number of services that are available on numerous electronic platforms. Companies depend on those applications to deliver value to customers; governments use them to provide the population with essential services; and many users rely on them for entertainment, shopping, and other activities. IT service providers must contend with the challenge of keeping the technological infrastructure that supports services running. As the world has grown to be dependent on IT systems, the downtime or drop in quality of such services is bad for business. Therefore, the efficient solving of the incidents that cause those problems is one of the critical aspects of IT service management. In this work, with the goal of reducing the workload involved in the management of incidents, a system that automatically assigns registered incidents to the areas that have the expertise to solve them is designed and tested with real-life incidents collected from a Brazilian IT service provider. To do so, techniques related to automatic text classification are employed, and problems related to the curse of dimensionality and imbalanced datasets are approached. Moreover, concept drift, which occurs as the data distribution of tickets changes overtime, is also handled via a traditional algorithm of the area and a group of proposed ensembles that bring better results in terms of workload reduction and classification accuracy.

Summary

List of Figures

List of Tables

List of Formulas

Chapter 1

Introduction

1.1 Contextualization

As technology advances and permeates all layers of society, the role of Information Technology (IT) grows for companies, governments, and the general population alike. In the business field, private companies in the United States spend over half of their invested money on IT (MARRONE *et al.*, 2014). In the public sector, meanwhile, governments have, in recent years, been increasing their online presence and the extent of services they offer through that channel; in a research conducted by the United Nation's Department of Economic and Social Affairs, out of the 193 nations recognized by the organization, 101 allow their citizens to create personal online accounts, 73 support the submission of income taxes over the Internet, and 60 execute the process of registering a business through virtual means (UNITED NATIONS, 2014). At the other end of the spectrum, where users lie, it has been reported that 92% of US adults own a mobile phone of some kind, while 73% of that same group have either laptop or desktop computers, and 45% have tablets (ANDERSON, 2015).

Concurrently with the widening of the grasp of IT, the world has globally entered a period in which services have become the economy's main driving force. Not only are they the largest source of employment in both developed and developing countries, but services also constitute the largest share of GDP in most nations around the globe (DAHLGAARD-PARK, 2015). This post-industrial scenario has been dubbed the service economy, and it has differed from the goods-production economy of the industrial society through three criteria (ZHOU, 2015). For starters, the biggest portion of the working force is engaged in service activities such as trade, finance, transportation, health care, research, entertainment, and others rather than in tasks whose focus is sheer production, like manufacturing and agriculture; additionally, professionals and knowledge have gained prominence over non-specialized individuals and manual power. As a consequence, companies involved in the service economy tend not to see products as the only assets

they deliver to customers; the focus, instead, shifts from producing goods and features to creating value and positive experiences (OSTERWALDER *et al.*, 2014).

In the intersection between the growth of information technology and the rise of the service economy, IT services appear. Given they exist in the overlapping area of these two phenomena, IT services are rather valuable in the world's current state, and since the tools used to access them (computers, cellphones, and tablets) are in the hands and homes of many, they are widely used. Retailers, such as Amazon and Walmart, execute many online sales transactions; in fact, eight-in-ten Americans are online shoppers (SMITH; ANDERSON, 2016). Banks and other financial institutions allow their customers to perform various types of operations via electronic devices, and such a method of finance managing has become so popular that while the number of people who visit their respective local bank branches diminishes the payments done via apps soar (JONES, 2016). Enterprises as Google, Apple, and Microsoft offer users the opportunity to store personal files (pictures, music, documents, etc.) on external servers located inside data centers maintained by those companies; and cloud storage is so vastly used that, in 2016, it surpassed file-sharing as the largest source of upstream traffic (that is, data that flows from computers to the network) during peak period on North American fixed access networks (SANDVINE, 2016). Streaming services like Netflix, Youtube, Spotify, and Steam hold vast catalogs of videos, music, and games that can be accessed by customers using different platforms, and the streaming of audio and video accounts for 71% of evening network traffic in North America (SANDVINE, 2016). Companies across multiple areas of the economy employ electronic tools and applications to produce and deliver value to their customers. And governments around the planet make essential services available to their citizens on the Internet.

In those cases, and in some others as well, the systems that allow consumers to perform those activities rely on the underlying technological infrastructure (such as servers, routers, and others) that supports them. The poor maintenance or management of that infrastructure can cause undesirable occurrences – which can range from punctual errors and service slowdowns to complete unavailability – to take place. Within the scope of an economic setting of high competition and where, thanks to the Internet, users can freely choose from an expansive deck of options regardless of the IT service in question, and considering the complexity of the technological environments which exist behind

most IT services, cost-effective service delivery and support are essential (ZHOU *et al.*, 2015).

Infrastructure-related problems can cause unsatisfied and frustrated customers to stop paying or consuming what is provided by one company and migrate to rival platforms. Therefore, it becomes key to find ways to both reduce the rates with which those issues appear and to quickly act upon them when they occur so that their effects are either minimal or unperceived, for, due to the world's considerable reliance on technology, service downtime is negatively perceived: it affects businesses, causes users to be unhappy, and can decrease the quality of the image of a company when compared to that of its competitors.

With the goal of establishing a series of practices to guarantee that IT services deliver value to the business, ITIL (Information Technology Infrastructure Library) was created (ITIL, 2011). In order to align what the company aims to perform with its technological infrastructure (in other words, its business needs) with the IT services it will implement, the library contains a body of knowledge that maps out the entire iterative life cycle of such a product; consequently, it is broken into five distinct pieces that define processes and functions that, if deployed correctly, will help IT companies achieve their goals and deliver quality services to customers. The parts that make up ITIL mirror the different phases a service goes through, following the path from the definition of the strategies that surround it (Service Strategy); going through its creation (Service Design), construction and deployment (Service Transition), and delivery (Service Operation); and maintaining a constant cycle of continuous improvement after the service is already in place (Continual Service Improvement).

Research on the usage of ITIL (IDEN; EIKEBROKK, 2013) reveals that managers and companies adopt it in order to improve: operational efficiency, focus on service delivery, the alignment between business and IT, service quality, and customer satisfaction. In particular, an exploratory study (POLLARD; CATER-STEEL, 2009) conducted with Australian and American companies that had successfully implemented ITIL showed the trigger towards that transition came from crises that took place due to failures in their infrastructure.

ITIL provides the knowledge to achieve the goal of avoiding, or reducing, infrastructural crises in one of its volumes: Service Operation, which establishes processes and functions that work towards delivering to customers the agreed levels of

service. Within the scope of the management of a company's technological infrastructure, incidents are defined as unexpected events that can cause the services that are supported by it to suffer losses in quality (ITIL, 2011). Inside Service Operation, incidents are handled by the Incident Management process and the Service Desk function. The former defines a series of steps for detecting, recording, classifying, and managing the incident throughout its life cycle. Meanwhile, the latter serves as the entry point for all incidents. service desk members are, therefore, contacted by users who are experiencing problems. Following that, it is the responsibility of the service desk to create an incident ticket by logging the issue into a system and describing the symptoms of the reported incident; and to either try to solve the ticket or choose to direct it to other areas (or service desk levels) when the incident proves to be too complex, requiring treatment by a more specialized team.

Due to how critical incidents are in the context of IT services, many applications exist or have been proposed to tackle the tasks that constitute Incident Management to make it more efficient. Automatic incident detection is done through system monitoring software (ZHOU *et al.*, 2015), which registers anomalous behaviors in the infrastructure and reports technical warnings regarding issues that may be happening. The next step, which is that of making sure that the incident tickets be directed to the technical departments that are capable of solving them, has been approached through the use of text analysis and automatic classification approaches (MOHARRERI *et al.*, 2016b). Finally, the process of ticket resolution has been optimized through systems that recommend potential solutions according to the issue that is being reported (ZHOU *et al.*, 2015), applications that analyze the text contained within tickets to verify how likely it is that they are correlated to one another (SALAH *et al.*, 2016), the reduction of the number of tickets that are created via the identification of repeating patterns (JAN *et al.*, 2015), the estimation of the resolution time (MOHARRERI *et al.*, 2016a), and the prediction of the workload of each ticket (KIKUCHI, 2015).

The work presented here is filed into the second category; that is, it addresses the process of analyzing the issue that has been registered and sending the ticket that holds its information towards the area that is best-suited to solve it. Depending on how a company is structured and on how it customizes the practices established by ITIL in order to better fit its reality, incident tickets can be solved by a large number of different departments. For instance, in some cases, teams may be grouped according to the level of

expertise required to deal with the tickets; as such, the first level of the service desk will be tasked with the easier issues, forwarding incidents that are harder to solve to the levels below it. In other cases, teams may be arranged by technical specialization; as such, for example, incidents related to network matters are handled by one group whereas incidents regarding database-level problems fall into the care of another team.

Regardless of structural approaches to service desk construction, though, the task of analyzing what has been logged into the ticket and identifying the team responsible for taking care of it represents a workload of varying weight that sometimes entails communication with the specialized areas and requires some height of technical and organizational knowledge. Systems that seek to address the problem of ticket forwarding, as a result, target the reduction or elimination of that workload through the automation of that process, which is in the hands of the teams involved in Incident Management and that are a part of the Service Desk function as described by ITIL.

In order to propose such a system, called Incident Routing, this work presents and investigates the real-life case of a large Brazilian IT service provider that adheres to the ITIL practices. The group of incident tickets that was obtained is employed in the analysis of the problem of incident classification through two different perspectives (one that does not take time-related changes in the data distribution into account and one that does); moreover, in the construction of the system via those two distinct views, techniques and problems belonging to the fields of automatic text analysis and classification are explored.

1.2 Goals

With Incident Routing, the objective is to reduce the workload of the Incident Management process and Service Desk function of a Brazilian IT Service Provider. That goal is achieved by analyzing the text inserted into incident tickets and automatically routing each of them (via classification algorithms) to one of the areas responsible for solving these issues. In order to report the gains obtained with the insertion of Incident Routing into the process, a comparison is made between the manual ticket-forwarding procedure that is currently in place and the automatic system itself. The difference is measured with a metric that seeks to represent the time and workload involved in the analysis process of each ticket.

Dubbed forwarding cost (MENEZES, 2009), the metric – in its introductory work – was defined as being a tool that makes it possible to quantify the effort spent in the manual forwarding of the tickets as well as the impact that an incorrect classification done by the automatic system can have on the process, representing the time and workload necessary to, after an incident is sent to the wrong team, relocate it to the appropriate department.

Throughout the work, text analysis and classification approaches (as well as techniques used to solve specific problems related to the two areas) are evaluated. The best ones are chosen and employed in the construction of a multi-leveled classification tree that mirrors the structure of the company's service desk. The different obtained configurations of the classification tree are then tested over a separate set of tickets and their performances are compared with the use of forwarding cost.

1.3 Organization

The work is divided into 6 chapters. Chapter 2 details the context in which the analyzed problem exists. It begins with the introduction of general concepts of Information Technology Service Management; proceeds into the realm of ITIL, the most widely known body of knowledge that is employed by companies to deal with issues in that area; discusses the two aspects of that library (Incident Management and Service Desk) that are most intricately connected to the problem that is approached in this work; and dissects the real-life scenario from which the data used in the experiments was acquired and that, therefore, guided the construction of the Incident Routing system.

Chapter 3 describes the problems and techniques that were used in the construction of Incident Routing and whose definition is, consequently, necessary to understand the system itself as well as the experiments developed in this work. Firstly, text transformation approaches (which turn texts into formats that are understandable by computers) are explained; subsequently, text classification algorithms that act upon the transformed text are detailed; finally, problems pertaining to those areas and that exist in the scenario and dataset being studied, as well as some approaches used to solve them, are exposed. The problems in question are those of Feature Selection, Imbalanced Datasets, and Concept Drift.

Chapter 4 discusses related works. The section is divided into two pieces as to represent the two perspectives of the ticket forwarding problem that are approached in this work. The first part looks into works that aimed to automatize aspects of the Incident Management process of ITIL. Meanwhile, the second part looks specifically towards the Concept Drift problem, detailing how it occurs in numerous areas in which artificial intelligence and classification are used.

Chapter 5 presents the dataset that was obtained, the evaluation means that were used, and the experiments the were performed. In the first set of experiments, the best approaches to text transformation, text classification, feature selection, and dealing with imbalanced datasets are chosen and then applied to a new set of tickets. The second set of experiments, on the other hand, is dedicated to concept drift, which is treated via both a traditional algorithm of the area as well as ensembles that are proposed in this work.

Chapter 6, finally, shows the conclusions reached in the two sets of experiments, reporting the gains that were obtained, and points towards possible future works.

Chapter 2

Problem Definition

2.1 Information Technology Service Management

As defined by ITIL (ITIL, 2011), a service is “a means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks”. In other words, customers want the tools to reach their goals without having to worry about the underlying network of practices, processes, people, and problems that are involved in the act of constructing those services, deploying them, and maintaining their operation. The concept of a service as well as the need to manage it predate information technology itself, as its origins can be traced to traditional businesses such as airlines, banks, hotels, and phone companies (ITIL, 2011). However, as the world has shifted from processes executed with physical documents and sheets of paper to activities performed via electronic devices and the applications they house, IT services have spread through all areas of the economy, which have become reliant on computers and smartphones. As a consequence, the need for the development of service management practices especially dedicated to the information technology field appeared.

The traditional businesses in which the idea of a service was born are, like many others, dependent on IT services. Airlines and hotels use systems and websites to book flights and rooms respectively; banks provide their clients with a big assortment of financial services that can be accessed from pretty much anywhere, as transfers, payments, and investments can be done with the click of a button; and phone companies register calls and manage their customers’ accounts through electronic means. If the systems that allow those companies to perform those activities experience problems, the business will suffer and clients may go looking, in the competition, for what they seek.

Such a dependency can cause a negative chain reaction. Businesses know that if their systems, which support the services they provide, do not work as smoothly as possible, there is a chance they will lose customers. Given that, in recent years, many firms have adopted outsourcing as a way to govern their IT operations (BAHLI; RIVARD, 2017), those businesses will, then, look for IT service providers that prove

themselves to be capable of delivering and supporting those systems with quality, avoiding issues such as slowdowns and periods of unavailability.

Naturally, considerable efforts have been made in the IT field in order to develop sets of practices and strategies that aim to allow IT service providers to better manage their assets so they can deliver value to their customers. As examples of those initiatives, it is possible to mention:

- **Control Objectives for Information and related Technology (COBIT):** A methodology that is focused on the business, oriented by processes, based on controls, and driven by metrics that seeks to control and manage the risks and vulnerability of IT by ensuring that the enterprise's IT becomes an extension of the company's strategies and objectives (LAINHART IV, 2000).
- **Service-Oriented Architecture (SOA):** An architectural style that is to be used in the construction of software so that the different individual functionalities of applications are seen and managed as standalone services, allowing for quick re-use of distinct software pieces, fast reaction to business changes, simpler infrastructural management, and a clearer alignment between the strategy of the IT service provider and the applications it supports (ERL, 2005).
- **The Open Group Architecture Framework (TOGAF):** A framework to the construction of an organization-wide architecture that maps the relationship between the building blocks of systems. Modeling is done in layers that represent the infrastructure, the applications, and the business. Therefore, it is possible to visualize how applications sustain business processes and which elements of the IT infrastructure are connected to each system and business process (HAREN, 2011).

IT services are built on infrastructure (servers, storage units, network equipment, among others) that is rather complex, and as the deck of services provided by an organization increases, so does the intricacy of the web of elements it stands on. If services are not to be disrupted, the supporting infrastructure must be closely managed: changes and updates must be planned so that they are carried out in an orderly manner that reduces risks; moreover, when failures occur, they need to be recorded, analyzed, and

treated with efficiency and care (JOHNSON, 2007). Therein lies the need for Information Technology Service Management (ITSM) practices.

ITSM is an approach to IT operations that puts an emphasis on IT services, customers, the maintenance of appropriate service levels and quality, and the handling of daily activities of the IT function through clear processes; consequently managing IT as a service (IDEN; EIKEBROKK, 2013). The clearest definition to the general structure that defines Information Technology Service Management comes through the international standard ISO/IEC 20000 (VAN SELM, 2009).

According to the norm, services differ from products due to the fact they are:

- Highly tangible, as they cannot be touched or weighed.
- Produced and consumed at the same time, as they cannot be stored.
- Variable, as human beings (whose behavior shows a lot of variability) are usually involved in their delivery.
- Measurable only after they have been delivered, as they cannot be assessed before they are put in place.
- Dependent on the user during the production cycle, as they are constructed with the customer's input.

Moreover, the norm describes quality management principles that must guide the delivery of IT services. These principles are:

- Customer focus, for organizations depend on them
- Leadership, for leaders establish unity and direction.
- Involvement of people, for they are the essence of the organization
- Process approach, for managing activities as processes leads to results being achieved more effectively.
- Continual improvement, for increasing consumer satisfaction should be a permanent objective of the organization.
- Factual approach to decision making, for effective decisions are based upon accurate information.
- Mutually beneficial supplier relationships, for an organization and its suppliers are interdependent.

- System approach to management, for understanding the relationship between processes is essential.

ITSM, therefore, seeks to manage all processes that are involved in the delivery of IT services so that quality is achieved and results are constantly improved. Such a goal is reached through the four-step Deming Cycle, which is a widely used framework to the management and control of businesses, and that dictates a continuous process of plan (that is, establishing the objectives and thinking about how to reach them), do (which marks the execution of the plan), check (the comparison of the obtained results with the reached results), and act (the performing of corrections and adjustments).

2.2 Information Technology Infrastructure Library

The creation of the Information Technology Infrastructure Library (ITIL) actually precedes that of the international standard ISO/IEC 20000 that establishes the framework for service management inside information technology companies. ITIL was born out of an initiative from the United Kingdom government, which was looking to compile the practices employed by leading organizations in the management of their IT services. In its current format, the core of the body of knowledge it contains is organized over five publications that represent a lifecycle guiding IT services all the way from their inception to their delivery and continuous improvement. Figure 1 – ITIL Lifecycle shows the service management lifecycle as presented in ITIL v3 (ITIL, 2011).

Service Strategy, located in the center of the lifecycle, helps organizations view IT services and their management in a strategical way. As such, it focuses on identifying opportunities, setting goals as well as expectations, and preparing companies to the costs, risks, and operational actions that are involved in the process of maintaining IT services and the infrastructure that supports them. Through Service Strategy, therefore, organizations must define which services to offer, how to deliver value, how to define service quality, how to manage their resources, and how to plan their investments.

Service Design is the step that turns what has been planned during Service Strategy into actual service assets. That way, it needs to make sure professionals keep the business needs in mind when creating the services and guarantee they use the recommended practices. Services need to be designed within time and cost constraints,

delivered on time, and supported by an infrastructure that will be able to handle the load of users and accesses it will be subjected to. In order to achieve that, Service Design outlines processes to document plans, policies, architecture, and training; create services that are reliable, maintainable, serviceable, continuous, and that have the availability and performance customers expect from them; and ensure information is secure and suppliers that are essential to the service are well-managed.

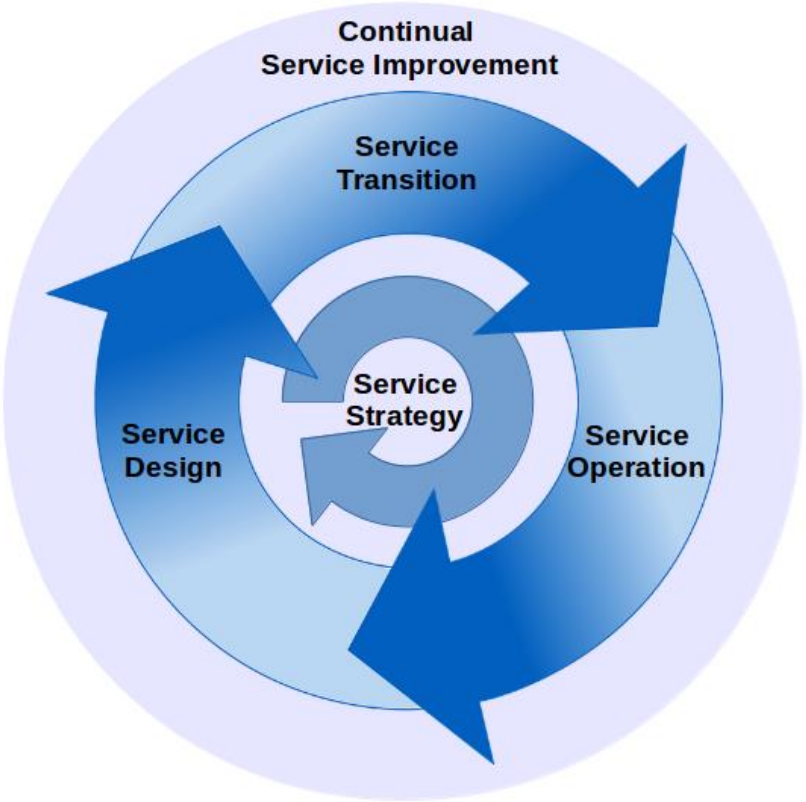

Figure 1 – ITIL Lifecycle

Service Transition deals with the management of the process of taking a service that is either new or that has been altered in the Service Design phase to a productive environment, where it will become available to be consumed by customers. Essentially, it intends to assure that changes that are executed on the IT infrastructure – as it gets prepared to receive services that are coming out of the Service Design step – be performed in an organized and controlled manner. Additionally, Service Transition includes the validation and testing of services, the maintenance of a record of all infrastructure items and their configurations, and the planning of how the transition of the services will be carried out.

Service Operation, meanwhile, takes care of day-to-day activities that are necessary to keep the IT infrastructure – and, as a consequence, the services it supports – running as smoothly as possible. The tasks necessary to accomplish that goal involve monitoring the infrastructure and dealing with low-risk changes of minimal impact, requests to give users access to services, and events that may occur.

Finally, Continual Service Improvement, which surrounds all previous lifecycle phases, is related to the Act step of the Deming Cycle; that is, it concerns new actions that come to be through the analysis of the outcomes of the four aforementioned steps. It is via Continual Service Improvement that all processes that belong to the previous categories are refined, allowing companies to perform incremental improvements in service quality, operational efficiency, and business continuity (ITIL, 2011).

All of the five pieces of the ITIL lifecycle are further divided into smaller units that cover the details of how IT services can be managed. Among those units, the two that are most important are processes and functions, which are defined as follows:

- **Process:** An enclosed system that receives inputs (such as data, information, and knowledge), alters those assets via a series of activities, and produces a desired outcome. A process is measurable, responds to precise events, and has specific results that are delivered to a customer.
- **Function:** One or more units within an organization that execute a specific work and are, therefore, responsible for certain results. Functions and the professionals found within them take on the roles that handle the processes as well as the activities within them.

The system proposed in this work, Incident Routing, exists within the context of the Service Operation lifecycle phase. More specifically, its goal is to bring more efficiency, via the reduction of the workload, to the Incident Management process; this process, in turn, requires the work of the Service Desk function.

2.3 Incident Management and Service Desk

Incidents are unexpected events that reduce the quality of services or cause interruptions. Moreover, the ITIL definition of incidents also includes events that have

the potential to cause such issues. As such, for example, a database that is being more accessed than expected and is getting dangerously close to the upper threshold of its supported load qualifies as an incident even if it is not affecting the service's performance.

The goal of the Incident Management process is, then, to handle those types of events as quickly as possible in order to minimize negative effects on the operation of the services, or avoid them altogether, and restore the infrastructure to its normal state. By doing so, companies will guarantee that the best levels of service quality and availability are maintained, something that is rather important not only from a business image perspective, but also from a financial point of view, as contracts between IT service providers and their customers usually present a Service Level Agreement (SLA), which specifies the quality standards and availability time that a service must have. If SLAs are not respected, providers may have to deal with fees that are written onto the contracts.

For that reason, a company's Incident Management must keep in mind the time limitations that surround the solving of an incident, look for ways to define specific procedures to handle incidents that are judged to be more serious, and define different incident models so that issues that are logged can be treated differently according to the type of problem they report. In a general way, however, ITIL defines generic activities that make up the Incident Management process. Figure 2 – ITIL Incident Management Process shows the information flow between them as adapted from ITIL v3 (ITIL, 2011).

The Incident Management process can be started in many ways: users, technical personnel, or management can report issues; monitoring software may emit warnings; and so forth. Once incidents are pointed out, the following procedure is followed:

- **Incident Identification:** As, ideally, incidents need to be resolved before they impact services, a strong monitoring process must be put in place so that incidents are identified quickly. Without identification, incidents cannot be treated.
- **Incident Logging:** After identification, incidents need to be registered. Essential information that needs to recorded includes description, symptoms, impact, priority, among others. In this step, the incident ticket is created.
- **Incident Categorization:** In order to maintain a certain level of control over what types of incidents have been occurring, as well as making it possible to forward them to specialized technicians, incidents are categorized. Following this step, if the reported incidents are labeled as mere service requests, which do not have the

potential to disrupt services, they are removed from the Incident Management flow and treated separately.

- **Incident Prioritization:** Assigning a priority to an incident can determine how it will be handled by support tools and staff. In general, incident priority is calculated by taking into account their impact and urgency. If incidents are considered to be critical, it is possible to treat them separately through an exclusive flow.
- **Initial Diagnosis:** During the diagnosis step, the analyst handling the incident – ideally while in contact with the source that has reported the issue – can dive further into the matter in order to get more details about what is happening so that such information is added to the incident ticket, which may help down the line when the incident is being solved. Following the diagnosis, if the professional registering the incident feels they are unable to solve it, they can escalate it to other more specialized teams.
- **Investigation and Diagnosis:** To solve the incident, it is necessary to investigate and diagnose what has occurred.
- **Resolution and Recovery:** A resolution is identified, applied, and tested; after that, the normal state of the service is recovered. It is important that information regarding these procedures be logged into the ticket so that a full history of the incidents is kept.
- **Incident Closure:** After it is verified that the applied resolution has actually solved the incident and that users are happy with the results achieved to the point where they agree the incident has been solved, the ticket needs to be closed. In this step, professionals handling the ticket can also verify if the categorization was done correctly and make sure the whole history of the incident has been registered.

As a whole, these Incident Management steps are performed by the Service Desk function. There are many different ways of structuring service desks (ITIL, 2011), so it is natural that organizations build that function in manners that best suit their general operational composition. Service desks can be local when they are positioned close to the community or infrastructure they are responsible for; they can be centralized when all of their professionals are located in the same place, sometimes far away from the users that are reporting the incidents; they can be virtual when, through the use of technology, a

service desk that is broken into geographically distributed pieces appears to users as a centralized unity; and they can follow the Sun, when they are structured in such a way that there are service desk units across the globe, with incidents always being forwarded to the ones found in countries that are on the standard working hours, ensuring – therefore – a 24-hour coverage.

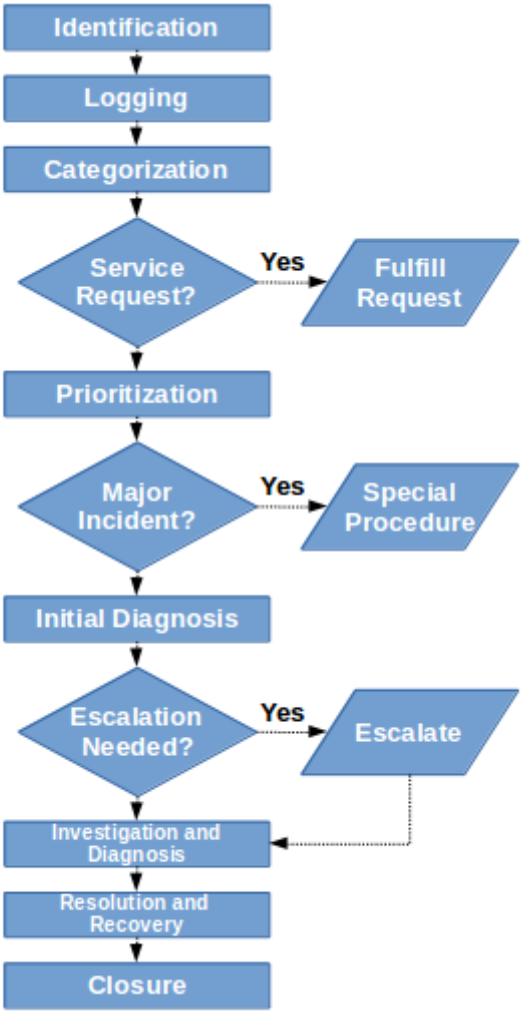

Figure 2 – ITIL Incident Management Process

Furthermore, service desks can be grouped either into skill levels or specialization, or even through the use of a mix of both approaches. If skill levels are used to construct service desks, the first level – the one responsible for receiving calls and incidents – will handle the simpler-to-resolve issues; if the incident in question proves to be too difficult for the professionals on the service desk's entry point, it can be forwarded to lower levels until it reaches analysts with the appropriate expertise to solve them. If specialization is

used, on the other hand, the step of the Incident Management process that categorizes the event can determine to which group of specialists it will be forwarded.

2.4 Case Study

The incident tickets analyzed in this work belong to a Brazilian IT service provider with multiple customers that, in turn, serve a considerable portion of the nation's population. The infrastructure supporting these services is spread across the country in multiple data centers. As it implements ITIL, the company has both an Incident Management process and a Service Desk function in place, each adapted from the general outline described in the body of knowledge.

The Service Desk function is broken into units allocated to each of the data centers as to work within proximity of the technological infrastructure they must take care of. Additionally, instead of being broken down into levels responsible for resolving incidents of different heights of complexity, the service desk is grouped by specialization; as such, tickets are routed through the function's structure according to the nature of the incident they describe. For example, network-related incidents, such as connection issues, are handled by one team, whereas faults in operational systems are sent to another group.

In the current scenario, for the case of the incident tickets that were used in the experiment, a single team centralizes the creation of the tickets and their forwarding to the areas that have the technical ability as well as the responsibility to solve them. In general, these tickets are opened due to three triggers: warnings from infrastructure-monitoring software, external calls from customers that are experiencing problems, and internal requests made by the company's own employees, covering by such means the Incident Identification step of the Incident Management process.

This centralized team, then, uses a system to fill up information about incidents (performing the initial diagnosis, logging, and prioritization); after that, via a dropdown list and usually following an initial analysis of the issue, which may entail further investigation of the event and communication with other areas and analysts, the professional logging the incident forwards the ticket to the area that, according to their evaluation, is better suited to act on the matter (marking the execution of the Incident Categorization step). The process proceeds with the investigation, diagnosis, and resolution of the incident; the recovery of the service; and the closure of the ticket as done

by the technical team that received it. In case the categorization of the incident is done incorrectly, it is the task of the specialized analysts to return the ticket to the team responsible for its initial reception and forwarding so that it can be correctly assigned.

Figure 3 shows an overview of the implemented process; in particular, it looks at the procedures of the Service Desk function by focusing on the piece of the process this work seeks to automatize in order to improve the efficiency with which Incident Management is performed: the forwarding of the tickets.

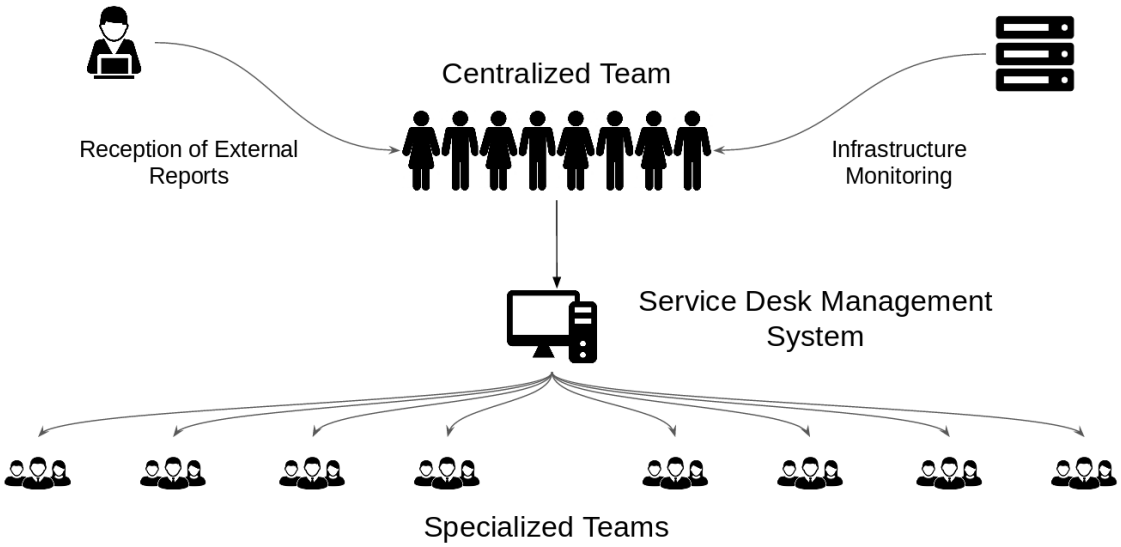

Figure 3 – Manual Ticket Forwarding Process

Despite the fact that, in practical terms, the company's service desk is not divided into multiple levels (featuring only a team that is responsible for sending tickets to specialized groups), a segmentation of the specialist teams into levels exists in theory. In this work, that theoretical division is taken into account in order to build a structure of classifiers that will direct the tickets to the target areas.

Not only will such a tree materialize a multi-level service desk setup that does not currently exist in the company, but it will also make it easier for the classification algorithms that will be used to accurately classify the incident records. That happens because, in total, there are 13 specialized teams; a division of those groups into levels will, consequently, avoid that a single classifier be required to learn patterns for 13 ticket categories. The theoretical organization of levels and the existing specialized teams are shown in Table 1.

Table 1 – Service Desk Levels and Specialized Teams

Level	Specialized Team
Level 0 (L0)	Cloud
	External Network
	Incident Management
	Production Support
Level 1 (L1)	Backup
	Control
	Job
	Monitoring
	Operation
Level 2 (L2)	Application
	Database
	Internal Network
	Platform

Seeking to replicate that structure in the ticket classification that will be done, Incident Routing will focus on the development of a classification tree, as seen in Figure 4.

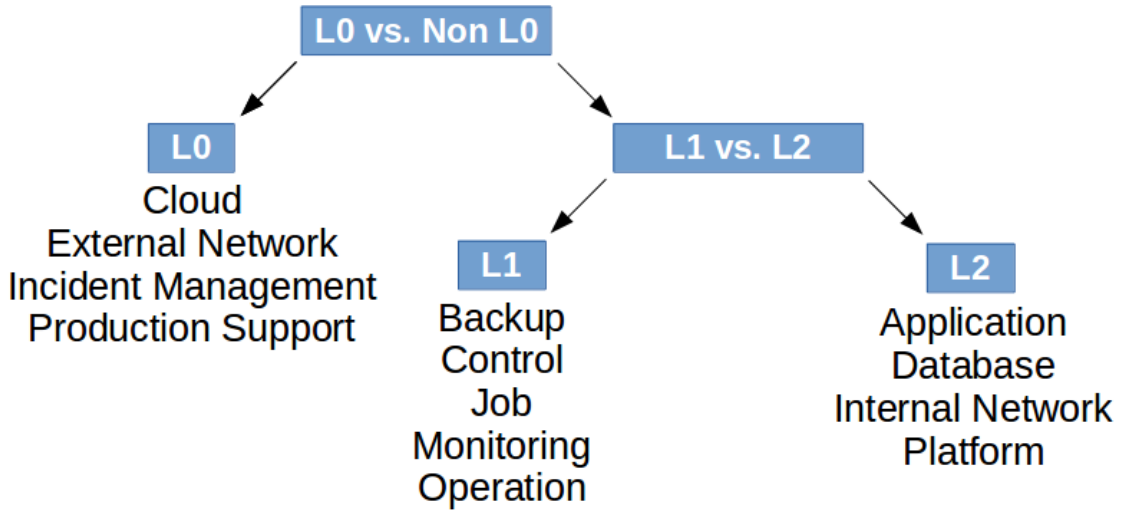

Figure 4 – Tree of Proposed Classifiers

The tree is composed of five classifiers:

- **L0 vs. Non L0:** It is the point on which all tickets arrive and start their routing journey. It will differ between tickets of Level 0 and those of Level 1 and Level 2 (Non L0).
- **L1 vs. L2:** It will receive tickets labeled as Non L0 by classifier L0 vs. Non L0 and determine whether they belong to Level 1 or Level 2.
- **L0:** It will receive tickets labeled as L0 by classifier L0 vs. Non L0 and determine to which technical team of Level 0 they belong to.
- **L1:** It will receive tickets labeled as L1 by classifier L1 vs. L2 and determine to which technical team of Level 1 they belong to.
- **L2:** It will receive tickets labeled as L2 by classifier L1 vs. L2 and determine to which technical team of Level 2 they belong to.

Chapter 3

Incident Routing – Components

3.1 Text Transformation

The essence of Incident Routing is removing, from the hands of human agents, the process of analyzing the text of the incident and figuring out the area to which the ticket must be sent. That process is to be done by the system itself, which will look into what has been written in order to, then, categorize the ticket. The automation of such a task through the use of computers is achieved in two steps, the first of which is text transformation.

Text transformation is the conversion of the content of a textual document so that it can be recognized by a computer, allowing the machine to process and classify it (LAN *et al.*, 2009). In the experiments that were done to investigate the best ways to categorize incident tickets, text was transformed into numerical vectors by vector space models, where each dimension of the vector corresponds to a term in the text; and one word embedding technique, where words are mapped to vectors.

3.1.1 Term Frequency and Variations

Term Frequency (TF) and its variations (ROELLEKE, 2013) come from the Information Retrieval (IR) field, which aims to recover information from relevant sources so that it can be employed according to one's needs.

With Term Frequency, terms that appear in the text become the dimensions of the vector that will work as the representation of a document (in the case of this work, a ticket). The value that is assigned to each dimension is the number of instances of the term that appear in the text. In order to illustrate how Term Frequency works, Table 2 exemplifies how two tickets (which are not real, given their content cannot be displayed for security reasons) are mapped into vectors through Term Frequency. For normalization purposes, the term counts can be divided by the total number of terms in the document; in the implementation used, however, normalization was not done.

Tickets:
"Unable to access main page. Page error."
"Unable to log into the system."

Table 2 – Term Frequency Example

Ticket ID	unable	to	access	main	page	error	log	into	the	system
1	1	1	1	1	2	1	0	0	0	0
2	1	1	0	0	0	0	1	1	1	1

In addition to TF, three of its variations were also tested: Term Frequency – Inverse Term Frequency (TF-IDF); Logarithmic Term Frequency (TF-LOG); and Fractional Term Frequency (TF-FRAC). In common between these three strategies is how they all start by doing the what TF does; in other words, they first count how many times terms appear in each document of the corpus that is being analyzed. What differs them is the operation that follows that procedure.

TF-IDF takes into consideration how frequent the term is in the rest of the documents; with that, terms that are bound to be frequent all over the set of documents, such as articles and prepositions, as well as words that are common in the domains the corpus includes, are weighted as to diminish their importance. In that case, therefore, TF is multiplied by IDF, which, for a given term *t*, is calculated by Formula 1.

$$idf(term) = \log\left(\frac{Number\ of\ Documents}{Number\ of\ Documents\ With\ Term}\right)$$

Formula 1 – Term Frequency - Inverse Term Frequency

With TF-LOG, meanwhile, the first occurrence of a term in a document counts in full while subsequent appearances count progressively less, with the second instance of a term in a document counting in half, the third instance counting as one-third, and so forth (ROELLEKE, 2013). TF-LOG for a given term *t* in a document *d* is defined by Formula 2.

$$tflog(term) = \log(1 - tf)$$

Formula 2 – Logarithmic Term Frequency

Finally, the TF-FRAC approach does the same as TF-LOG: it diminishes the weight of occurrences of a term following the first one. That is achieved, as seen in Formula 3, by dividing the value of TF by itself plus an adjustable constant *K*. In the implementation used, *K* was set to 1.

$$tffrac(term) = \frac{tf}{tf + K}$$

Formula 3 – Fractional Term Frequency

3.1.2 Word2Vec

Created by a team of researchers at Google, Word2Vec (W2V) distinguishes itself from the classical models of text transformation that stem from TF due to how it does not view words as atomic units represented by indices that cannot be differentiated from one another (MIKOLOV *et al.*, 2013b). By turning all words into mere dimensions of a vector, TF loses the semantic value the terms carry, as all of them become word counts: their meanings cannot be compared and the similarity between words cannot be estimated.

Word2Vec uses neural networks that analyze the content of a corpus in order to build a language model in which every word is represented by a vector that carries semantic value. That way, words that have similar meanings will be shown as similar vectors; due to that, CEOs and the companies they manage, countries and their capitals, verbs in the infinitive and gerund forms, the singular and plural forms of nouns, and other associations will also be captured by vector representations. Due to that, relationships between the terms can be extracted via arithmetic operations: for example, the representation of the word “queen” can be recovered from the representations of “king” , “man” and “woman”.

In order to achieve that, Word2Vec takes into consideration the context in which words tend to appear in the text from which it learns its language model, such context is marked by the words that both precede and follow the appearance of the term in the text. Figure 5, adapted from a work that shows vector-space word representations capture meaningful syntactic and semantic value (MIKOLOV *et al.*, 2013c), illustrates the kinds of relations between words the Word2Vec approach can yield.

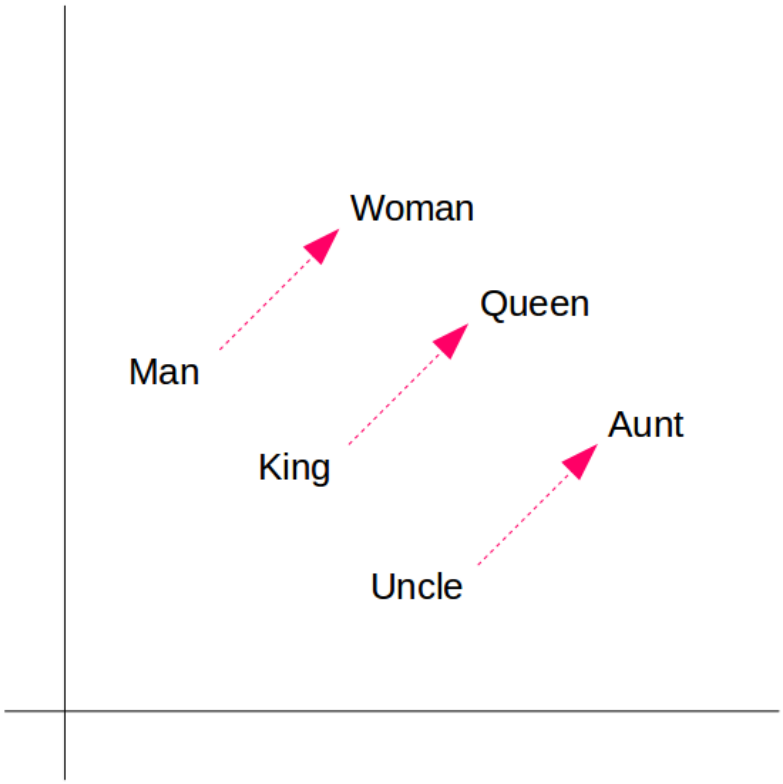

Figure 5 – Examples of Word2Vec Representations

Due to its ability to capture the essence of words, Word2Vec has been used in tasks such as the search for word analogies (for example, to seek what term that is to "Japan" as "Paris" is to "France"); the calculation of the similarity between terms; and the identification of named entities, i.e., people, companies, brands, products, and others (PENNINGTON *et al.*, 2014). It has, however, also been employed in works dealing with text classification (LILLEBERG *et al.*, 2015). In that area in particular, Word2Vec is a good alternative to TF and its peers because it translates documents into vectors with a lower number of dimensions.

While in the former approaches a vector representing a document is as big as the total number of words found in the corpus being analyzed (as each term represents a dimension); in the latter, all words become vectors of predetermined configurable dimensions, with the work in which Word2Vec is introduced (MIKOLOV *et al.*, 2013b) dealing with vectors whose dimensions range from 50 to 640. And given texts can be represented by the element-wise aggregation of the vectors of its words, such as via addition (MIKOLOV *et al.*, 2013b), the entire corpus can have its dimensionality reduced to that level.

It is important to highlight that pre-trained models for English – such as GloVe (PENNINGTON *et al.*, 2014), which was trained on the Wikipedia database – and Portuguese – such as NILC-Embeddings (HARTMANN *et al.*, 2017) – that yield vector representations for words in those languages individually exist. However, given the corpus of the incident tickets presents a mixture of Portuguese and English, the model was trained on the training corpus itself; that is vector representations of the words were learned based on the tickets. The Word2Vec implementation used was that of the Gensim (REHUREK; SOJKA, 2010) Python package.

3.2 Text Classification

Text classification, or text categorization, is the task of automatically classifying natural language documents that are unlabeled (that is, it is not known to which class they belong) into a predefined set of semantic categories (LAN *et al.*, 2009). After the text of said documents has been transformed, text classification is done by feeding that data into machine learning algorithms. Machine learning has been defined as an intersection between Computer Science and Statistics that tries to give computers the power to program themselves by inferring from available data and learning patterns in order to solve problems with a certain level of reliability (MITCHELL, 2006).

In general, machine learning problems can be separated into two groups: supervised learning and unsupervised learning. In supervised learning, the data that is given to the algorithms contains both an input and an output; whereas in unsupervised learning, the data does not feature any information regarding the output.

In a sentiment classification problem where computers are trained to identify whether a text is positive or negative (PANG *et al.*, 2002), for example, a supervised learning approach would require a dataset with documents (the input) and the sentiment expressed in them (the output). If the documents are available but the output is not, it is possible to group those that are similar into different categories via unsupervised learning techniques such as clustering (STEINBACH *et al.*, 2000); those categories, however, will be unlabeled. As such, unsupervised learning is a task of spontaneously finding patterns and structure in input data (ABU-MOSTAFA *et al.*, 2012) rather than looking to file them into predetermined classes.

Since this work involves identifying to which of the 13 specialized teams inside a company an incident ticket must be sent, and given labeled tickets are available (as incidents that are solved by a particular area are automatically tagged as belonging to that group), the problem approached falls into supervised learning.

Figure 6, adapted from the "Learning from Data" book (ABU-MOSTAFA *et al.*, 2012), shows the basic setup of the learning problem in the supervised scenario, breaking it up into distinct components.

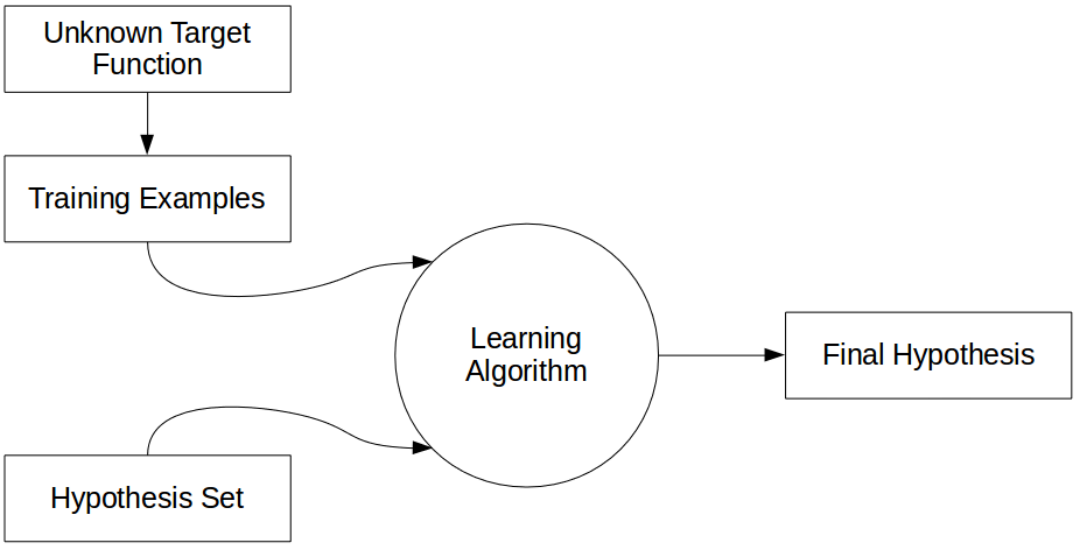

Figure 6 – The Learning Problem

Firstly, there is an unknown target function. This function is the ideal result of the machine learning algorithm, as it perfectly transforms the input into the output. Therefore, if a computer were capable of capturing it, it would be able to correctly classify all of the data. Such a target function, however, is unknown and remains so through the duration of the problem. The training examples that are available are fed into the algorithm; they are generated by the target function and are used by the algorithm to try to approximate the function as well as possible. With the training examples, the algorithm navigates through a hypothesis set, which is a group of candidate formulas; this process of navigation is done by picking a formula, applying it to the labeled training data, verifying the error metric, and either choosing the picked hypothesis as the final one or executing a procedure to select another hypothesis that will supposedly bring better results (that is, a smaller error rate). When the algorithm produces a final hypothesis, the training process is concluded and the picked formula can be used to classify new data that has yet to be labeled.

The hypothesis set and the learning algorithm are referred to informally as the learning model (ABU-MOSTAFA *et al.*, 2012). In the construction of Incident Routing, 7 popular machine learning models were tested:

- Decision Tree;
- Naive Bayes;
- Nearest Neighbors;
- Neural Network;
- Random Forest;
- Stochastic Gradient Descent;
- Support Vector Machines.

Additionally to those models, the ensemble methods bagging and boosting were also employed in experiments to deal with class imbalance problems. The implementation that was used for all of the models was that of the Python Scikit-learn package (PEDREGOSA *et al.*, 2011). The learning models were tested in their standard configuration; no changes or refinements were done to their predefined parameters. Their descriptions, as presented in this work, are (except when noted) based on those found in the "Data Mining: Concepts and Techniques" book (HAN *et al.*, 2011).

3.2.1 Decision Tree

Decision trees are a popular kind of classifier, with generally good performance, whose inner workings are easy to understand given they are based on simple decisions made according to attributes of the items being analyzed. They are made up of the following components:

- **Root node:** The initial node where all of the labeled data that is being used for the training process (that is, the finding of the final hypothesis) is placed.
- **Internal node:** Represents a test that is done on an attribute of the data. The goal of the tests performed by internal nodes is to separate the different classes of the dataset. As the tree's root node performs a test, it is also considered an internal node.

- **Branch:** Branches are responsible for holding the outcome of the tests executed by internal nodes.
- **Leaf node:** Leaf nodes are special branches, as they are the final components of a decision tree and represent a specific class. In basic decision tree algorithms, trees usually stop being constructed when all items in a leaf node belong to the same class; if that does not happen, majority voting is executed and the class associated with the leaf node is that of most of the items. After the construction of the tree is done (the training phase), the classification of new incoming records is determined by the leaf node they fall into.

Figure 7 illustrates how decision trees work towards the classification of data. Leaf nodes are represented as circles, while internal nodes are shown as rectangles.

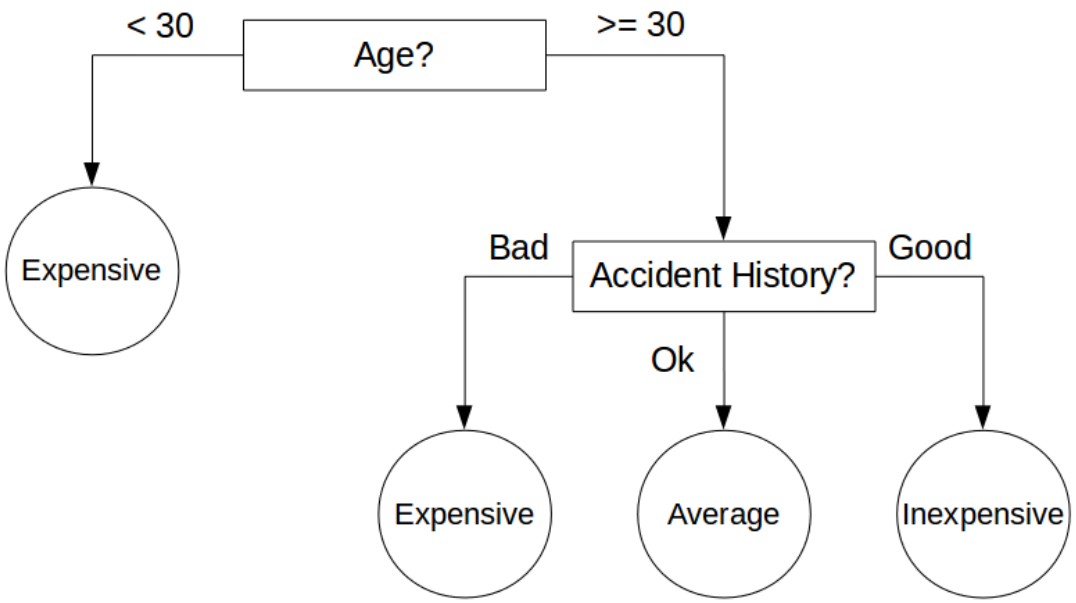

Figure 7 – Decision Tree Example

In the example, the decision tree is learning how to determine whether someone's auto insurance will be inexpensive, have an average value, or be expensive according to the attributes that are found in the data. Decision trees can be binary, if all tests that are performed lead to two outcomes (two branches); or nonbinary, if internal nodes can produce more than two branches. The one shown in the image is of the nonbinary kind.

The trick to decision trees rests in how they choose to divide the data. The records being classified can have hundreds of attributes; in fact, the ones from the dataset used in

this work reach into the thousands. Therefore, decision trees need to identify what are the features that best differ items from a class from those of another class. In the case of the implementation used in this work, that measure is the Gini Index. That measure, which only considers binary splits, evaluates all possible divisions that can be made with each attribute, choosing the division that generates the sets with least impurity, meaning the smallest value of the Gini Index. Minimum impurity is achieved when all items inside a split have the same labels. That happens because impurity is calculated by the probability that items with specific labels be misclassified if their class is chosen according to the distribution of classes in the split. Therefore, if all records inside a subset are of the same class, misclassifications will not occur.

In cases where the dataset is very large and decision trees are constructed until all leaf nodes have items of the same class, they can become too complex and overfit the data, which can lead to worse predictive performance as such a detailed level of construction can mean the model has memorized the data (with its outliers and noise) instead of learning patterns from it. To counter that problem, trees can pruned either by prepruning approaches that stop their construction earlier by limiting attributes such as the depth of the tree, or by postpruning techniques that remove subtrees and turn them into branches. The increasing of a tree's performance via pruning, though, goes through choosing the best way to do so, as pruning a tree ineffectively can lead to worse generalization (MURTHY, 1998). As such, the decision trees used in the classification of incident tickets were left unpruned.

3.2.2 Random Forest

A random forest is a type of ensemble classification method. Instead of working with the construction of a single model, ensembles combine numerous learning models in an attempt to improve the quality of the achieved classification. These models, whose results are aggregated to yield the final classification output, are called base classifiers; they are constructed using subdivisions (or subsets) of the original dataset. Ensembles tend to produce better results than single classifiers, because while classifiers that work alone will make categorization errors just by choosing the wrong class, ensembles only make mistakes when the majority of the base classifiers that come together to form them misclassify a record.

Random forests receive such a name because they are made up of multiple decision trees. The random forest employed in Incident Routing consists of 10 base estimators (in other words, 10 decision trees). Those trees are constructed with random subsamples of the original training dataset, and the subsamples have the same number of items as the training dataset; that is, if the full training dataset has 100 records, the subsample of each tree will also have 100 records.

The randomness of the subsets and the way they differ from one another stems from how the sampling is done: with replacement. That sampling scheme means that once an item from the full set is randomly chosen and taken inside the subset, it is also placed back into the full set. That means that there may be repeated records in the subsets, and that even though they all have the same number of items as the full set, they are not equal. Figure 8 illustrates how the sampling scheme works, with two different 6-item subsets being constructed out of a full set that also has 6 items.

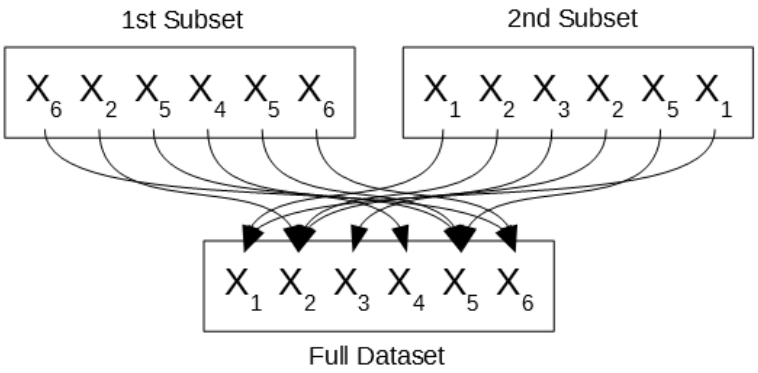

Figure 8 – Sampling With Replacement

Furthermore, as it was the case with the decision tree algorithm, the Gini Index is used in the choice of the best split inside each node of the tree, estimating how impure the sets produced in the filtering by a specific value of a certain attribute will be and choosing the best possible binary division.

3.2.3 Naïve Bayes

Naive Bayes is a type of Bayesian classifier that can statistically predict how likely it is that a record belongs to a certain class. They are often used in text classification tasks as a baseline because they are fast and easy to implement (RENNIE *et al.*, 2003); such a speed, however, comes from the fact they assume that, given a certain class, the effect of

an attribute value on that class is independent of the values of the other attributes. For example, if one desired to estimate the probability of a patient having a heart attack, a Naive Bayes classifier would look at the attributes of "age" and "weight" as contributing independently to the development of that disease, ignoring any correlation that may exist between them. That assumption has negative effects on its classification results and is what makes those classifiers earn the adjective "naïve".

Bayesian classifiers are filed into such a group because they are based on Bayes' theorem. As depicted by Formula 4, Bayes' theorem aims to calculate *P(H|X);* in a classification problem, that formula can be explained as the probability that a record with the attributes *X* belongs to a given class (an affirmation represented by *H*). More specifically, as a fictional example based on the incident tickets of this work, it describes how likely it would be that a ticket containing the terms "internet" and "connection" belonged of the "Internal Network" class.

$$P(H|X) = \frac{P(X|H)\,P(H)}{P(X)}$$

Formula 4 – Bayes' Theorem

To reach that result, Bayes' theorem uses: *P(X|H),* called posterior probability, which is the probability that a ticket *X* has the terms "internet" and "connection" given it is of the "Internal Network" class; *P(H),* called prior probability of *H*, which is the probability a ticket is of the "Internal Network" class regardless of the terms it contains; and *P(X),* called prior probability of *X*, which is the probability that a ticket from the dataset has the terms "internet" and "connection".

For one record *X*, therefore, the Naive Bayes classifier works by estimating *P(H|X)* for all classes of the dataset and labeling it as being from the category it is most likely to belong to. *P(X)* is the same for all classes; as a consequence, the classifier looks to maximize the two other terms of the equation: *P(X|H)* and *P(H)*. The computational gains of the Naive Bayes algorithm and its naiveté come into play in *P(X|H)*. For datasets where records have a lot of attributes, such as is usually the case of text classification problems due to the lengthy vectors produced by Term Frequency and its variations, computing *P(X|H)* for all classes and all attributes would be too computationally expensive if there was no assumption made on their independence.

However, as correlation between attributes is ignored, for a ticket with terms "internet" (X_1) and "connection" (X_2), $P(X|H)$ can be simply calculated by multiplying $P(X_1|H)$ and $P(X_2|H)$, which are – respectively – the probability that a ticket has the term "internet" given it is of the "Internal Network" class, and the probability that a ticket has the term "connection" given it is of the "Internal Network" class.

3.2.4 Nearest Neighbors

Differently from the other classification algorithms used in Incident Routing, a nearest neighbors classifier is a lazy learner. Lazy learners oppose eager learners because while the latter work with the training dataset of labeled data to produce a model that approximates the target function generating the examples, the former do not. Lazy learners delay the work. Firstly, they store all of the training data; it is only when the new unlabeled examples arrive that a lazy learner will look for a way to categorize it. On the negative side, as rather than coming up with a clean function they keep the data, lazy learners require plenty of storage; on a brighter note, they support incremental learning without any need for adaptation.

Incremental learning entails the update of the final hypothesis obtained by the classifier when new data arrives and is correctly labeled; given lazy learners do not work towards the building of a hypothesis, there is no updating to be done: all they have to manage is the storing of yet another data point.

Nearest neighbors classifiers work by representing each record inside a space of N dimensions, where N is the number of attributes of the training dataset items. For a text classification task, that means the algorithm will potentially be working with thousands of dimensions, each one representing a term in the text. When an unlabeled item arrives, the nearest neighbors classifier: places it in the space; identifies the labeled points that are the closest to it; and conducts a voting between those points, which – naturally – vote for the class they belong to. The most-voted class is assigned to the new point.

Figure 9 gives an example of a classification problem where a nearest neighbors classifier is trying to predict whether a customer that goes into a bank searching for credit will have their request approved or not. One axis represents the value of the age attribute; the other represents the value of the customer's income. Green circles are customers who

have had their credit requests approved, while red circles show customers who have had their requests denied. The black dot shows the new customer whose fate is undecided.

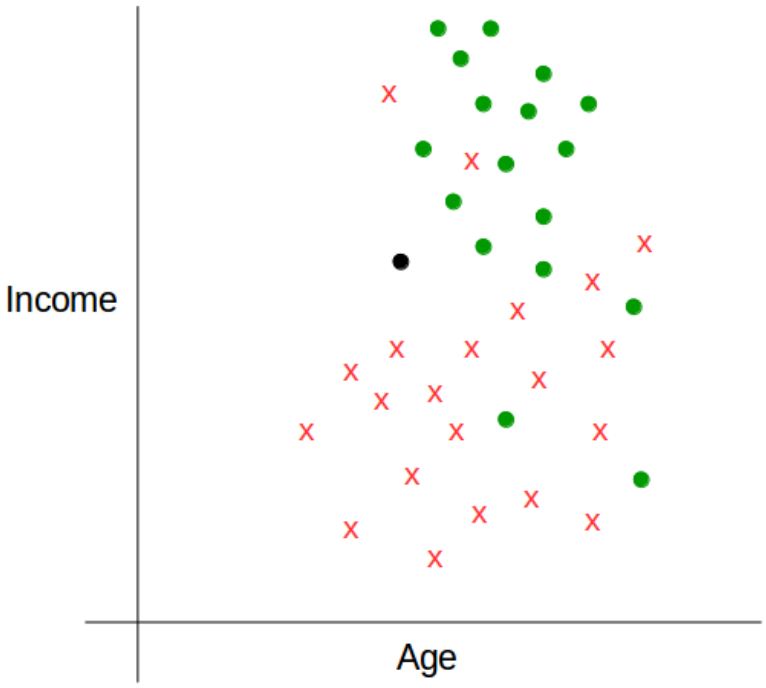

Figure 9 – Nearest Neighbors Example

Parameters of nearest neighbors classifiers include how many neighbors will be considered in the voting and what distance metric will be used to evaluate which ones are closest to the new item. In the experiments performed in this work, 5 neighbors were considered; and the distance metric employed was the Minkowski distance (CUNNINGHAM; DELANY, 2007), defined by Formula 5, with $p=2$ (which makes it equivalent to the Euclidean distance), and where x_i and y_i are the attributes of points X and Y.

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

Formula 5 – Minkowski Distance

The calculation of the distance between a new point and all other previously stored labeled items can cause nearest neighbors classifiers to be extremely slow in terms of classification, as all points need to be tested so that the closest ones can be discovered. The number of necessary comparisons can be diminished, however, by storing the tuples in search trees, using partial distance calculations (where only a subset of the attributes is

considered and if the distance exceeds a threshold the comparison is stopped), or editing the stored tuples. That last process is called pruning, and it eliminates records that prove useless for classification; therefore, it can also positively affect the storage issue that can plague nearest neighbors algorithms.

3.2.5 Neural Network

Neural networks receive such a name due to an analogy that is made between the structure of the learning model and the general way in which brain cells work. Neurons are processing units that transmit information between themselves via synapses. Likewise, in a neural network, there are small processing units that receive information, process it, and send it forward to other units through connections that have weights associated with them.

On the negative side, neural networks: tend to have lengthy training times, as they take a while to converge to a final hypothesis; require the setting of numerous parameters, such as the topology of their construction, that are better adjusted through empirical evaluations; and, differently from other machine learning algorithms such as decision trees and nearest neighbors, it is hard to interpret the knowledge they acquire given the nodes that make them up work like black boxes. On the positive side, they have been used successfully in numerous fields due to their ability to: deal with noisy data, which occurs when items with similar or equal inputs lead to different outputs; achieve good results with continuous-valued inputs and outputs; and handle situations where the relationship between the classes whose patterns they must learn and the attributes of the dataset is not clearly understood.

The neural network used in this work is of the feed-forward type and learns patterns via the backpropagation algorithm. Feed-forward neural networks are dubbed as so because they do not feature cycles: the information their units process is always sent forward. Figure 10 exemplifies how such networks are formed, consisting of an input layer, one or more hidden layers, and an output layer. In the image, it is possible to see a fully connected network; that is, each of the units of one layer sends its output to each of the units of the next layer.

Inputs (which are the attributes of a specific record) are placed in the nodes of the input layer. These values are sent forward to the hidden layer after being weighted. The

nodes of the hidden layer receive the weighted inputs, sum them, apply a specific activation function on the resulting value, and send it forward. Given the number of hidden layers is adjustable, that process is repeated until a final value, which – in the case of classification problem – represents a class, reaches the output layer.

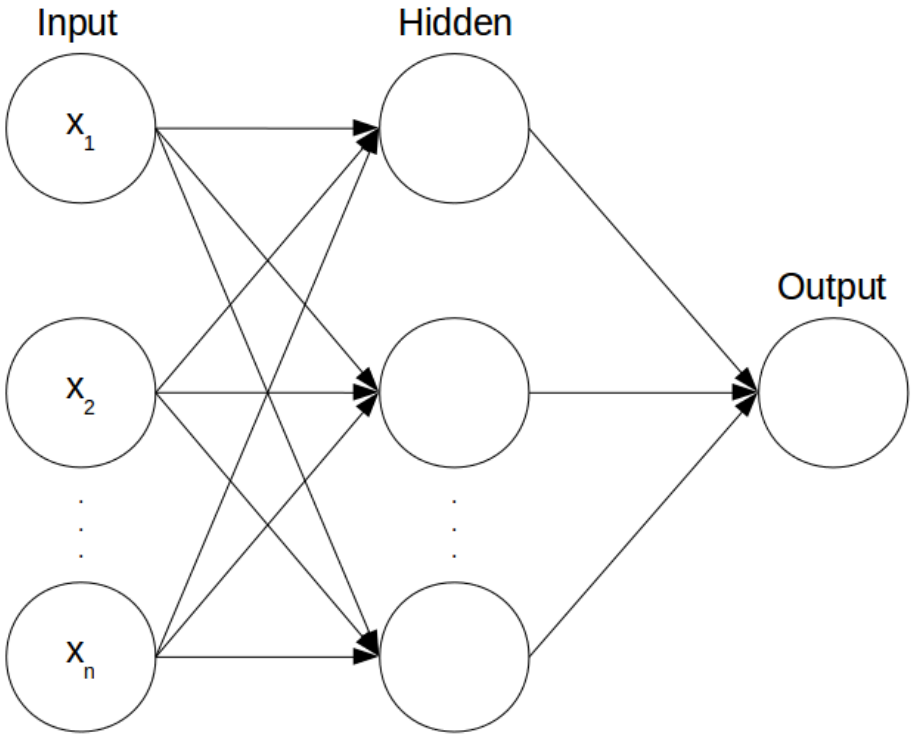

Figure 10 – Feed-Forward Neural Network

If the classification is to be done within two classes, the output layer will only feature a unit (with 0 representing a class, and 1 representing another). If the classification is to be done within more classes, the number of units in the output layer will be the same as the number of classes; therefore, the resulting class of a record will be that whose output unit produces a value equal to 1.

Despite the ability to use many hidden layers, one is usually considered to be enough in practice. The one employed in Incident Routing, for example, features one hidden layer equipped with one hundred nodes. The activation function used by the nodes was the rectifier, expressed in Formula 6.

$$f(x) = \max(0, x)$$

Formula 6 – Rectifier Function

Given the activation function is fixed, in order to learn Feed-Forward neural networks rely on the weights employed in the transformation of the values as they navigate between layers. Such weights are adjusted through the backpropagation algorithm. After a tuple is processed by the network and its class (or numerical value, in the case where numbers and not classes are being predicted) is returned, the algorithm compares the outcome with the value it was meant to return. For each tuple that is processed, the value of the weights is modified so that the desired value (the actual label of the class) is returned. Such a process is done from the output layer to the input layer; in other words, the adjustment is done backwards. The training stops if one of three conditions is met: the percentage of misclassified tuples falls below a specific threshold; the value of the adjustment of the weights falls below a specific threshold; or the entire training dataset has been run through the neural network a certain specified number of times (200, in the case of the used implementation). For a node in the final layer, the error is calculated by Formula 7, where O_j is the output of the unit and T_j is the expected result.

$$Error_j = O_j\,(1 - O_j)\,(T_j - O_j)$$

Formula 7 – Backpropagation Error for Nodes in Output Layer

For nodes in the hidden layers, the error is calculated by Formula 8. In that case, the errors ($Error_k$) of the nodes that belong to the next layer and that receive the output of the node in question are multiplied by the weight of the connection that leads to each of them, represented by W_{jk}. Once again, O_j represents the output of the node.

$$Error_j = O_j\,(1 - O_j)\sum_k Error_k W_{jk}$$

Formula 8 – Backpropagation Error for Nodes in Hidden Layer

Finally, the weight of the link connecting a node from the previous layer *i* with a node of the current layer *j* is updated by Formula 9, where O_i is the output from the node of the previous layer and *l* is the network's learning rate. In the implementation employed, the initial learning rate was set up to 0.001. As the neural network learns, though, that rate is updated through the Gradient Descent method Adam (KINGMA; BA, 2014).

$$\Delta w_{ij} = (l) Error_j O_i$$

Formula 9 – Adjustment of Weights in the Backpropagation Algorithm

3.2.6 Stochastic Gradient Descent

Suppose that, as shown by Formula 10, each record of the dataset (X_k) is represented by a vector where each component (x_n) is an attribute of the record; in the case of an incident ticket transformed with Term Frequency, those components would be the number of times a certain word appears in the text.

$$X_k = (x_0, x_1, \dots, x_n)$$

Formula 10 – Vector of Attributes

At the same time, as shown by Formula 11, consider W to be a vector where each component (w_n) is a weight.

$$W = (w_0, w_1, \dots, w_n)$$

Formula 11 – Vector of Weights

Suppose, then, that the result of the classification of a specific item is given by the sum of the multiplication of each weight and its corresponding attribute, as seen in Formula 12.

$$f(x) = \sum_{i=0}^{n} w_i x_i$$

Formula 12 – Classification Outcome

For a given set of weights, therefore, by passing all different records of the dataset through Formula 12, classification labels will be produced for all of them. These labels can then be compared to the actual expected labels and the error corresponding to the set of weights can be calculated. Such a relation between weights and classification error can be better visualized in Figure 11, where a chart displays the resulting curve.

As described in the "Learning from Data" book (ABU-MOSTAFA *et al.*, 2012), a learning model that employs the gradient descent looks to find the set of weights that, when multiplied by the values of the attributes, will produce the smallest error. In other words, the goal of the gradient descent is to find the weights that correspond to the lowest point of the curve (the lowest error). To do so, weights are initialized randomly, the error is computed, its derivative (that is, its gradient, which corresponds to the slope of that point) is calculated, and the weights are updated in the descending direction of the slope according to a specified learning rate (which determines how big of a step in the descending direction the algorithm will take).

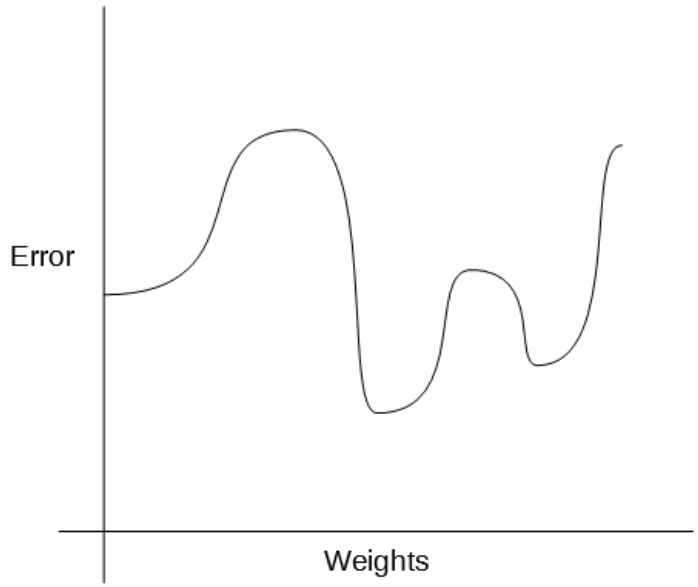

Figure 11 – Weights x Error Curve

One of the problems of the gradient descent approach is that, for very large datasets, calculating the total error of a set of weights considering all items can be costly in terms of time. That variation is called batch gradient descent, and is opposed by the stochastic gradient descent, which was the algorithm used in Incident Routing. In the stochastic gradient descent method, the error (and the gradient of the point) is calculated for only one item that is randomly (or stochastically) picked, therefore considerably decreasing the time it takes for the weights to be updated and the time it takes the algorithm to go down the slope. As the change in the weights happens based only on a single point rather than on the entire dataset, the fluctuations that occur are somewhat random; however, as data points are chosen stochastically and many iterations are performed, these fluctuations even out and the weights, on average, proceed in the right direction of the curve (ABU-MOSTAFA *et al.*, 2012).

Another question that surrounds gradient descent algorithms is the determination of when to stop the learning process. Many different stopping criteria can be used for that task, including: reaching a point where the gradient is equal to 0 (that is, the region of the curve is flat); and having the error or change in error fall below a specific threshold. These last two approaches are, however, better suited for the batch gradient descent technique, as in the stochastic mode the error available in each iteration is only that of a record, consequently not being representative of what is happening in the entire dataset. Meanwhile, stopping on a flat region does not necessarily mean a minimum point in the curve has been reached.

That is why in the implementation of the stochastic gradient descent that was used in the experiments, another stopping criterion was considered: the number of iterations, which was set to five times the size of the dataset.

3.2.7 Support Vector Machine

Support vector machines (SVMs) work towards finding a boundary that will perfectly separate data points of one class from those of another. If the items of the dataset only have two attributes, as the ones shown in Figure 12, that hyperplane will be a 1-dimensional line; if they have three attributes, the boundary will be a 2-dimensional plane; and so forth. If the points, as those on the right-hand side of Figure 12, are linearly inseparable, which means they cannot be separated by a surface in their original dimension, the SVM algorithm can, through a nonlinear transformation, map the points into a higher dimension in which they can be separated.

However, if the SVM algorithm is linear, such as is the case of the implementation employed in the experiments executed in this work, such a transformation is not done and the method will, instead, look for a separating hyperplane that will minimize the error by taking into consideration how distant the misclassified points are to the area of their target region.

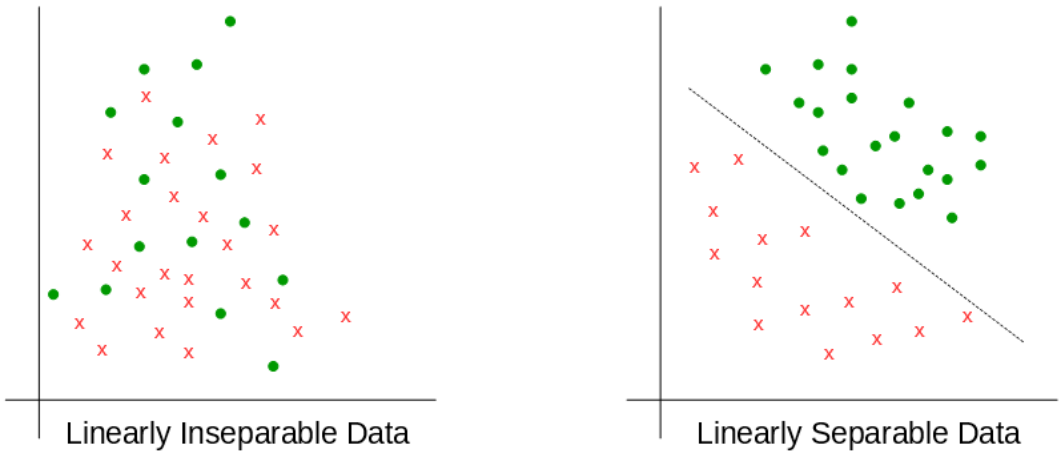

Figure 12 – Linearly Inseparable Data vs. Linearly Separable Data

If the data can in fact be separated, whether it is in their original dimension or in a higher one that is reached after a non-linear transformation, there is an infinite amount of hyperplanes that can be used to achieve perfect separation. Support vector machines

do not look for any hyperplane, though. As a machine learning algorithm, it wants to find a way to learn with the available data so that future records that are yet to be labeled are classified in the best way possible. Due to that, it looks for a special hyperplane: the one that will be the furthest apart from the data points of both classes that are the closest to it. It does so by minimizing the hinge loss, which is a loss function.

Those marginal points are called support vectors, because they are used to support the definition of the separating hyperplane. Meanwhile, the distance between those vectors is called the margin. Therefore, SVMs look for the support vectors that will maximize the margin that separates both classes. Figure 13 illustrates how SVMs work, with the support vectors chosen to form the hyperplane being highlighted in blue. On the left-hand side, the support vectors that are picked do not produce an ideal margin, for there is another hyperplane, the one on the right-hand side, that maximizes the margin between the two groups of data points.

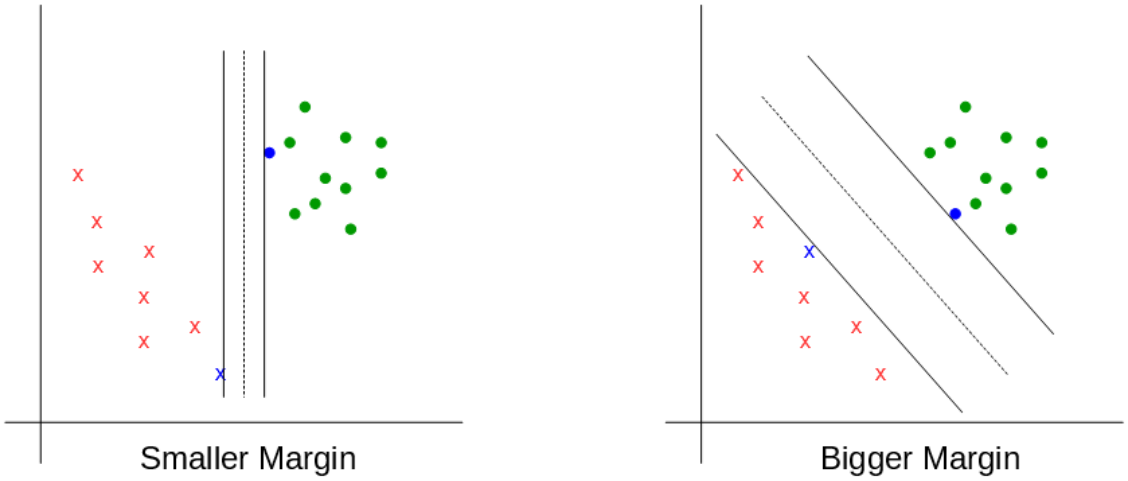

Figure 13 – Smaller Margin vs. Bigger Margin

A hyperplane that separates a set of training data points X is defined by Formula 13, where W is a vector of weights.

$$W \cdot X = 0$$

Formula 13 – Separating SVM Hyperplane

With that in mind, the hyperplanes that determine the margins (the ones that touch the support vectors of choice) can be defined by Formula 14 and Formula 15.

$$W \cdot X = 1$$

Formula 14 – Margin Hyperplane 1

$$W \cdot X = -1$$

Formula 15 – Margin Hyperplane 2

As a consequence, given an incoming data point that needs to be labeled, and supposing such a data point has two attributes x_0 and x_1, its classification will be given by Formula 16.

$$x_0\, w_0 + x_1\, w_1$$

Formula 16 – SVM Classification

If the value produced is higher than 0, the data point will be above the separating hyperplane and will belong to the class grouped in that area. If the value produced is lower than 0, the data point will be below the separating hyperplane and will belong to the other class being evaluated. All the main components of the SVM classifier, including distance *d*, which needs to be maximized, are show in Figure 14.

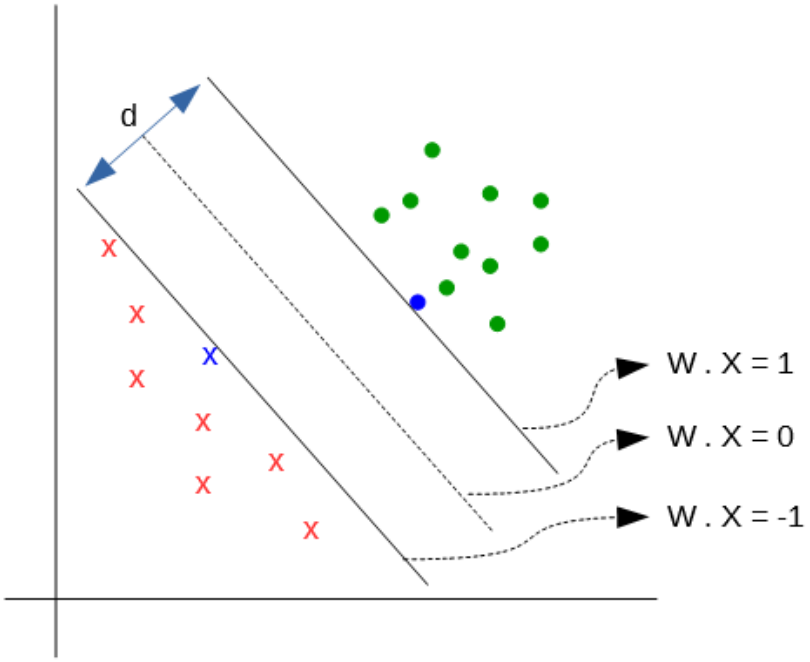

Figure 14 – SVM Illustration

Differently from the other classification algorithms used in Incident Routing, SVMs have the inherent characteristic of only being able to differ between two classes, as the hyperplanes they seek to uncover divide the *n*-dimensional space into two regions. Algorithms of that sort, however, can be adapted to be used when more than two classes are in play (as is the case of some of the classification groups that will be tackled during the experiment sections of this work). These adaptations are achieved through two approaches known as one-versus-all (OVA) and all-versus-all (AVA).

In OVA, considering a problem where *n* classes exist, n binary classifiers are trained. Suppose, therefore, that a ticket needs to be assigned to one of the three following groups: Application, Database, and Network. The three classifiers constructed will, then, be: Application vs. All (Database and Network); Database vs. All (Application and Network), and Network vs. All (Application and Database). When an unlabeled ticket arrives, it is sent to the three classifiers, which will – in turn – return the class they think the ticket belongs to. The class that will be assigned to the ticket will be the one that receives the biggest amount of votes. If the Application vs. All classifier considers the ticket to be a part of the "All" class, then both Network and Database get a vote.

In AVA, on the other hand, for *n* given classes, $\frac{n\,(n-1)}{2}$ binary classifiers are built. Then, if the classes Application, Database, and Network are, once more, the targets, three classifiers will be used considering all possible combinations of the three classes; that is, the classifiers will be: Database vs. Application, Database vs. Network, and Application vs. Network. Again, in the arrival of a new ticket that needs to be classified, it is sent to all constructed classifiers, which then vote to determine the class that will be assigned to the ticket.

The SVM classifiers used in the experiments of this work, for classification scenarios that included more than two classes, were built with the OVA approach.

3.2.8 Bagging and Bosting

Bagging (BREIMAN, 1996) and Boosting (FREUND; SCHAPIRE, 1995), like decision trees, are ensemble methods. That means that, in an attempt to improve the performance of a single classifier that labors on its own, they work by combining the results of numerous base classifiers that are constructed with different subsets of the training dataset. Differently from random forests, though, they do not necessarily have to be made up of decision trees; bagging and boosting can be employed to form groups and aggregate the votes of any classification algorithm.

For a training dataset with *n* items, bagging creates *k* (10, in the case of the implementation that was used) random subsets of *n* records. The items of the subsets are drawn with replacement (items that are chosen are not removed from the pool of selection); therefore, repeated samples can be acquired and put into the same subset. Each one of these subsets is used, then, in the training process of an instance of the classifier

that is being used in the bagging. After the training is done and the models are constructed, when a new unlabeled record arrives all of the classifiers vote in order to determine which class will be assigned to that item.

Boosting, meanwhile, starts by training a single classifier that works on the entire training dataset. After the training is done, weights are assigned to the records; those that have been misclassified by the initial classifier receive a larger weight so that subsequent classifiers that are created be more careful with data that is being incorrectly labeled. Like it happens in the case of bagging, after the training of all instances is done, new records are classified through voting; unlike bagging, though, the weight of the votes is not equal.

The variation of boosting used in the experiments of Incident Routing was AdaBoost (ZHU *et al.*, 2009), which is a popular boosting algorithm. In AdaBoost, all records start with the same attached weight. For a training dataset with *n* items, *n* records are drawn – once more with replacement – and that subset is used in the training of a model. Records that are correctly classified have their weights decreased; records that are not correctly classified have their weights increased. If ten base classifiers are to be used, that process is repeated ten times; if, however, the error rate of a classifier exceeds 0.5, it is thrown out and a new subset is drawn so that a better classifier is created. That error rate is calculated by summing the weights of all misclassified tuples, and it is taken into consideration when the weights of the correctly classified records are being updated, as shown in Formula 17.

$$New\ Weight = Current\ Weight\ \left(\frac{Error\ Rate\ of\ Classifier}{1 - Error\ Rate\ of\ Classifier}\right)$$

Formula 17 – Weight Calculation of Correctly Classified Records

Following the update of only the weights of correctly classified records, all of the weights are normalized by multiplying them by the sum of the old weights. That is how while the weight of correctly classified items decreases, the weight of incorrectly classified items increases. When it comes to the voting, the error rate of each classifier plays a part in the weighing of its vote, which is determined by Formula 18. The smaller the error rate of a classifier is, the more its vote will count.

$$Vote\ Weight = \log\left(\frac{1 - Error\ Rate\ of\ Classifier}{Error\ Rate\ of\ Classifier}\right)$$

Formula 18 – Vote Weight Calculation

3.3 Feature Selection

One of the major characteristics and difficulties of text classification as a whole is the inborn high dimensionality of the feature space (YANG; PEDERSEN, 1997). As text is processed by the text transformation techniques, the data that is being treated results in vectors that – depending on the corpus – can have tens of thousands of dimensions, each representing a term that exists in the set of texts. Word2Vec does diminish that issue to some degree, as it generates vectors of a fixed size for each word, which can then be aggregated element-wise in order to represent sentences, paragraphs, and documents.

When it comes to Term Frequency and its variations, though, large corpora that contain a lot of words will generate lengthy vectors whose high dimensionality will burden classifiers.

A naïve Bayes algorithm will have to calculate probabilities for thousands of terms; a decision tree and a random forest will need to analyze thousands of attributes when performing the split; a neural network will have thousands of nodes in its input layer, which will cause each node in the hidden layer to receive thousands of weighted values; support vector machines and the gradient descent method will perform their respective optimizations in an extremely elevated number of dimensions, having to find the best value for thousands of weights; and the lazy learner nearest neighbors will calculate distances between points in a number of dimensions that will be equal to the number of attributes.

The problem of high dimensionality is common in text classification. A corpus of tweets used for sentiment analysis (ROSENTHAL *et al.*, 2015) contained 17,152 distinct terms spread through 7,460 messages. Meanwhile, another set of texts, in this case of Rotten Tomatoes movie reviews (PANG; LEE, 2005) that was used for the same goal of identifying the sentiment (positive or negative) in the text presented 21,384 distinct terms in 10,662 analyses. In the collection of incident tickets used in this work, that problem also appears.

High dimensionality is treated through feature selection methods, which have two main benefits (SHARMA; DEY, 2012). The first one is how classifiers that work with fewer attributes (that is, in a space of lower dimensionality) deal with fewer parameters, therefore being able to converge faster. The second one comes in terms of accuracy, as – in managing fewer attributes – classifiers tend to better identify the ones responsible for

distinguishing between different classes. As a consequence, feature selection can – in a smaller amount of time – bring results that are as good as, or even better than, those obtained with classifications done with the full set of attributes, and in some cases it might even reduce the effects of overfitting (TANG *et al.*, 2014).

Given how common the problem of high dimensionality is in the domain of text classification, and due to the numerous benefits feature selection can bring, many researches and surveys comparing different approaches have been conducted in the field (TANG *et al.*, 2014) (CHANDRASHEKAR; SAHIN, 2014). Among the most popular feature selection methods, one can cite Log Likelihood Ratio (DUNNING, 1993) and Chi-Squared Test (MANNING *et al*, 1999). Particularly, for this work on incident tickets, we implement Improved Fisher's Discriminant Ratio (WANG *et al.*, 2011), which in its introductory paper obtained good results when applied to a sentiment analysis problem.

All of the mentioned feature selection methods, however, use the same basis. Randomly selecting a certain amount of features so that classifiers have less variables to deal with would be potentially non-productive, given valuable terms that could be used to differ one class from another could end up being lost. That is why Log Likelihood Ratio, Chi-Squared Test, and Improved Fisher's Discriminant Ratio attempt to identify which terms are more significant for a given class.

In particular, Improved Fisher's Discriminant Ratio achieves that goal via Formula 19.

$$F(t_k) = \frac{E(t_k|P) - E(t_k|N)^2}{D(t_k|P) + D(t_k|N)}$$

Formula 19 – Improved Fisher's Discriminant Ratio

The term being analyzed is represented by t_k. $E(t_k|P)$ and $E(t_k|N)$ are the conditional averages of t_k in relation to the classes *P* and *N*. Finally, $D(t_k|P)$ and $D(t_k|N)$ are the conditional variances of t_k in relation to the classes *P* and *N*.

The result of the formula is that the bigger the average number of appearances of a term inside the documents of a class is in relation to the average number of appearances of that term inside the documents of another class, the bigger the value for Improved Fisher's Discriminant Ratio will be. At the same time, the variances at the bottom of the fraction indicate that the smaller the difference between the appearance of a term inside

the documents of the classes is in relation to the average, the bigger the value for Improved Fisher's Discriminant Ratio will be.

Consider, for example, in a sentiment analysis problem, the terms "good" and "bad". When the importance of the term "good" is being analyzed for the class "positive", it is likely the average of its appearance in that class will be much higher than its appearance in the class "negative". The opposite applies to the term "bad". Therefore, the value in the numerator will tend to be quite high. At the same time, given it is likely the terms "bad" and "good" are quite evenly spread through the items of both classes (that is, they will show up with a certain regularity), the variances will tend to be low, further increasing the value of the numerator.

The result achieved in the feature selection as done via Improved Fisher's Discriminant Ratio is displayed in Table 3. Supposing the classification of tickets is being done for three classes (Application, Database, and Network), the importance of each term (its Improved Fisher's Discriminant Ratio) will be calculated for all classes in the one-versus-all format, which means *P* and *N* will be:

- Application vs. Not Application (Database and Network);
- Database vs. Not Database (Application and Network);
- Network vs. Not Network (Application and Database).

Three scores for each term will then be yielded. As shown in Table 3, the one that will be considered for feature selection will be the biggest one. Therefore, by ordering terms according to their maximum score, it is possible to select only a certain amount of the ones that are most useful for the differentiation between classes, allowing for the dimensionality of the vectors to be reduced, for classifiers to work with less variables and converge faster, and for terms that are not relevant for classification to be ignored.

Table 3 – Improved Fisher's Discriminant Ratio Example

Term	**Application**	**Database**	**Network**	**Maximum**
term1	0.56	0.98	0.32	0.98
term2	0.12	0.71	0.76	0.76

Improved Fisher's Discriminant Ratio was introduced in two variations: binary and frequency. In the first, it is only taken into account whether the term exists in the text

(1) or not (0); in the second approach, meanwhile, the number of appearances of the term in the text is counted. Both approaches will be tested in this work.

3.4 Imbalanced Datasets

Just like high dimensionality, imbalanced datasets can have negative effects on text classification. Class imbalance refers to cases where one class is highly represented in the dataset when compared to others; such a distribution can cause classifiers to become biased (WANG; YAO, 2012), accurately making predictions for samples of the majority class while failing to have good enough results for the minority categories. Such a phenomenon occurs because, in order to adjust the hypothesis and reduce misclassifications, classifiers use error metrics. Therefore, using a hypothesis that performs very well when it comes to the majority samples while failing to appropriately learn the patterns of minority classes has positive effects on those metrics (GALAR *et al.*, 2012). Moreover, a low number of examples from one class may also mean there is just not enough information available for any sort of clear differing pattern between the classes to be discovered by the algorithms. The solving of the problem of imbalanced data, thereby, is the obtaining of a classifier that is able to learn how to properly classify items from both minority and majority classes.

Imbalanced data occurs in various domains in which machine learning is applied. Imbalance of the ratio of 10,000 to 1 has been reported in problems of fraud detection (CHAWLA *et al.*, 2002); and the issue has also appeared in oil-spill detection through satellite images (CHAWLA *et al.*, 2002), credit scoring (BROWN, 2012), medical diagnosis (MAZUROWSKI *et al.*, 2008), and face recognition (LIU; CHEN, 2005). The problem of incident ticket forwarding explored in this work also features an imbalanced dataset, albeit not to the degree of the 10,000 to 1 rate encountered in fraud detection, where normal financial transactions far outnumber fraudulent ones.

Due to the widespread nature of the problem, many approaches for dealing with imbalanced data have been proposed. They can be grouped into three categories (FERNÁNDEZ *et al.*, 2013):

- **Data Level Solutions:** In this case, the goal is to either eliminate samples of the majority class, or create or replicate samples of the minority class. As a

consequence, a dataset that is originally imbalanced can be artificially made to be more balanced.

- **Algorithmic Level Solutions:** Here, the adaptation is not performed on the data, but on the classification algorithms. They are changed in order to give more focus to the minority class.
- **Cost-Sensitive Solutions:** Through this approach, algorithms are forced to pay more attention to minority classes because higher misclassification costs are applied to their samples.

Data level solutions have the advantage of being versatile (FERNÁNDEZ *et al.*, 2013). Given they act upon the very basis on which classifiers are constructed (the data), rather than on the algorithms themselves, they can be applied to the dataset and subsequently be tested with different algorithms and mixed with other approaches. Additionally, if a classifier to deal with the data has already been picked, numerous data level solutions can be tested in order to try to improve the classification performance once it is noticed the algorithms are performing poorly on imbalanced data.

These data level solutions are further divided into three groups of techniques: oversampling, undersampling, and hybrid methods. In oversampling, samples of the minority classes are either replicated or synthetically created through different approaches. In undersampling, samples of the majority classes are selected and eliminated. Finally, in hybrid methods, a combination of oversampling and undersampling occurs.

Historically, oversampling techniques have obtained better results than undersampling and hybrid methods (BARUA *et al.*, 2014). It is believed that this advantage occurs because undersampling causes data that can be valuable to the learning of patterns from the majority class to be ignored, therefore negatively impacting classification (YAP *et al.*, 2014). The superior performance of these methods was attested in a study (BATISTA *et al.*, 2004) in which ten data level solutions (belonging to all categories) were applied to fifteen imbalanced datasets. For all of the 15 datasets, the best result was obtained by one of the following four methods:

- Random Oversampling, an oversampling technique;
- SMOTE (CHAWLA *et al.*, 2002), an oversampling technique;

- SMOTE-ENN (BATISTA *et al.*, 2004), a hybrid technique;
- SMOTE-Tomek Links (BATISTA *et al.*, 2003), a hybrid technique.

Random oversampling is the most straightforward approach of the group. It randomly selects, with replacement, samples from the minority classes and replicates them. The replication is done until the number of items in the minority classes is equal to that of the majority class.

SMOTE (Synthetic Minority Over-sampling Technique) operates in the feature space. It creates synthetic samples by selecting points of the minority class, identifying the *k* nearest neighbors to that point that are also of the minority class, and creating a new example at a random part of the line segment that connects two data points. The value *k* (the amount of nearest neighbors to be considered) is calculated according to the rate of oversampling that needs to be done for the minority classes to have the same amount of samples as the majority class. Mathematically, new points are created through the following operations: the difference between the vectors that represent the data points of two nearest neighbors is calculated; that difference is multiplied by a random number between 0 and 1; the result is added to the data point that is under consideration.

The hybrid technique SMOTE-ENN, meanwhile, combines SMOTE (an oversampling algorithm) with ENN (which stands for Edited Nearest Neighbors and is an undersampling approach). ENN works by eliminating all points (independently of whether they are from the minority or majority class) that are misclassified by its three nearest neighbors. Differently from a traditional nearest neighbors approach, though, the votes of the neighbors are not counted equally: they are weighted according to how distant they are from the point that is being analyzed. The closer it is, the more weight its vote will carry. That weight is given by the inverse of the distance. Once more, the techniques SMOTE and ENN are applied until the number of points of all classes is the same.

At last, the hybrid approach SMOTE-Tomek Links employs SMOTE and the undersampling technique Tomek Links. Tomek Links (TOMEK, 1976) occur when two points of different classes are the nearest neighbors to each other. If two examples form a Tomek Link, they are either noise or borderline samples (BATISTA *et al.*, 2004). Differently from the ENN technique, Tomek Links (when used for undersampling rather than for data cleaning, which is an area where they can also be employed) only eliminate

samples from the majority class. SMOTE and Tomek Links are performed in conjunction until the number of items of all classes is equal.

Due to the outcomes obtained by these oversampling and hybrid methods in the study (BATISTA *et al.*, 2004) that used them in multiple imbalanced datasets, they were chosen to be applied to the imbalanced dataset of incident tickets of this work. The implementations used were those of the Imbalanced Learn Python package (LEMAITRE *et al.*, 2017). Figure 15 presents an illustration of SMOTE, ENN, and Tomek Links. Blue data points are the ones being created (in the case of SMOTE) and eliminated (in the case of ENN and Tomek Links).

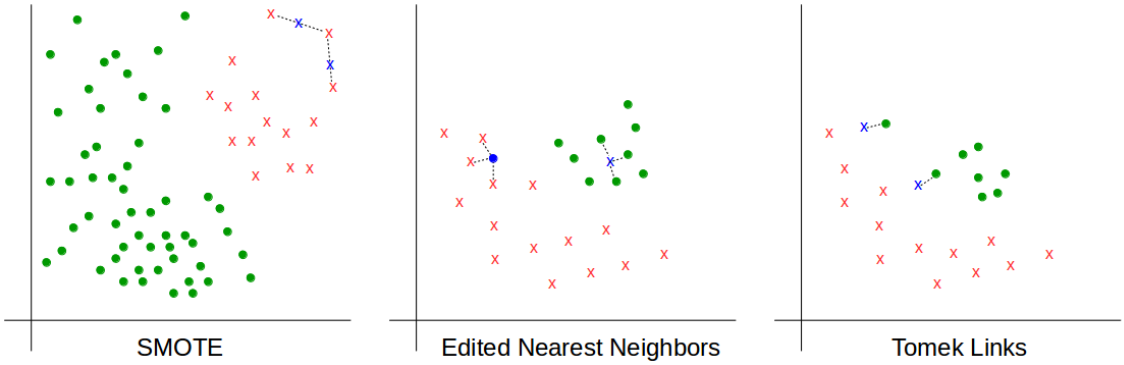

Figure 15 – SMOTE, Edited Nearest Neighbors, and Tomek Links

Additionally to the data level, algorithmic, and cost-sensitive solutions to imbalanced datasets, recent works show that ensembles have achieved good results in classifying data that is imbalanced (GALAR *et al.*, 2012) (BARUA *et al.*, 2014).

Bagging stood out when matched up with undersampling and oversampling techniques, especially SMOTE (GALAR *et al.*, 2012). Boosting, meanwhile, especially in its AdaBoosting variation, performed well without data balancing (WANG; YAO, 2012), which can be attributed to the fact boosting works by giving more weight to misclassified records so that subsequent instances of trained classifiers pay more attention to data that is not being correctly labeled. As such, boosting qualifies as a cost-sensitive solution for dealing with imbalanced datasets. Another study indicated that, with data balancing, boosting was able to improve the performance of four tested oversampling techniques (BARUA *et al.*, 2014).

Given those evidences, besides the selected data balancing approaches (Random Oversampling, SMOTE, SMOTE-ENN, and SMOTE-Tomek Links), boosting and bagging were also tested in Incident Routing as solutions to deal with the class-imbalance problem.

3.5 Concept Drift

Concept drift happens in non-stationary environments, in which data distribution can change over time (GAMA *et al.*, 2014). In e-commerce applications, for example, concept drift means the preference of customers can change as time passes; in the field of weather prediction, the rules may vary according to the season (TSYMBAL, 2004); and in the case of users who are following an online stream of news, their interests can mutate with the passing of hours, days, weeks, and months (GAMA *et al.*, 2014).

If such changes are not somehow taken into account by classifiers, they may end up learning with data that is no longer representative of the distribution that is currently in place. In the example of e-commerce applications, suppose there is a recommender system (RESNICK; VARIAN, 1997) that is making recommendations to users regarding items they may be interested in. Such recommendations are made by taking into account items they have either bought or visited; the system looks to those items, checks other users who have visited those pages, evaluates what those users have bought or visited, and sends these purchased or viewed items as recommendations to the target user.

In that scenario, concept drift can come into play, for example, in changes that occur in the user's life. Suppose that one month ago that user lived with his parents, and his purchases gravitated around entertainment items such as games, music, and movies. But now that user has moved out and has drifted his focus towards furniture and appliances. Recommending new games, therefore, may not be as interesting and valuable for the system (and the user) as it used to be; and the recommendation of a dinner table, something the user would unlikely be interested in just a few days in the past, may be preferable.

As a consequence, the system needs to adapt to that change. Although there are records that indicate the user visited and bought many entertainment-related items in the past, considering them when it comes to calculating recommendations may no longer be a good idea; perhaps it would be better to ignore, or give less weight to, those page visits and product purchases and focus, instead, on their recent behavior of taking a look at new pieces of furniture and ordering appliances.

In general concept drift occurs due to change in variables surrounding the analyzed problem (TSYMBAL, 2004), such as the user moving out in the case of a recommender system of an e-commerce page. As, in many occasions, such contexts are

not mapped to the features that are fed into the algorithm, these contexts are called "hidden contexts", which makes the detection and treatment of concept drift in real-world applications a tough task. For the e-commerce store, that means there is no information in its system that indicates something in the life of the user has changed; yet, it must find a way to react to the concept drift.

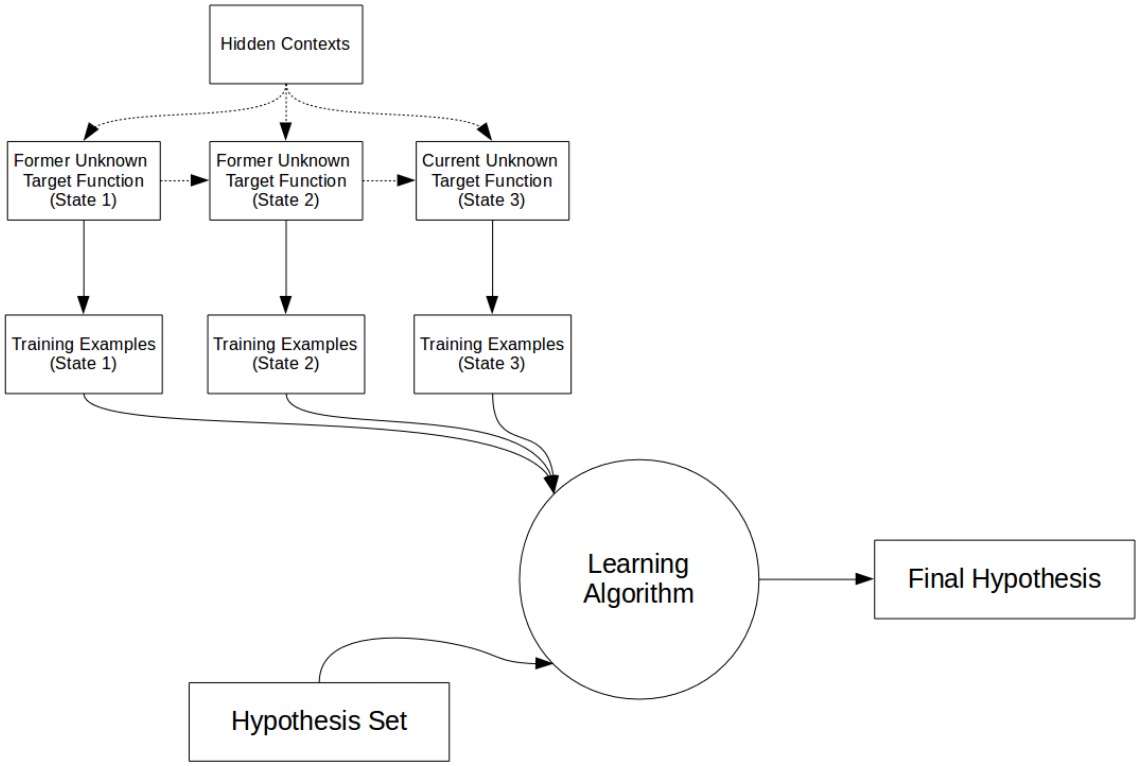

Figure 16 – The Learning Problem With Concept Drift

By looking back to the learning problem, as presented in the "Learning from Data" book (ABU-MOSTAFA *et al.*, 2012), concept drift means that the target function the machine learning algorithm is trying to approximate changes as time moves on, as it is influenced by changes in the environment in which it exists. As, in the case of a supervised learning problem, machine learning algorithms learn from data, the data that is being used in the learning may not be the product of the function in its current state. Figure 16, which adds a concept drift variable to the learning problem image adapted from the "Learning from Data" book, illustrates that issue.

In Figure 16, affected by hidden contexts, the target function changes two times. As, by using the attributes it has for each record it needs to classify, the classifier does not know the target function it is trying to approximate has changed, it continues to use training records that were produced by past unknown target functions. Given the

classifier's objective is to diminish error by getting as close as possible to the current unknown target function, considering training examples of previous functions when finding patterns in the data is not helpful.

Take the supervised learning problem of spam-filtering (SHEU *et al.*, 2017), in which concept drifts are known to occur. The goal is to separate incoming e-mails into two classes (spam or not-spam), and the attributes sent to the classifiers so they can learn what is and what is not a spam are the terms found in the e-mail (nothing in the used set of features indicates changes in the context of the target function). Spam creators, however, do react to spam filters and try to overcome them by changing the patterns employed in the production of spam. Classifiers, therefore, need to find ways to adapt to those new patterns, and old patterns produced before the current spam-creation function was in place may not be useful for classification.

A recent survey in the area of concept drift (GAMA *et al.*, 2014) presented four distinct ways in which the phenomenon can occur. Figure 17, which is adapted from that work, shows those types of concept drift.

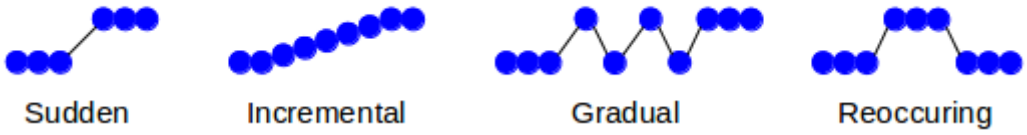

Figure 17 – Types of Concept Drift

Hence, concept drifts can be sudden (one day the user of a music-streaming platform abruptly develops a taste for rock and abandons the pop genre they used to listen); incremental (that user slowly moves from pop to rock by slowly adding rock songs to their playlist until a point in which they completely drop the former style); gradual (before completely leaving pop behind, the user alternates streaks of rock-listening days with days when they go back to pop); or reoccurring (where periods that lean towards rock and pop keep on coming and going).

For incident tickets, concept drift can come into play in many ways. Firstly, as the technology supporting the IT services changes, applications are replaced, database features are introduced, and all layers of the IT infrastructure are updated, the features (in other words, the terms) present in the tickets may be altered as well. Technology that once indicated a ticket may pertain to an area may no longer be used; systems that were frequently used may be replaced by other applications with different names; and given

the fast-changing nature of the technological scenario in which IT infrastructure exists, new terms may appear with a certain frequency.

Secondly, changes can also happen regarding the employees of the area responsible for creating the tickets. Different employees have different writing styles and find distinct ways to present the nature of the incidents they are reporting. As time passes, and employees retire, leave the company, or start working in a new area, the team that works as the centralized entry point for tickets and that is responsible for logging the information into the system may change entirely. Moreover, as management changes, shifts in the writing style or pattern of the tickets can also occur, as ticket-writing practices may be put in place to better control the quality of what is reported to the specialized areas.

Finally, tickets that are created during a specific time-window may be closely related to one another or have their origins traced to the same event (SALAH *et al.*, 2016). When new applications or new technologies are deployed, or when an update of those items occurs, the amount of tickets related to them may increase.

For those reasons, which are hidden contexts because they exist outside the attributes (the terms in the tickets) used for ticket classification in Incident Routing, the treatment of the concept drift problem may bring better classification performance than in scenarios in which it is not treated.

3.5.1 FLORA Algorithm

Adaptive learning is the name given to a series of techniques used to update predictive models during their operation in order to allow them to react to concept drift (GAMA *et al.*, 2014). Since the dynamically changing environments that affect target functions and make them drift are common in numerous areas (ZLIOBAITĖ, 2010), various different solutions have been proposed. One of the first systems developed for handling that problem was the FLORA family of algorithms (WIDMER; KUBAT, 1996).

As shown in Figure 18, which is adapted from the work that introduced the FLORA family of algorithms (WIDMER; KUBAT, 1996), the original FLORA algorithm works with a learning window of fixed size *n*. Only examples that are inside the window are considered in the building of the predictive model; examples that are outside it are ignored. At predetermined time steps, the window moves forward and the model is

reconstructed (that is, a new final hypothesis that approximates the target function is found by taking into account only the data that is inside the window).

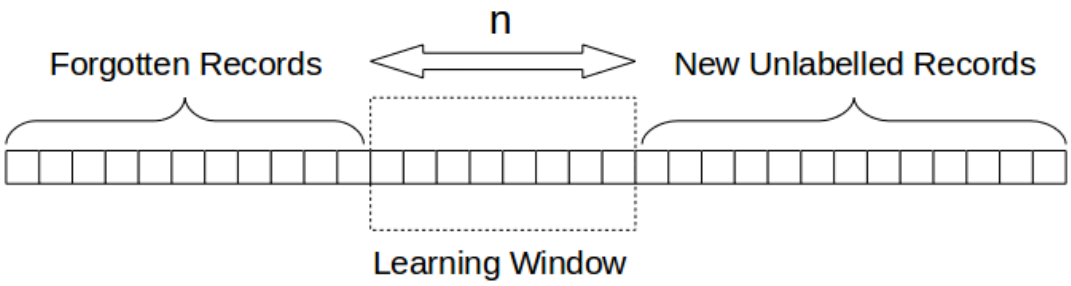

Figure 18 – FLORA Algorithm

The challenge of that approach lies in how long the window needs to be (GAMA *et al.*, 2014). If the window is too big, it will react more slowly to changes in the unknown target function; at the same time, if the data distribution is going through an extended period of stability (in other words, no significant concept drift has occurred), longer windows will tend to perform better. Shorter windows, conversely, reflect changes in the unknown target function more quickly; however, they will tend to perform worse when the data distribution has been stable for a period that is longer than the window size that is being used.

The FLORA algorithm has a few variations: FLORA2, for example, works with an adaptive and variable time window; FLORA3 focuses on handling reoccurring concept drifts; and FLORA4 deals with noisy data. Likewise, many of the techniques used when concept drift is taken into consideration try to detect changes to either adapt their learning windows; or alter the weights of the items, in cases where the approaches instead of ignoring older data – such as FLORA does – use all of the available examples in the learning but give less importance to the older ones.

In the experiments performed for the purpose of this work, though, the aim was to treat concept drift in the domain of incident tickets while observing how training windows of varying size behave on the dataset in order to not only verify if concept drift does indeed happen in the area but also evaluate if shorter training windows are beneficial to classification performance. As such, the original FLORA algorithm was chosen.

3.5.2 Proposed Ensembles

In Incident Routing, the windows of the FLORA algorithm will be set up in the scale of months. That is, for the classification of the tickets of a given month, one tree of classifiers (as the one see in Figure 4) will be trained only with the records of the previous month; another tree of classifiers will be trained with the records of the previous two months; and so forth. As they are trained with data from distinct periods, they will produce distinct final hypotheses, which will – in turn – differ in their performance.

Given, however, that both shorter and longer windows have their own advantages and disadvantages; under the light of the treatment of concept drift, this work aims to improve the performance of the individual models that learn with windows of varied size by aggregating their results. Such a task is achieved by the creation of ensembles, which produce predictions that are known to be better than those of a classifier that works on its own (LÖFSTRÖM, 2015).

In the creation of these ensembles, each classifier of the tree shown in Figure 4 will be made up of multiple classifiers (each learning with a specific window). The class to which a ticket belongs to will be determined through different voting schemes that will take into account the results obtained from all learning windows.

In the field of adaptive learning, the use of ensembles is not uncommon; their deployment is usually attached to the notion that, during a concept drift, data is generated from a combined distribution that mixes elements of the original target function with elements of the new target function that will be the destination of the concept drift (GAMA *et al.*, 2014). As a whole, concept drift ensembles have been arranged into three general groups (GAMA *et al.*, 2014):

- **Dynamic Combination:** In which base learners are trained in advance and the rules establishing the combination of their results are dynamically changed to respond to changes in the environment.
- **Continuous Update:** In which the learners are either retrained online or in batch mode using the new data.
- **Structural Update:** In which new learners are added to the ensemble while poorly performing ones are removed.

The ensembles proposed in this work belong to the first two categories. All of them are of the continuous update type; in all cases, the classifiers that compose them receive continuous updates executed in batch: as a new month comes, the window of each classifier moves one month forward so tickets from the latest month are taken into account in the construction of the predictive models while those of the month that has fallen out of the window are ignored.

Meanwhile, only some of them are of the dynamic combination type. The combination rules that define their voting schemes are dynamic, changing according to the recent performance of the classifiers

The proposed ensembles that belong only to the continuous update group are:

- **Unweighted Voting:** The class predicted by each classifier counts as one vote, regardless of the size of its window and its recent performance.
- **Weighted by Shortest Window:** The shorter the learning window of the classifier is, the more weight will be given to its vote.
- **Weighted by Longest Window:** The longer the learning window of the classifier is, the more weight will be given to its vote.

The remaining three ensemble strategies have the weights of the votes of the classifiers that form them altered according to their performance on the previous month. Therefore, not only are they continuously updated, but the strategy used for the combination of their results is dynamic.

These next three proposed ensembles require that a ticket be classified not only by the tree of ensembles itself (where each of the five classifiers of Figure 4 is made up of classifiers that learn with different windows) but by individual trees where all five classifiers work with a specific learning window. In all cases, the ticket will be forwarded to the area indicated by the tree of ensembles. However, the individual trees where all five classifiers work with a certain learning window (in one tree, all classifiers learn with one month; in another, all classifiers learn with two months; etc.) are required so that the performance of the windows can be evaluated individually, which – in turn – allows the weighting of the votes in the tree of ensembles. These three ensemble strategies are:

- **Weighted Monthly:** In this ensemble if, for example, the tree that trained with only one month was the one that had the best performance during the previous month, for all classifiers of the tree of ensembles, the classifier that considers only one month of the data in its training will have the most significant vote. Therefore, for all five classifiers that form the tree of ensembles, the weight of the votes is the same. The weight of the votes is given by the performance ranking of the individual trees in the previous month: the better the performance, the larger the weight.
- **Weighted by Classifier Accuracy:** In this ensemble, for all five classifiers that form the tree of ensembles, the weight of the votes is different. That happens because the performance is evaluated by the accuracy obtained by the individual trees on each classifier that forms the tree seen in Figure 4. If, for example, the individual tree that only employed one month in its learning step had the best performance in terms of accuracy for classifier L0 vs. Non L0, its vote will have the largest weight for that classifier; at the same time, if that individual one-month tree had the worst performance in terms of accuracy for classifier L1 vs. L2, its vote will have the smallest weight for that classifier. The weight of the votes is given by the performance ranking in the previous month: the better the performance, the larger the weight.
- **Weighted by Classifier Total:** This ensemble works similarly to the Weighted by Classifier Accuracy strategy; the difference, as the name implies, is that the metric that determines the weight given to the votes of the classifiers that make up the ensembles is the total of correctly classified tickets obtained by the individual tree for that classifier, and not the accuracy. The weight of the votes is given by the performance ranking in the previous month: the better the performance, the larger the weight.

The difference in the use of the metrics accuracy and total of correctly classified records stems from how the classifiers proposed for Incident Routing work as a tree. For a classifier that works on its own, that is, in the case of the dataset used for Incident Routing, for a classifier that tries to learn patterns for thirteen classes, those metrics would produce the same rank. In other words, if a classifier that works with a one-month window has the best accuracy it will also be the one that correctly classifies the biggest amount of tickets. In a tree structure, however, that does not hold, and the ranks can differ.

That happens because, for example, the number of incident tickets that reaches classifier L2 depends on the performance of classifiers L0 vs. Non L0 and L1 vs. L2. Consider, for example, a tree of classifiers learning from data of the previous month (called K1) and a tree of classifiers learning from data of the previous two months (called K2). Suppose that the classifier L2 in K1 receives 100 records, while the classifier L2 in K2 receives 200 records.

If L2 in K1 correctly classifies all records, it will have 100% accuracy and 100 correctly classified records. If L2 in K2 has an 80% accuracy – that is, lower than the one from L2 in K1 – it will have correctly classified 160 records. The ranks, therefore, differ: while, for classifier L2, K1 performed better in terms of accuracy, K2 did better in terms of the total of correctly classified records. As such, in the case of a tree structure as the one from the experiment, the approaches Weighted by Classifier Accuracy and Weighted by Classifier Total differ from one another.

Finally, for all five ensembles that involve weighted voting, the weighting scheme is simple. Given a number *N* of classifiers of varying windows, the weight of the votes is inversely proportional to the rank. For example, if 7 classifiers are involved, as is the case in the experiments, the vote of the top-ranked classifier will be worth 7 points; the vote of the second-ranked classifier will be worth 6 points; and that proceeds until the vote of the last-ranked classifier, which is worth 1 point.

Chapter 4

Related Works

4.1 Incident Ticket Management

Because of the importance of IT service providing and the critical nature that infrastructure failures can have if not treated in an efficient way, as they can cause vital applications and technological resources to be inoperative for long periods of time (hence affecting business negatively), many studies have tackled the matter of incident-ticket handling.

As service desks are usually divided into teams that are formed according to their level of expertise, when new incident tickets arrive, a decision must be made regarding which level will treat them. One work (PALSHIKAR *et al.*, 2012) used statistics-based algorithms to execute right-shifts or left-shifts with the tickets. A right-shift is the sending of a ticket from a team with a lower level of expertise to a team with a higher level of expertise (that is, the ticket is moving down in the structure of the service desk); and a left-shift is the opposite. Such shifts may occur due to a number of reasons. Firstly, and intricately connected with ITIL's proposed service desk architecture, shifts can be related to the tickets' own level of difficulty; after all, tickets that are harder to deal with and cannot be resolved by the expertise of the current level need to be escalated. Finally, shifts can also happen due to reasons that are external to the service desk, such as costs (as a less experienced team is less expensive, and should therefore be focused on less critical tickets) or time (as a more experienced team will solve tickets much faster than a less experienced one).

Likewise, another work (MOHARRERI *et al.*, 2016b) set out with the goal of improving key ITIL metrics that look to measure service-providing quality. In that research, the metrics that were targeted for improvement were: Mean-Time-To-Resolve, which evaluates the average time it takes for incidents to be resolved after they have been initially reported; Service Level Compliance, which indicates the percentage with which Service Level Agreements are being respected; and Mean-Steps-To-Resolve, a metric that counts how many times a ticket has been forwarded between service desk areas

before being closed. To reach that goal, the research treats the problem of routing the incident tickets through a Collective Expert Network (CEN). A CEN is defined as the network of experts the ticket goes through before being closed; as so, differently from the tickets used in this work, which are sent to an area and closed, tickets used in that research move between areas. Therefore, the research looks for a way to recommend the appropriate sequence of experts that must treat every ticket. In order to do so, it builds a two-level classification framework. In the first level, it determines the paths through the Collective Expert Network that are most commonly used (such a task is performed via Process Mining techniques that show the path followed by each ticket and produce statistics on their behavior). After narrowing down on the most common paths, in the second level of the classification framework it uses text processing and machine learning techniques to, first, extract data from the ticket and, then, determine which one of the most commonly used paths the ticket will need to take in order for the problem it describes to be solved on time.

In the same area of Expert Networks, with the use of TF and a probabilistic model to recommend reliable routes for new tickets, researchers (MIAO *et al.*, 2012) attempted to minimize the expected number of steps a ticket takes to reach the correct level. Their proposed model, called Optimized Network Model (ONM), uses maximum likelihood estimation to calculate, for each node of the Collective Expert Network, the most reliable transfers according to the content within the ticket. With that, for each new ticket, it is able to evaluate all possible routes to resolvers and estimate the one that will be more optimal. By succeeding in doing so, the work avoids the bouncing around of tickets, as different specialized areas struggle to find out who is actually responsible for taking care of the incident. In critical scenarios, such events may be rather problematic in the keeping of the agreed service levels; with reliable routing, not only is that issue diminished, but the time it takes for tickets to be resolved is also minimized.

While most research in the area of incident ticket handling focuses on the recommendation of transfers between experts so tickets are treated efficiently enough to preserve the agreed service levels, there has been some research (MOHARRERI *et al.*, 2016a) focusing on estimating the time it will take for a ticket to be solved, since that is a ticket characteristic that is closely related to the meeting of service agreement levels. Similarly, another study (KIKUCHI, 2015) used TF-IDF and machine learning to solve one of the biggest issues regarding incident management: the balanced allocation of

incident tickets among the professionals who are responsible for treating them; as it attempts to identify the workload that will be necessary to treat each ticket according to the update history of similar incident records. The article seeks to solve that problem because, as it states, efficient incident handling is important due to how it reduces system management costs and achieves efficient incident management. The workload of each ticket is estimated through its update history. Tickets that have already been solved are labelled as one of two categories: easy or hard, the differentiation between both is done by a predetermined threshold in the number of updates. The text of tickets is transformed via TF-IDF and the Naïve Bayes algorithm is used to learn both classes; then, new tickets are analyzed by the learned predictive model, which indicates whether they are more likely to be easy or hard. With the tickets' workloads automatically estimated, it is possible to manage them with more certainty; in addition, the effort spent in analyzing the ticket and trying to determine how hard it will be to solve it is eliminated.

TF and machine learning are also used in a work (BOGOJESKA *et al.*, 2014) where a multiclassification problem with incident tickets was explored. In it, tickets were classified into five distinct types according to the issues they were related to: server unavailable, disk, performance, non-actionable, and other.

Given the considerable amount of incident tickets that are generated due to both the automatic monitoring of IT infrastructure and the quantity of users that are interacting with the systems it supports and experiencing problems, many tickets share resolutions that are either similar or identical. For that reason, one article (ZHOU *et al.*, 2015) developed a framework that includes text processing and machine learning to recommend resolutions to incoming tickets. Given how different words are used to describe similar problems, a feature adaptation algorithm was also employed to identify these words and consider them as a single entity.

In order to deal with the same issue of the high volume of tickets service desks need to take care of, another article (JAN *et al.*, 2015) used text processing and clustering to group tickets according to their causes, so that it can be viable to identify repeating patterns as well as pain points that, if treated more carefully, could end up reducing the amount of tickets that is being generated. In order to do so, the article builds a framework in which it extracts keywords from the text of tickets opened by users, groups them into clusters, and then allows analysts to use a search mechanism to better understand them.

To try to improve the work of the service desk, one research (SALAH *et al.*, 2016) uses the correlation between the tickets as metric to group several of them together. Many tickets are sometimes created for one incident, given one infrastructure issue can affect a lot of users, who will then report it to the service desk. The experiment, then, attempts to automatically summarize the information the tickets related to the same incident into just one, therefore avoiding redundancy and unnecessary extra work.

Finally, this work is closely related to another thesis (MENEZES, 2009) developed with the goal of routing incident tickets through the service desk of another Brazilian company with a vast array of information technology services, which are essential for its business. In that thesis, text transformation and classification techniques were evaluated so that the best ones could be chosen to rout tickets through a three-level service desk. The thesis reported a decrease in the total workload of 85%; that result was based on the proposed forwarding cost metric, which will also be employed in this work to evaluate how Incident Routing will bring gains to the service desk that is being studied.

4.2 Treatment of Concept Drift

Two recent surveys that explored the problem of concept drift and mapped the current state of its surrounding research (ZLIOBAITĖ *et al.*, 2010) (GAMA *et al.*, 2014) organized works that deal with real life problems that present concept drift into four groups:

- Monitoring and Control;
- Decision Making;
- Artificial Intelligence;
- Personal Assistance and Information.

Monitoring and control applications treat the detection of anomalous behavior in various areas, from the internet and other telecommunication channels to financial services; decision making involves predictive tasks that help a company's management define tactical and strategic paths to be taken; artificial intelligence englobes moving and stationary systems that need to adapt to changes in the environment in which they operate; and personal assistance and information solutions, a category under which this work is

filed, is concerned with recommender systems and the classification of information (be it textual or not) into different bins.

In monitoring and control scenarios, concept drift has been studied in the online mass flow prediction of an industrial boiler (PECHENIZKIY *et al.*, 2010), which improves the boiler's control; shifts in context happen when boiler operators are changed, as the operation process is not standardized, or when the fuel that is used presents alterations in quality and type. Other monitoring and control scenarios include intrusion detection (LANE; BRODLEY, 1999), the identification of unauthorized use of mobile terminals by monitoring the behavior and environment of the current user and then matching those with the behavior and the environment of the legitimate user (MAZHELIS; PUURONEN, 2007), and the evaluation of financial transactions to check if they are fraudulent (BOLTON; HAND, 2002). In these settings, concept drift appears in how those trying to break into these systems change their behavior and look for new strategies once they realize the effectiveness of their old approaches has been nullified due to advances in the security features protecting those systems.

Electric power generation presents two examples of how the consideration of concept drift can lead to stronger decision making systems: smart grids, for example, which use data collected from sensors to provide electricity to customers in more efficient ways can forecast consumption patterns (ALBERG; LAST, 2017) (which may change due to factors such as the weather) so that companies from the sector can plan their resources and balance supply and demand; additionally, and with the same goals, the power generated by wind parks can be predicted according to the ever-changing characteristics of the wind (BESSA *et al.*, 2010). Outside of that sector, concept drift in decision making also appears in bankruptcy prediction (where shifts from normal to crisis conditions are considered) (SUNG; CHANG; LEE, 1999), biometric recognition systems that take physiological changes into account (POH *et al.*, 2009), and the treatment of patients (BLACK; HICKEY, 2004), which needs to consider changing factors such as the stage of the disease's progression and the resistance of pathologies to antibiotics (TSYMBAL *et al*, 2006).

In artificial intelligence (AI) systems, meanwhile, changes in context have appeared in research that tries to make the AI in digital games adapt to the actions of players (COWLEY; CHARLES, 2016); as well as self-driving cars (BOJARSKI, 2016), smart homes (DINATA; HARDIAN, 2014), and robots (HAASDIJK *et al.*, 2014), which

have to learn how to react to the ever-changing conditions of the environment that surrounds them.

Regarding personal assistance and information solutions, more specifically the text-analysis subarea into which this work fits, adaptive learners have been used in spam detection (SHEU *et al.*, 2017), as spammers are constantly trying to find ways to change both the text and presentation of their e-mails in order to overcome spam filters; the recommendation of new product features via the analysis of feedback found on social networks (MIRTALAIE *et al.*, 2017); the categorization of tweets into distinct topics in order to evaluate trending subjects (LIFNA; VIJAYALAKSHMI, 2015), which is achieved via learning models based on sliding windows such as the ones used in this work; and sentiment analysis (GUERRA *et al.*, 2014), where one of the major problems in treating concept drift, and an issue that is also present in many other domains within text-analysis, arises from how in order to update the classifier the recent data needs to be correctly labeled.

In the field explored here (incident ticket management), such a problem is minimized due to how new tickets are automatically and accurately labeled as they are solved by a specific technical area. With that, they can be instantly fed into the classifiers, which learn with updated data.

Chapter 5

Experimental Evaluation

The experimental evaluation done in this work is divided into two parts. In the first one, the classification of the tickets is refined via the selection of the best methods for text transformation, text classification, feature selection, and treatment of imbalanced datasets. In the second one, with the use of a base classifier, the presence of concept drift in the field of incident ticket management is evaluated. Before going into the experiments, though, common points between both are discussed: the general format of the dataset, and the evaluation metrics that were used.

5.1 Dataset

The dataset used in the experiment was obtained through the collection of closed incident tickets. For each set of experiments, however, the period of collection and, consequently, the number of acquired tickets was different.

Table 4 – General Dataset Information

Experiment	Period of Collection	Number of Tickets
Classification Refinement	04/24/2016 – 04/24/2017	38,933
Concept Drift	04/24/2016 – 07/24/2017	50,755

The collected incident tickets featured the fields: incident identification, description, summary, symptom, affected configuration item, opening date, closing date, priority, and area. For the purpose of the experiment, the description and summary fields (as well as, naturally, the ID of the tickets and the area that solved them) were considered. That choice was made because upon the opening of the incident (which is the moment when it must be forwarded to the responsible area) the fields symptom, affected configuration item, and priority are not mandatory; therefore, the information they contain may have been filled up after the tickets were sent to their respective areas.

These two fields are freely filled up by the area responsible for creating and forwarding the tickets; given they are written in the users' language and also contain

copied error messages, their text features a mixture of Portuguese and English terms. Before the text transformation and other procedures were executed, that text was treated through the following procedures:

- Removal of numbers;
- Removal of server names;
- Removal of punctuation;
- Conversion of all terms to lower case;
- Removal of stop words from the English and Portuguese languages.

It is worth noting that not all incident tickets are relevant for the training of all classifiers. For example, a ticket assigned to the Application team (which is a part of Level 2) holds no learning value for the L0 and L1 classifiers; similarly, a ticket handled by the Cloud team (which is a part of Level 0) is not to be used in the learning process of the L1 vs. L2 classifier. As such, before being fed to each classifier, the training dataset was filtered so that only relevant tickets be used in the learning. Due to that, the only classifier trained with the whole dataset was L0 vs. Non L0; as the classifier that lies on the root of the classification tree of Figure 4, it is the entry point for all tickets that are being forwarded and must therefore learn patterns for all of them in order to differentiate those of Level 0 from those that are Non Level 0 (that is, Level 1 and Level 2).

5.2 Evaluation Means

5.2.1 F1 Score

The F1 score is a commonly used metric for model selection (HAN *et al.*, 2011); in other words, the choosing of the best classifier for a given problem. The metric is a combination of precision and recall via the harmonic mean, as shown in Formula 20.

$$F1 = 2\left(\frac{precision\ .\ recall}{precision + recall}\right)$$

Formula 20 – F1 Score

The precision measures the exactness of the classifier. Its value corresponds to the percentage of items that have been labeled as belonging to a class that are actually part of that class. A value of precision that equals 1 (its maximum), then, means that every item the classifier identified as belonging to a class was indeed a part of it. For example, if a classifier working to distinguish between classes Application and Database labels 100 records as Database, but only 50 of those are right, its precision for that class will be 50%.

The recall, meanwhile, measures the completeness of the classifier. Its value corresponds to the percentage of items that belong to a class and that were correctly labeled as so; it shows how many of the items of each class were recovered. A value of recall that equals 1 (its maximum), then, means that the classifier correctly labeled all items of all classes. For example, if a classifier, once more, is working to distinguish between classes Application and Database and out of the 100 database records it only labels 30 as so, its recall for that class will be 30%.

5.2.2 Confusion Matrix

A confusion matrix (HAN *et al.*, 2011) allows the detailed visualization of a classifier's behavior. It shows, through totalized quantities, how items from each of the evaluated classes are being classified. Table 5 shows an example of what a confusion matrix looks like for a classifier working to separate tickets between classes Application, Database, and Network.

Table 5 – Confusion Matrix Example

	Application	**Database**	**Network**
Application	**25**	2	3
Database	5	**12**	4
Network	7	6	**50**

Each row represents the items of a class; each column shows how those items have been classified. In the first row, for example, it is possible to see 25 Application tickets have been correctly labeled; while 2 of them were classified as being of the Database category and 3 were classified as being Network tickets. The main diagonal of the confusion matrix (which is highlighted in bold), therefore, shows the number of correctly classified records for each class.

5.2.3 Forwarding Cost

With the goal of assessing the workload difference between the current scenario shown in Figure 3 (in which tickets are manually forwarded to one of the 13 specialized areas), and the implemented solution (in which the routing of the tickets to the areas is done through the tree of classifiers seen in Figure 4), the forwarding cost metric (MENEZES, 2009) is used.

The forwarding cost is meant to represent the labor dispensed in the process of analyzing the ticket (which may entail learning details about it and contacting various areas in order to determine if they are capable of handling it) and sending it to the respective group.

Within the manual setting of Figure 3, and assuming no errors occur in the process (in other words, no tickets are forwarded to the wrong areas and thereby sent back to the team responsible for assigning them to the specific groups), the forwarding cost of each ticket is equivalent to 1. The assumption of no errors occurring in the process stems from how there are, currently, no metrics available regarding the incorrect forwarding of tickets in the company that serves as this work's case study. Therefore, in order to measure the improvements Incident Routing will bring, the comparison that will be made will be between the automatic proposed setting and a best-case manual scenario where no mistakes are made.

Within the proposed setup, on the other hand, tickets that are correctly routed by the classifiers will have a forwarding cost of 0 given no manual labor and analysis efforts are required, representing a workload gain in the service desk process; meanwhile, tickets that are incorrectly routed will have a forwarding cost of 2.

As shown in Figure 19, a misclassification will require that the team that receives the ticket send it back to the routing team (the first forwarding) so that it can, then, be manually assigned to those who are truly responsible for and capable of taking care of the issue (the second forwarding).

With these considerations, it will be possible to identify the gain obtained with the deployment of the ticket routing system.

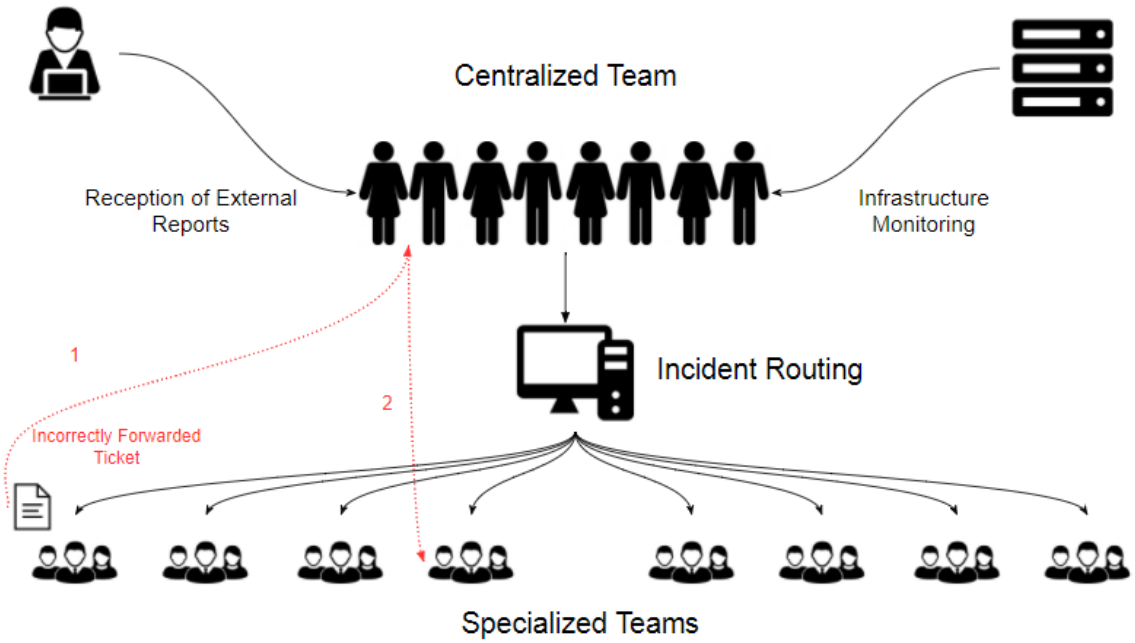

Figure 19 – Automatic Ticket Forwarding Process

5.3 Classification Refinement

5.3.1 Methodology

For the classification refinement experiments, the obtained dataset, consisting of 38,933 tickets collected between 04/24/2016 and 04/24/2017, was divided into two pieces: 80% of its records (31,146 tickets) were used for training; the other 20% of the records (7,787 tickets) were used for testing, which is presented in Experiment V.

Consequently, during the building of the models and the selection of the best classification configurations, the testing tickets were not used. Through the first four experiments, they remained set aside so that the performance of the constructed classifiers could be evaluated on tickets they had never worked with; consequently replicating a scenario in which new tickets arrive and the classifiers need to identify their patterns and send them to the correct specialized teams.

During the first four experiments, when only the training tickets were used, the method of evaluation employed was the k-fold cross-validation (HAN *et al.*, 2011), where *k* was 5.

The k-fold cross-validation works by breaking up the training dataset into *k* distinct pieces, as shown in Figure 20. The training (that is, the building of the predictive models) is done by using *k-1* of the pieces; the slice of the dataset that is not used for training is employed as a validation dataset. As so, with models constructed, the items of that slice are classified, and the results (in terms of the metrics employed, which in the case of the experiment was the F1 score) is calculated. The k-fold cross-validation repeats that procedure *k* times, until all pieces have been used for validation. The returned final result is the average of all *k* iterations of the procedure.

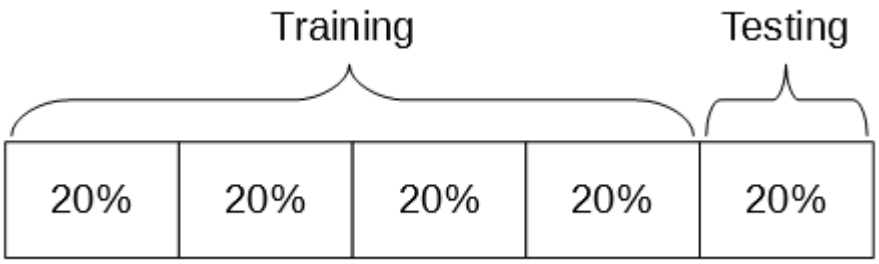

Figure 20 – 5-Fold Cross-Validation (First Iteration)

The k-fold cross-validation is a good tool for model selection (ABU-MOSTAFA *et al.*, 2012) for it brings a reliable estimate of how the learning model being evaluated will perform when it is moved to a scenario outside of the training sample (either when it is employed on the test dataset or when it is placed in the real world to face real new data).

5.3.2 Experiment I – Text Transformation and Classification

Experiment I describes the selection of the best text transformation and classification approaches for each of the five classifiers of the classification tree of Figure 4. As so, for classifiers L0 vs. Non L0; L1 vs. L2; L0; L1; and L2, these approaches will be evaluated via the 5-fold cross validation and the pair that performs best according to the reported F1 score will be chosen.

The text transformation techniques that were tested were: Term Frequency (TF); Term Frequency – Inverse Term Frequency (TF-IDF); Logarithmic Term Frequency (TF-LOG); Fractional Term Frequency (TF-FRAC); and Word2Vec (W2V).

Meanwhile, the machine learning algorithms that were tasked with text classification were: Decision Tree (DT), Naïve Bayes (NB), Nearest Neighbors (KNN),

Neural Network (NN), Random Forest (RF), Stochastic Gradient Descent (SGD), and Support Vector Machine (SVM).

Table 6 brings a summary of the configuration details of the text transformation approaches that were used. In particular, it is worth highlighting that Term Frequency and its variations were calculated with the use of unigrams (as seen in Table 2); in other words, terms were considered on their own, and not in combinations of two (bigrams), three (trigrams), or more.

Table 6 – Configuration of Text Transformation Approaches

	Configuration
TF	**N-grams Considered:** Unigrams
TF-IDF	**N-grams Considered:** Unigrams
TF-LOG	**N-grams Considered:** Unigrams
TF-FRAC	**K:** 1 **N-grams Considered:** Unigrams
W2V	**Aggregation Method for Word Vectors:** Average **Dimensions of Word Vectors:** 300 **Model**: Trained on the Corpus Itself

Table 7, on the other hand, carries a summary of the configuration details of the machine learning algorithms used in the experiment.

Table 7 – Configuration of Text Classification Approaches

	Configuration
DT	**Measure of Split Quality:** Gini Index
KNN	**Distance Measure:** Euclidean Distance **Number of Neighbors Considered:** 5
NB	**-**
NN	**Activation Function:** Rectifier **Number of Hidden Layers:** 1 **Number of Neurons in Hidden Layer:** 100
RF	**Measure of Split Quality:** Gini Index **Number of Trees:** 10
SGD	**Stopping Criterion:** 5 Iterations Through Dataset
SVM	**Kernel:** Linear **Multiclass Approach:** One-Versus-All

Classifier L0 vs. Non L0

Table 8 shows, for the L0 vs. Non L0 classifier, the overall results for the thirty-five combinations of text transformation and text classification techniques. Results reported are the F1 scores weighted according to the number of records of each class obtained during the 5-fold cross-validation. The best overall result is highlighted in bold.

Table 8 – L0 vs. Non L0 (Experiment I – Overall Results)

	TF	TF-IDF	TF-LOG	TF-FRAC	W2V
DT	0.9593	0.9568	0.9578	0.9571	0.9464
KNN	0.9600	0.9384	0.9588	0.9600	0.9591
NB	0.8840	0.9540	0.9562	0.9221	0.9313
NN	0.9615	0.9600	0.9602	0.9603	0.9593
RF	**0.9656**	0.9648	0.9637	0.9639	0.9626
SGD	0.9540	0.9616	0.9615	0.9571	0.9383
SVM	0.9565	0.9641	0.9645	0.9625	0.9572

All combinations of transformation techniques and classifiers were able to separate between tickets of the service desk's Level 0 and those of Level 1 and Level 2 (Non Level 0) with a good degree of success. With the only negative highlight being the Naïve Bayes classifier with Term Frequency, which yielded an F1 Score of 0.8840; a number that albeit not bad is much worse than the results reported by the other configurations. The best result overall was achieved by the combination TF and RF. The results of the pair are detailed in Table 9, where it is possible to see the F1 score, precision, and recall for both classes involved in the classification as well as the total number of records of each class.

Table 9 – L0 vs. Non L0 (Experiment I – Best Results – TF / RF)

	Precision	Recall	F1 Score	Number of Records
L0	0.9061	0.8704	0.8879	4,832
Non L0	0.9764	0.9834	0.9799	26,314
Average / Total	0.9655	0.9659	0.9656	31,146

As evidenced by Table 9, the classifier obtained better results in all metrics for the class that is more represented in the dataset. A question that will be addressed in the third and fourth experiments.

Classifier L1 vs. L2

Table 8 Table 10 shows, for the L1 vs. L2 classifier, the overall results for the thirty-five combinations of text transformation and text classification techniques. Results reported are the F1 scores weighted according to the number of records of each class obtained during the 5-fold cross-validation. The best overall result is highlighted in bold.

Table 10 – L1 vs. L2 (Experiment I – Overall Results)

	TF	TF-IDF	TF-LOG	TF-FRAC	W2V
DT	0.9330	0.9307	0.9333	0.9319	0.9152
KNN	0.9353	0.9216	0.9380	0.9366	0.9379
NB	0.8722	0.9273	0.9243	0.8621	0.8344
NN	0.9405	0.9402	0.9398	0.9377	0.9382
RF	**0.9415**	0.9403	0.9404	0.9405	0.9397
SGD	0.9213	0.9345	0.9340	0.9231	0.8980
SVM	0.9298	0.9380	0.9397	0.9337	0.9261

As it had happened with the L0 vs. Non L0 classifier, all learning models achieved good results when separating tickets from Level 1 and Level 2 in terms of the weighted F1 score. The only instances in which the score fell below 0.9 were with the combinations of the NB classifier and TF, TF-FRAC, and W2V; and the combination of SGD and W2V. Once more the best result appeared for TF and RF, and it is detailed in Table 11.

Table 11 – L1 vs. L2 (Experiment I – Best Results – TF / RF)

	Precision	Recall	F1 Score	Number of Records
L1	0.8058	0.6639	0.7280	2,975
L2	0.9581	0.9796	0.9687	23,339
Average / Total	0.9409	0.9439	0.9415	26,314

Table 11 shows a considerable imbalance between classes L1 and L2; furthermore, it reveals a disparity in the quality of the classification done for both classes. While the classifier is easily able to successfully label L2 tickets, it struggles with items belonging to L1 (the least-represented class). In fact, all classification algorithms had similar issues with that class. Table 12 shows the average F1 score for all algorithms with all text transformation techniques in relation to L1, and it is possible to see RFs and NNs stand out from the crowd, but that even their F1 score is lower that obtained for the L2 class.

Table 12 – L1 vs. L2 (Average F1 Score For All Classifiers – Class L1)

	DT	KNN	NB	NN	RF	SGD	SVM
L1	0.6868	0.6857	0.4826	0.7204	0.7207	0.6219	0.6787

Classifier L0

Table 13 shows, for the L0 classifier, the overall results for the thirty-five combinations of text transformation and text classification techniques. Results reported are the F1 scores weighted according to the number of records of each class obtained during the 5-fold cross-validation. The best overall result is highlighted in bold.

Table 13 – L0 (Experiment I – Overall Results)

	TF	TF-IDF	TF-LOG	TF-FRAC	W2V
DT	0.9788	0.9805	0.9801	0.9773	0.9498
KNN	0.9092	0.9740	0.9758	0.9574	0.9486
NB	0.9797	0.9595	0.9581	0.8199	0.9122
NN	0.9870	0.9842	0.9875	0.9863	0.9569
RF	0.9696	0.9747	0.9727	0.9737	0.9596
SGD	0.9865	0.9894	0.9880	0.9847	0.9177
SVM	0.9863	**0.9898**	0.9876	0.9827	0.9332

Differently from what had happened in the binary classification problems of L0 vs. Non L0 and L1 vs. L2, where the combination of TF and RF produced the best results, for classifier L0, TF-IDF and SVM achieved the best weighted F1 score. It is also possible to see how the W2V approach, for all classifiers (with the exception of NB) yielded the worst numbers. Table 14 shows the detailed results of the best combination.

Table 14 – L0 (Experiment I – Best Results – TF-IDF / SVM)

	Precision	Recall	F1 Score	Number of Records
Cloud	1.0	0.8421	0.9143	114
External Network	0.9981	0.9993	0.9987	4,239
Incident Management	0.9106	0.9645	0.9368	338
Production Support	0.9328	0.8865	0.9091	141
Average / Total	0.9901	0.9899	0.9898	4,832

Again, there is a level of imbalance between the classes in the dataset, and the ticket categories that are not well represented have the lowest F1 scores. Contrarily to what had occurred for the L1 vs. L2 classifier (Table 11), though, the F1 scores for all classes are greater than 0.9. Such solid results, however, were not common to all combinations of text transformation techniques and classifiers.

Table 15 presents the average F1 score with all text transformation techniques for all algorithms in relation to the classes of L0 and it is possible to see how classifiers tend to struggle with the least represented classes, for while they achieve good results with the External Network class, the same does not hold for others.

Table 15 – L0 (Average F1 Score For All Classifiers – All Classes)

	DT	**KNN**	**NB**	**NN**	**RF**	**SGD**	**SVM**
Cloud	0.7818	0.6987	0.4360	0.8345	0.7357	0.7445	0.7301
External Network	0.9968	0.9647	0.9787	0.9976	0.9951	0.9955	0.9976
Incident Management	0.8353	0.7262	0.6079	0.8864	0.8321	0.8614	0.8723
Production Support	0.7505	0.6600	0.4940	0.8055	0.7360	0.7577	0.7684

Classifier L1

Table 16 shows, for the L1 classifier, the overall results for the thirty-five combinations of text transformation and text classification techniques. Results reported are the F1 scores weighted according to the number of records of each class obtained during the 5-fold cross-validation. The best overall result is highlighted in bold.

Table 16 – L1 (Experiment I – Overall Results)

	TF	**TF-IDF**	**TF-LOG**	**TF-FRAC**	**W2V**
DT	0.8818	0.8845	0.8798	0.8788	0.8552
KNN	0.8642	0.8573	0.8722	0.8668	0.8610
NB	0.8348	0.8307	0.7639	0.7071	0.7570
NN	0.8954	0.8909	0.8934	0.8919	0.8804
RF	0.8891	0.8763	0.8790	0.8797	0.8817
SGD	0.8649	0.9009	0.9041	0.8805	0.7814
SVM	0.8883	0.9036	**0.9065**	0.8824	0.8330

Again, SVM is the best classifier. This time, though, its best results occur when the text that is fed into it has been transformed into vectors by TF-LOG, not TF-IDF. Its

F1score is good for all text transformation techniques, with the exception of W2V, whose score of 0.8330 stands out negatively when compared to the other SVM scores. Aside from NB and the combination of SGD and W2V, the overall numbers for all combinations are on a similar level, always standing between 0.86 and 0.91.

Table 17 shows the detailed results of the best combination. As it has been the case for all classifiers, an imbalance exists between the classes, and the one that is most represented in the dataset is the class for which the best F1 score appears. A negative highlight of the L1 classes is found in the numbers for the Job and Operation classes, whose F1 scores are particularly low when compared to those of others.

Table 17 – L1 (Experiment I – Best Results – TF-LOG / SVM)

	Precision	**Recall**	**F1 Score**	**Number of Records**
Backup	0.9305	0.9564	0.9433	826
Control	0.8103	0.8650	0.8368	163
Job	0.7784	0.6681	0.7195	238
Monitoring	0.9358	0.9557	0.9456	1,602
Operation	0.7500	0.5753	0.6512	146
Average / Total	0.9058	0.9092	0.9065	2,975

For those two classes, such a phenomenon is not exclusive to the TF-LOG and SVM combination: it happens for all text transformation techniques and classifiers. Table 18 shows the average F1 score, considering all text transformation approaches, that the employed classifiers produced for the classes Job and Operation; scores are consistently low.

Table 18 – L1 (Average F1 Score For All Classifiers – Job and Operation Classes)

	DT	**KNN**	**NB**	**NN**	**RF**	**SGD**	**SVM**
Job	0.6220	0.6199	0.3192	0.6712	0.6210	0.5886	0.6470
Operation	0.5662	0.5048	0.1945	0.6165	0.6158	0.5208	0.5612

Classifier L2

Table 20 shows, for the L2 classifier, the overall results for the thirty-five combinations of text transformation and text classification techniques. Results reported are the F1 scores weighted according to the number of records of each class obtained during the 5-fold cross-validation. The best overall result is highlighted in bold.

Table 19 – L2 (Experiment I – Overall Results)

	TF	TF-IDF	TF-LOG	TF-FRAC	W2V
DT	0.8451	0.8399	0.8358	0.8338	0.8156
KNN	0.8161	0.7911	0.7905	0.7965	0.8502
NB	0.7757	0.7821	0.7567	0.7319	0.6248
NN	**0.8713**	0.8643	0.8651	0.8551	0.8649
RF	0.8568	0.8528	0.8559	0.8532	0.8636
SGD	0.8269	0.8521	0.8441	0.8012	0.7654
SVM	0.8535	0.8665	0.8636	0.8449	0.8326

The best combination was, therefore, TF and NN. The detailed view of the classification done by that combination, which is shown in Table 20, reveals that although this piece of the dataset has a certain degree of imbalance, as it has been the case for all of the subsets used for the training of each classifier, the results are more balanced.

Table 20 – L2 (Experiment I – Best Results – TF / NN)

	Precision	Recall	F1 Score	Number of Records
Application	0.8754	0.8941	0.8847	8,952
Database	0.8601	0.8396	0.8497	4,077
Internal Network	0.9116	0.8955	0.9035	1,761
Platform	0.8641	0.8578	0.8609	8,549
Average / Total	0.8713	0.8714	0.8713	23,339

There is no class that appears as a negative highlight, as F1 scores for all of them fall between 0.84 and 0.91. That more balanced behavior may be attributed to how, despite the imbalance, all classes are solidly represented: there are over 1,000 tickets for each one of them. That characteristic is a considerable contrast to the subsets of L0 and L1, in which a few classes had over 1,000 representatives while others featured ticket numbers in the low hundreds. Another noticeable distinction of the L2 classifier lies in how the best F1 score was not reached by the most represented class (Platform), but by Internal Network, which is the least represented ticket category of the subset.

Overview

With the end of Experiment I, for each of the five classifiers that compose the classification tree of Incident Routing, the best combination of text transformation and classification approaches were chosen. Those combinations were:

- **L0 vs. Non L0:** TF and RF;
- **L1 vs. L2:** TF and RF;
- **L0:** TF-IDF and SVM;
- **L1:** TF-LOG and SVM;
- **L2:** TF and NN.

Therefore, classifiers DT, KNN, NB, and SGD; as well as the text transformation approaches TF-FRAC and W2V were not a part of any winning combination. Table 21 and Table 22 present the average of the weighted F1 score achieved by the text transformation techniques and the classification algorithms throughout the experiment (that is, for all five classifiers of the Incident Routing tree). TF-FRAC and, especially, W2V are negative highlights – albeit by a small margin – among the text transformation approaches while DT, KNN, NB, and SGD fall slightly below NN, RF, and SVM among classification algorithms.

Table 21 – Average Of Weighted F1 Score (Text Transformation Techniques)

TF	TF-IDF	TF-LOG	TF-FRAC	W2V
0.9107	0.9164	0.9152	0.9009	0.8882

Table 22 – Average Of Weighted F1 Score (Classification Techniques)

DT	KNN	NB	NN	RF	SGD	SVM
0.9135	0.9030	0.8504	0.9268	0.9224	0.9068	0.9210

W2V, in particular, stood out negatively for classifiers L0 and L1. For instance, as Table 23 reveals, for L0, the SVM classifier (which was the winning algorithm when paired up with TF-IDF, as shown in Table 14) went through a considerable drop in performance when the W2V approach was plugged into it in place of TF-IDF.

Table 23 – L0 (Experiment I – W2V / SVM)

	Precision	Recall	F1 Score	Number of Records
Cloud	0.2632	0.0439	0.0752	114
External Network	0.9920	0.9941	0.9930	4,239
Incident Management	0.5820	0.8521	0.6965	338
Production Support	0.5658	0.3050	0.3963	141
Average / Total	0.9342	0.9416	0.9332	4,832

As it can be seen, the drop in performance is very prominent for the poorly represented classes. Cloud goes from 0.9143 when TF-IDF and SVM are joined to 0.0752 when W2V is used instead; Incident Management goes from 0.9368 to 0.6965; and Production Support drops all the way from 0.8669 to 0.3963.

That pattern reappears for L1, as shown in Table 24. In that case, the best combination was TF-LOG and SVM (which is shown in Table 17). When TF-LOG is replaced by W2V, the F1 score goes from 0.8368 to 0.7486 for Control; from 0.7195 to 0.4628 for Job; and from 0.6512 to 0.3846 for Operation.

Table 24 – L1 (Experiment I – W2V / SVM)

	Precision	Recall	F1 Score	Number of Records
Backup	0.9178	0.9189	0.9183	826
Control	0.7005	0.8037	0.7486	163
Job	0.6720	0.3529	0.4628	238
Monitoring	0.8501	0.9413	0.8934	1,602
Operation	0.6452	0.2740	0.3846	146
Average / Total	0.8364	0.8477	0.8330	2,975

Figure 21 shows a visual summary of the results achieved in Experiment I. The setup of the classification tree, its classifiers, the chosen text transformation and classification techniques, and the F1 score for each of its classes are presented.

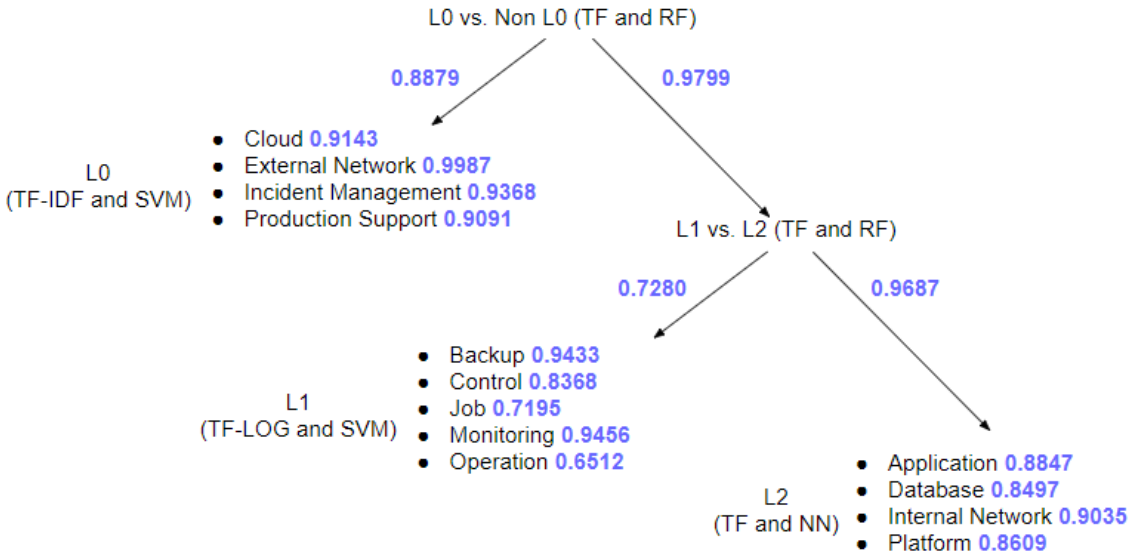

Figure 21 – Visual Summary (Results Of Experiment I)

Upon the closing of Experiment I, two possible ways through which classification can be improved present themselves. Firstly, as discussed, there is how the imbalanced subsets used for the training of each classifier yielded results that are biased towards the majority classes, as the F1 score for the least represented categories tended to be inferior in all cases, an effect that was quite clear in all classifiers with the exception of L2.

Secondly, there is how the employed subsets have an elevated number of attributes. Given each term in the text of the tickets corresponds to one feature (or dimension) all chosen algorithms are operating in a high number of dimensions as they seek for the best hypothesis that fits the training dataset. The number of terms (features) for the subset of each classifier is:

- **L0 vs. Non L0:** 20,895.
- **L1 vs. L2:** 19,205.
- **L0:** 6,790.
- **L1:** 5,995.
- **L2:** 17,597.

With those questions in mind, subsequent experiments will aim to address the high dimensionality and imbalanced datasets issues. Given high dimensionality has been reported to increase the bias of classifiers towards the majority classes (CHAWLA *et al.*,

2002), feature selection will be treated first, as it may potentialize the effects of data balancing.

5.3.3 Experiment II – Feature Selection

Experiment II involves applying Improved Fisher's Discriminant Ratio (IFDR) to all subsets used for the training of the classifiers. That means that for classifier L0 vs. Non L0, IFDR will be calculated for all terms of the tickets belonging to both classes; the goal will be to determine how important the term is for the distinction between L0 tickets and Non L0 tickets. Likewise, for example, when it comes to classifier L2, IFDR will be calculated for each term for all L2 classes; that is, Application, Database, External Network and Platform.

As exemplified in Table 3, for any given term, the maximum IFDR score will be selected. All terms present in the tickets will be ordered according to that maximum score; with that, it will be possible to select only the terms that are most important for the differentiation between classes. Given IFDR exists in two variations, binary and frequency, both were evaluated.

The tests done in Experiment II consist of the following procedure: for each classifier (L0 vs. Non L0, L1 vs. L2, L0, L1, and L2), the combination of text transformation and classification approaches selected during Experiment I is used; the top *N* terms (according to one IFDR variation) in each of the five subsets is selected; the 5-fold cross-validation is performed by considering only the selected terms in the classification; the results, in terms of the weighted F1 score, are reported.

In the experiment, the number of selected attributes was varied between the top 1% and the whole set of terms (100%) in steps of 1%. In other words, for each classifier and IFDR variation, one hundred results are reported. The goal is to select a percentage of attributes with which the five classifiers perform well in terms of the weighted F1 score so that they can eventually be applied on the test dataset that has been reserved.

The red dots on the charts represent the point on which the highest F1 Score was achieved. Although some of the peaks appear towards relatively high percentages, all of the classifiers (for the two IFDR variations) reach performance levels that are close to the maximum when smaller percentages of attributes are selected. Moreover, stability in

performance (in other words, F1 Score levels that are consistently good) are also achieved early.

It is worth pointing out that since both tested feature selection variations are very similar, the difference in their performance is minimal.

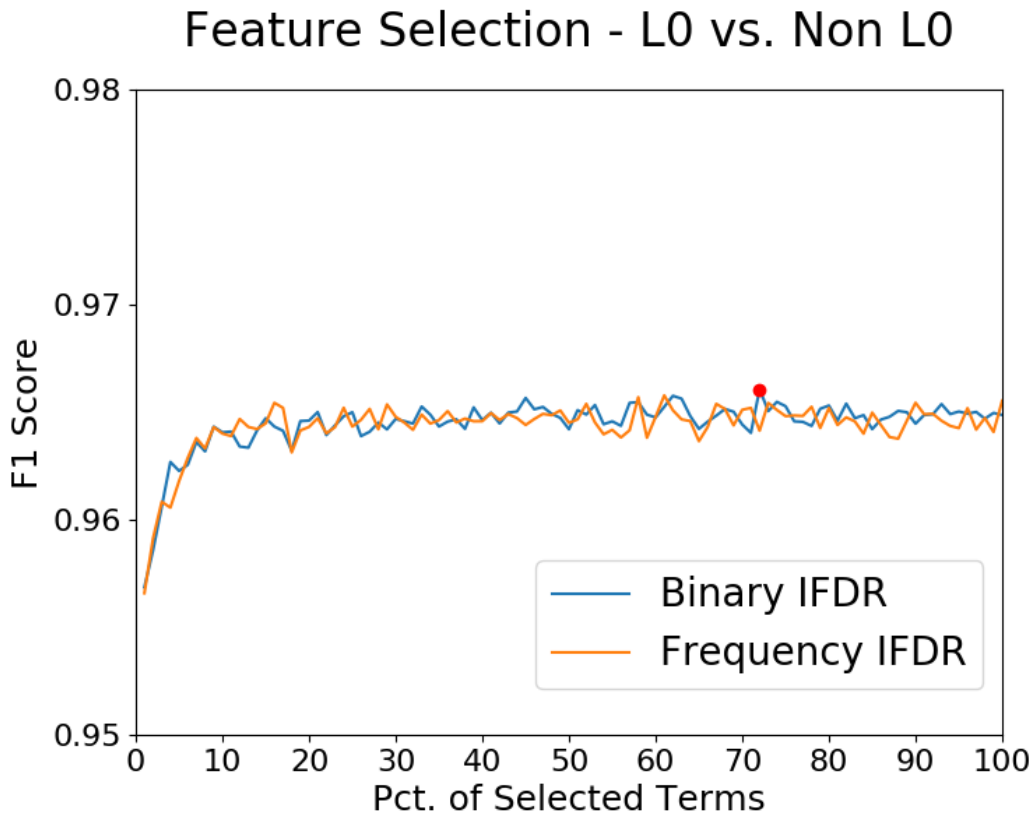

Figure 22 – L0 vs. Non L0 (Experiment II – Feature Selection)

Figure 22 shows the results obtained for classifier L0 vs. Non L0 (using the combination TF and RF). It is possible to notice that a very good F1 score is already achieved when a meager 1% of the terms (2,089 out of 20,895) is used. Still, between the 1% and 15% marks there is a small but steady increase in performance, which reaches a certain level of stability after that point.

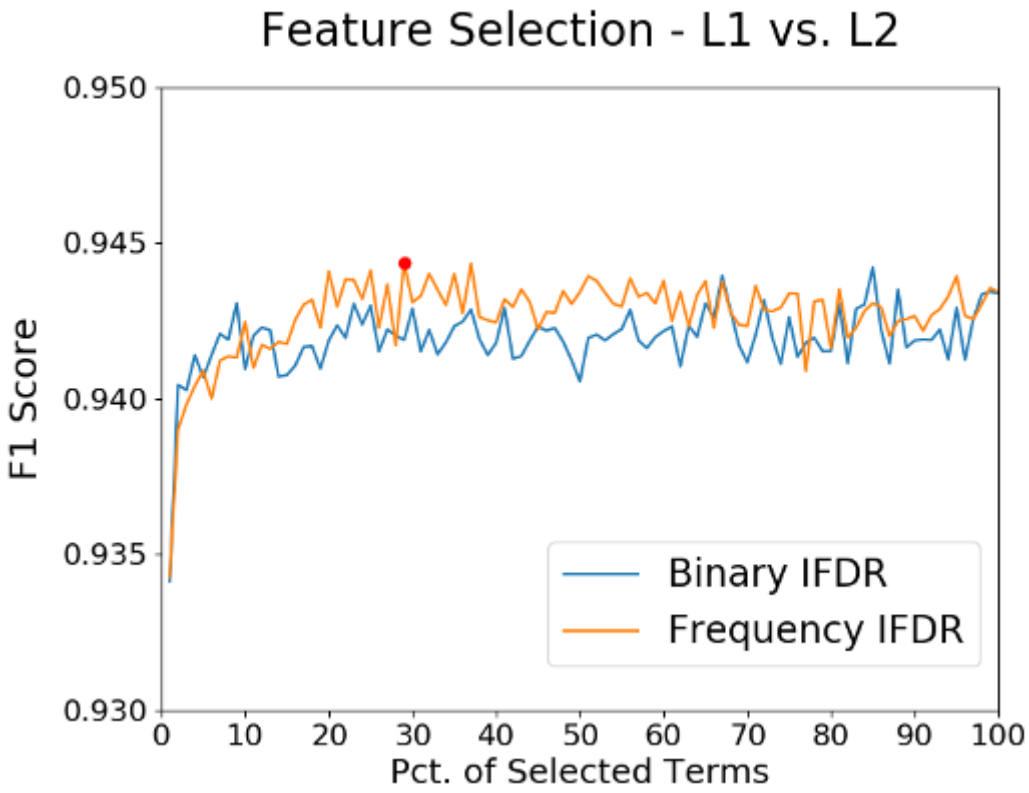

Figure 23 – L1 vs. L2 (Experiment II – Feature Selection)

Figure 23 reveals a similar pattern for classifier L1 vs. L2 (using the combination TF and RF). In this case, the peak (reached by the frequency variation of IFDR) comes sooner, at the 29% mark; but the overall behavior is the same. Through the early percentages, a steady but slow rise can be seen, which is sustained until the range between 20% and 25%; after that point, via peaks and valleys, the weighted F1 score stays within a certain good interval that is quite close to the peak, showing – once more – that with a much lower quantity of attributes it is possible to achieve results that are equal or slightly better than the ones reached with the full set of terms.

Figure 24 presents the results for classifier L0 (TF-IDF and SVM). As it can be seen, the peak of performance is reached by the binary variation of IFDR at two points: 68% and 69%. Again, the performance improves through the early percentages and reaches stability after a point; in this case, the 10% mark.

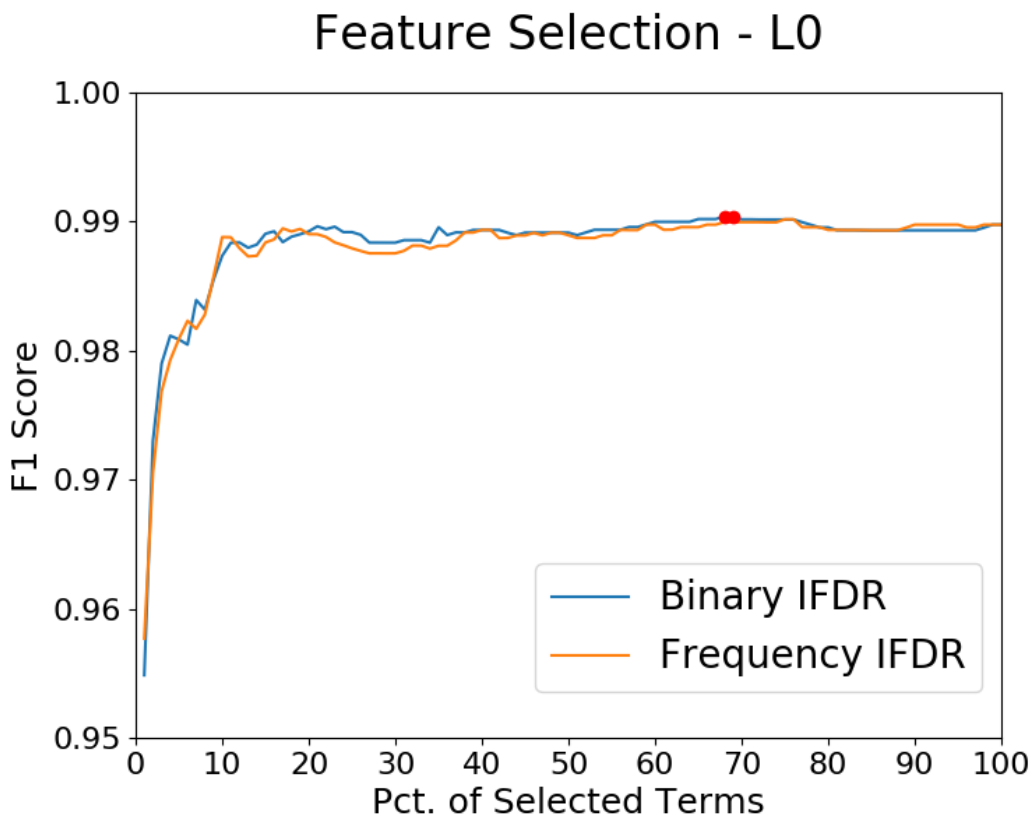

Figure 24 – L0 (Experiment II – Feature Selection)

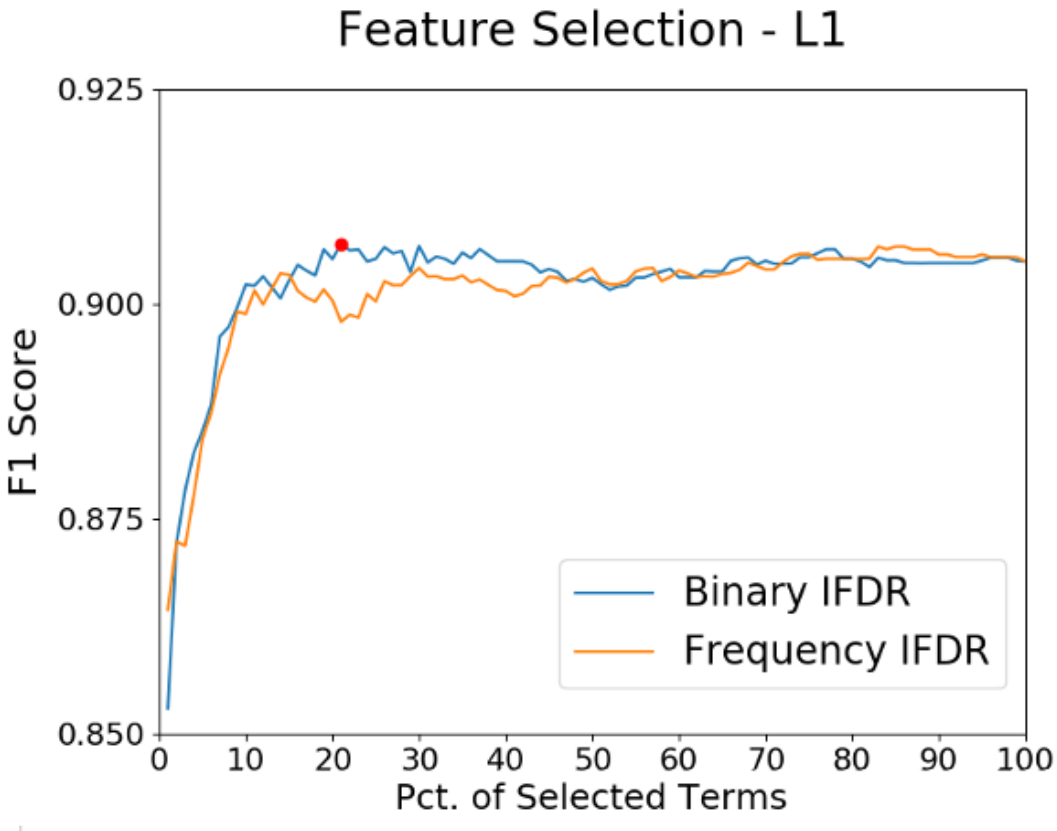

Figure 25 – L1 (Experiment II – Feature Selection)

Figure 25 presents the results for classifier L1 (TF-LOG and SVM). The performance peak is reached at 20% by the frequency variation of IFDR. Figure 26 presents the results for classifier L2 (TF and NN). The performance peak is reached at 39% by the binary variation of IFDR.

One noteworthy particularity of the behavior of classifier L2 is how the rising period of its performance is more prominent. At 1% of attributes, its performance for both IFDR variations is at around 0.75; when its stability comes, at the 25% mark, its performance has risen to almost 0.875: an increase of almost 0.125. For classifier L0 vs. Non L0, for example, at 1% its F1 score is already between 0.95 and 0.96, and its peak comes at below 0.97. For classifier L1 vs. L2, at 1% its performance is at 0.935 and its peak has an F1 score that is close to 0.945. Classifiers L1 and L2, meanwhile, present increases of about 0.3 and 0.5, respectively, between their performance at 1% and their respective peaks.

Therefore, classifier L2 needs a bigger percentage of attributes to achieve peak performance when compared to its four peers.

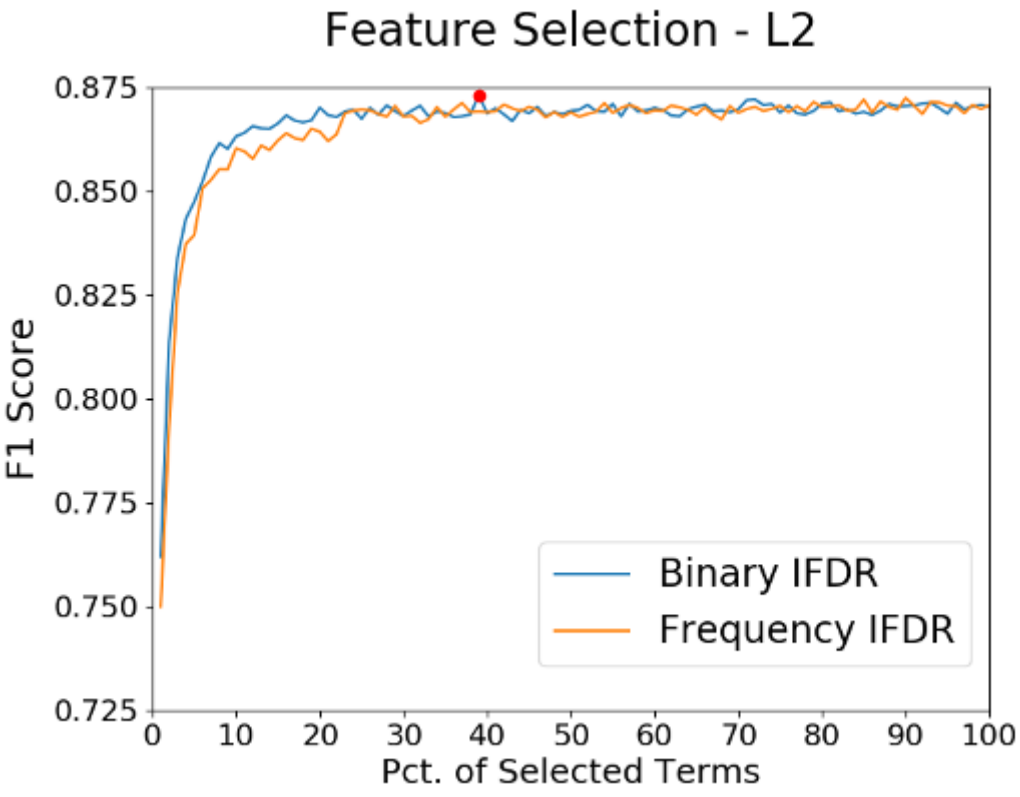

Figure 26 – L2 (Experiment II – Feature Selection)

That considerable increase can be attributed to the behavior of classes Application, Database, and Platform. Figure 27 shows, for classifier L2, the detailed results for the feature selection done with the binary variation of IFDR. It can be seen that while Internal Network tickets are, at 1%, classified at a rate that is close to the peak performance of the class, the same does not apply to the other kinds of tickets of the level. In order to differ between them and identify them successfully, more attributes are necessary, so the F1 scores for those classes take a while longer to stabilize, which reflects on the aggregated F1 score shown in Figure 26.

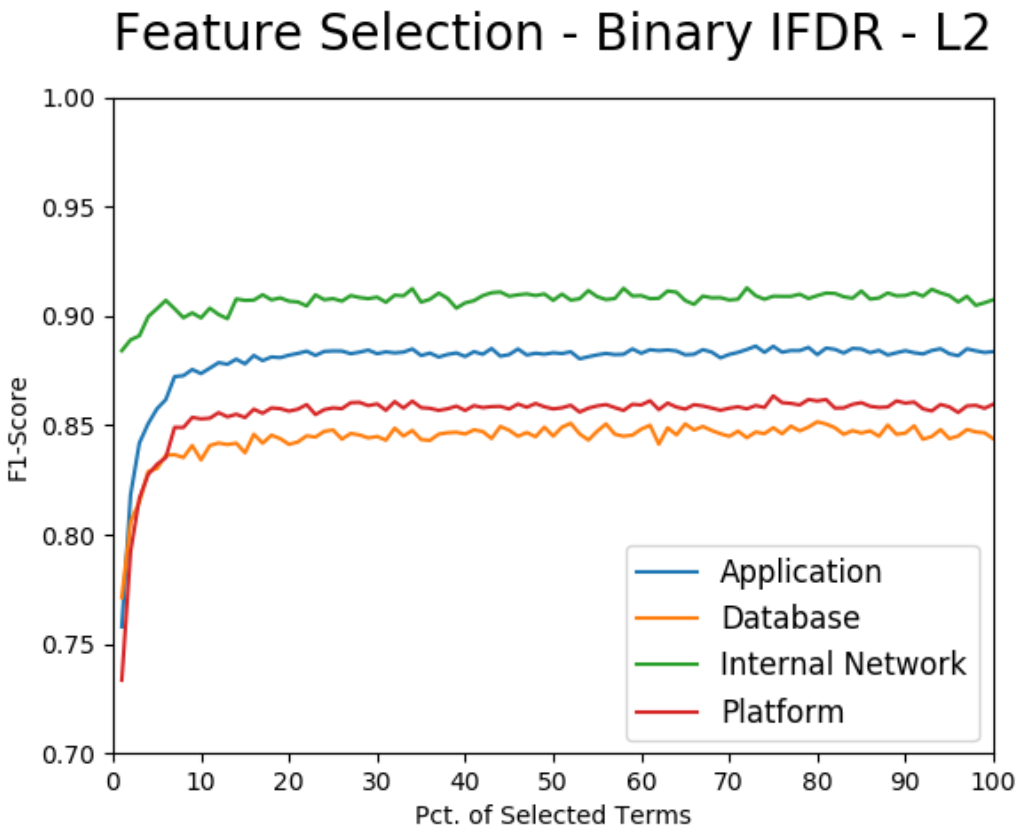

Figure 27 – L2 (Experiment II – Feature Selection – Detailed View)

Due to the positive performance feature selection had, and given the L2 classifier only stabilized its performance at around the 25% mark, a choice was made to, for each classifier, use the top 25% best-ranked terms in their respective subsets. The selection of the IFDR variation to use was made by, for each classifier, looking at the performance of both methods at that percentage. Table 7 shows the F1 Score at 25% for both IFDR versions for all classifiers, with the highest values highlighted in bold. Differences were minimal given the similarity between both approaches.

Table 25 – IFDR Scores At 25% For All Classifiers

Classifier	Binary IFDR	Frequency IFDR	Selected Method
L0 vs. Non L0	**0.9649**	0.9643	Binary IFDR
L1 vs. L2	0.9429	**0.9441**	Frequency IFDR
L0	**0.9891**	0.9879	Binary IFDR
L1	**0.9052**	0.9003	Binary IFDR
L2	0.8674	**0.8697**	Frequency IFDR

At the end of Experiment II, then, the selected configuration of each classifier of Incident Routing's tree of classifiers was:

- **L0 vs. Non L0:** TF and RF; Binary IFDR at 25%.
- **L1 vs. L2:** TF and RF; Frequency IFDR at 25%.
- **L0:** TF-IDF and SVM; Binary IFDR at 25%.
- **L1:** TF-LOG and SVM; Binary IFDR at 25%.
- **L2:** TF and NN; Frequency IFDR at 25%.

Moreover, the achieved reduction in the number of used attributes in the training of each classifier was:

- **L0 vs. Non L0:** From 20,895 to 5,223.
- **L1 vs. L2:** From 19,205 to 4,801.
- **L0:** From 6,790 to 1,967.
- **L1:** From 5,995 to 1,498.
- **L2:** From 17,597 to 4,399.

5.3.4 Experiment III – Imbalanced Data (Data Level Solution)

Experiment III involves the attempt to solve, through the use of data level solutions, the data imbalance problem that exists in the subsets used in the training of all classifiers. Even though classifier L2, differently from the others, did not display signs that it was being negatively affected by data imbalance – as all its classes are relatively well-represented and their respective F1 scores in Experiment I were even – data level

solutions were nevertheless applied on its dataset in an attempt to achieve performance gains.

For all classifiers, the configuration that was used was the one obtained at the end of Experiment II; that is, the best text transformation technique and classification algorithm of Experiment I plus the feature selection parameters chosen during Experiment II. Four data level solutions for class imbalance were tested:

- SMOTE;
- Random Oversampling (ROS);
- SMOTE-Edited Nearest Neighbors (SMOTE-ENN);
- SMOTE-Tomek Links (SMOTE-TL).

In all four cases, the procedure that was followed was the same. The 5-fold cross-validation was executed, and for each fold – that is, in every one of its five iterations – the data-balancing approaches were applied on the four pieces used for training (as shown in Figure 20); the data contained in the one piece of each iteration that is used for testing (or validation) was, then, ignored during the data-balancing procedure. After all, given its data points are meant to simulate tickets the classifier has never seen in order to accurately evaluate its performance, the data they contain cannot be taken into account by the balancing algorithms.

The results of the experiment are reported for each class of the five classifiers. The F1 scores obtained through all data level solutions are presented as well as their average as weighted according to the number of records of each class. For comparative purposes, the F1 scores obtained for all classes on Experiment I are also shown; in those cases, the scores are those of the best classifiers that were selected at the end of the experiment. Those scores are denoted by PC, which stands for pure classifiers; in other words, the classifiers without the feature selection of Experiment II and the data-balancing of this experiment.

Classifier L0 vs. Non L0

Table 26 shows the overall results obtained for classifier L0 vs. Non L0. The best result for each class is highlighted in bold. And, as far as the F1 score goes, absolutely no gains are visible. For both classes, the best performance is achieved by the pure classifier (PC) of Experiment I. At the same time, the use of data balancing techniques brings no considerable performance losses; ROS has the lowest overall score, but its outcome is just 0.0075 below that of PC.

Table 26 – L0 vs. Non L0 (Experiment III – Overall Results)

Class	PC	SMOTE	ROS	SMOTE-ENN	SMOTE-TL	Number of Records
L0	**0.8879**	0.8843	0.8797	0.8673	0.8853	4,832
Non L0	**0.9799**	0.9794	0.9781	0.9746	0.9796	26,314
Average / Total	**0.9655**	0.9647	0.9628	0.9580	0.9649	31,146

For classifier L0 vs. Non L0, the best performance among the data level solutions employed comes from the hybrid approach SMOTE-TL. By looking at its detailed scores in Table 27 and once more comparing them with those of the pure classifier (PC) of Experiment I, a couple of tiny improvements can be seen in the metrics that compose the F1 score: precision and recall. For the L0 class, there is a gain in precision; for the Non L0 class, there is a gain in recall. Those gains, however, are minimal, and the improvements that can be seen are nullified by declines in recall (in the case of the L0 class) and precision (in the case of the Non L0 class), which end up resulting in inferior F1 scores for both classes. In Table 27, the best results for each metric for each class are highlighted in bold.

Table 27 – L0 vs. Non L0 (Experiment III – Best Results – SMOTE-TL)

Class	Precision		Recall		F1		Number of Records
	PC	SMOTE-TL	PC	SMOTE-TL	PC	SMOTE-TL	
L0	0.9061	**0.9093**	**0.8704**	0.8628	**0.8879**	0.8853	4,832
Non L0	**0.9764**	0.9751	0.9834	**0.9842**	**0.9799**	0.9796	26,314
Average / Total	**0.9655**	0.9649	**0.9659**	0.9654	**0.9655**	0.9649	31,146

Classifier L1 vs. L2

Table 28 shows the overall results obtained for classifier L1 vs. L2. The best result for each class is highlighted in bold. Differently from what happened to classifier L0 vs. Non L0, there are noticeable (albeit small) gains in the application of data-balancing algorithms to the subset of classifier L1 vs. L2. Those gains, however, are not universal among the applied techniques. Both ROS and SMOTE-ENN make the classifier perform more poorly for both classes; SMOTE and SMOTE-TL, on the other hand, bring improvements in the F1 score for the least-represented class (L1) while reaching F1 scores for the most-represented class (L2) that are almost equal to the one that is obtained by the pure classifier. SMOTE, in particular, increases the F1 score of L1 by more than 0.1, which causes its overall F1 score be superior to the one of PC.

For SMOTE-ENN, the big drop in terms of F1 score is caused by a very poor result in the precision metric for the L1 class. While it is able to bring a great improvement in terms of recall for that class, which reaches 0.7818, it does so at the cost of the precision value, which falls down to 0.6310. As such, it is able to recover a much bigger portion of L1 tickets than its peers; but it achieves that number by labeling quite a few tickets that are not L1 as belonging to that class, which causes its precision to drop.

Table 28 – L1 vs. L2 (Experiment III – Overall Results)

Class	PC	SMOTE	ROS	SMOTE-ENN	SMOTE-TL	Number of Records
L1	0.7280	**0.7391**	0.7144	0.6980	0.7347	2,975
L2	**0.9687**	0.9683	0.9630	0.9561	0.9676	23,339
Average / Total	0.9415	**0.9424**	0.9349	0.9269	0.9412	26,314

For classifier L1 vs. L2, the best performance among the data level solutions employed comes from the oversampling approach SMOTE. By looking at its detailed scores in Table 29, it can be seen that it reaches superior performance when it comes to the least-represented class (L1) through an increase in the recall metric that exceeds 0.4; an improvement that does sacrifice the precision metric to a certain point, but not heavily enough to significantly harm the F1 score to a degree that makes it smaller than the one from the pure classifier. Moreover, SMOTE also brings a small gain to the precision metric of the most-represented class (L2). As so, in the case of classifier L1 vs. L2, the application of a data-level approach for treating the problem of imbalanced data yielded

positive results. In Table 29, the best results for each metric for each class are highlighted in bold.

Table 29 – L1 vs. L2 (Experiment III – Best Results – SMOTE)

Class	Precision		Recall		F1		Number of Records
	PC	SMOTE	PC	SMOTE	PC	SMOTE	
L1	**0.8058**	0.7748	0.6639	**0.7069**	0.7280	**0.7391**	2,975
L2	0.9581	**0.9630**	**0.9796**	0.9737	**0.9687**	0.9683	23,339
Average / Total	0.9409	**0.9418**	**0.9439**	0.9435	0.9415	**0.9424**	26,314

Classifier L0

Table 30 shows the overall results obtained for classifier L0. The best result for each class is highlighted in bold. In call cases, the weighted average of the F1 score is quite similar, with SMOTE obtaining the best overall result by a minimum margin (0.0001) over the pure classifier of Experiment I. Surprisingly, SMOTE reaches that result by improving not only the F1 score of the least-represented class (Cloud) by over 0.02, but by also having the best performance for the most-numerous class (External Network). In the latter case, though, the improvement is far smaller, of only 0.0005, and that same number is achieved by ROS. For the other less-numerous classes of the dataset (Incident Management and Production Support), however, neither SMOTE nor any of its peers is able to overcome the results of PC.

Table 30 – L0 (Experiment III – Overall Results)

Class	PC	SMOTE	ROS	SMOTE-ENN	SMOTE-TL	Number of Records
Cloud	0.9143	**0.9371**	0.9290	0.9101	0.9152	114
External Network	0.9987	**0.9992**	**0.9992**	0.9976	0.8832	4,239
Incident Management	**0.9368**	0.9321	0.9305	0.9181	0.9259	338
Production Support	**0.9091**	0.8920	0.8880	0.8924	0.8832	141
Average / Total	0.9898	**0.9899**	0.9895	0.9870	0.9883	4,832

For classifier L0, the best performance among the data level solutions employed comes from the oversampling approach SMOTE. By looking at its detailed scores in Table 31, the most considerable improvement the data-balancing brings appears in the recall metric of the Cloud class, which rises from 0.8421 in the case of PC to 0.9211 with

the use of SMOTE; an increase which has a positive reflection on the F1 score for that class even if the perfect precision metric achieved by PC is not matched by SMOTE. Everywhere else, PC and SMOTE alternate quite a bit in terms of which one reaches the best performance. For Production Support, SMOTE causes the recall metric to rise, but the precision drops, as the classifier recovers more tickets than it did without data-balancing, but achieves that result by labeling more tickets as Production Support even when they do not belong to that class. For External Network, SMOTE brings a small gain in precision while maintaining the exact same performance when it comes to recall. Finally, for Incident Management, the same happens: the precision increases from 0.9106 to 0.9329, but the recall tumbles from 0.9645 to 0.9320. In Table 31, the best results for each metric for each class are highlighted in bold.

Table 31 – L0 (Experiment III – Best Results – SMOTE)

Class	Precision		Recall		F1		Number of Records
	PC	SMOTE	PC	SMOTE	PC	SMOTE	
Cloud	**1.0**	0.9559	0.8421	**0.9211**	0.9143	**0.9371**	114
External Network	0.9981	**0.9991**	**0.9993**	**0.9993**	0.9987	**0.9992**	4,239
Incident Management	0.9106	**0.9329**	**0.9645**	0.9320	**0.9368**	0.9321	338
Production Support	**0.9328**	0.8857	0.8865	**0.9007**	**0.9091**	0.8920	141
Average / Total	**0.9901**	**0.9901**	**0.9899**	**0.9899**	0.9898	**0.9899**	4,832

Classifier L1

Table 32 shows the overall results obtained for classifier L1. The best result for each class is highlighted in bold. As it happened with classifier L0 vs. Non L0, apart from a small improvement in the F1 score of class Control when the dataset is processed by the oversampling approaches SMOTE and ROS, no gains were noticed. In fact, overall, there is a general drop in performance in relation to the results obtained in Experiment I. For the Backup class, the difference in F1 score between PC and the best scenario in which a data level solution for class imbalance is applied (SMOTE-TL) is greater than 0.015. For the Job class, the gap is also of the same scale; however, in that case, the best-performing technique is ROS. ROS also outperforms other data-balancing algorithms when it comes to the Monitoring class, but – once more – it is unable to beat PC, and the distance between both is of 0.0072. The negative highlight of the set is the Operation class; even though it

is the least-represented category of L1, it is the class whose classification performance suffers the largest fall, going from 0.6512 when it is classified by PC all the way to 0.6037 when SMOTE-TK, the best-performing algorithm for that class, is applied.

Despite having a very negative result for Operation, SMOTE still overcomes the other algorithms in relation to the weighted F1 score. Much of that can be attributed to how regular its performance is across the dataset: it is the best-performing option for Control tickets, and it reaches scores that are close to the best ones for Backup, Job, and Monitoring tickets.

Table 32 – L1 (Experiment III – Overall Results)

Class	PC	SMOTE	ROS	SMOTE-ENN	SMOTE-TL	Number of Records
Backup	**0.9433**	0.9244	0.9167	0.9257	0.9276	826
Control	0.8368	**0.8480**	0.8419	0.7844	0.8011	163
Job	**0.7195**	0.6943	0.6958	0.6458	0.6891	238
Monitoring	**0.9456**	0.9373	0.9384	0.9039	0.9310	1,602
Operation	**0.6512**	0.5892	0.5808	0.5571	0.6037	146
Average / Total	**0.9065**	0.8923	0.8901	0.8657	0.8875	2,975

Table 33 displays the detailed performance of SMOTE for the L1 classifier. With a more granular view, it can be seen how – as a whole – SMOTE improves the precision metric for most classes by a small margin. However, as precision rises slightly, the Backup, Job, and Monitoring classes suffer falls in terms of recall. For the Backup class, the number goes from 0.9564 to 0.9104; for the Job class, the number goes from 0.6661 to 0.6261; and for the Monitoring class, the number goes from 0.9557 to 0.9245.

As a consequence, for those classes, the F1 score achieved by SMOTE is smaller than that of PC. The only class in which improvements are seen in both precision and recall is Control; the margins, however, are all very small. For the Operation class, it is interesting to notice that SMOTE considerably improves recall: from 0.5753 to 0.7877. The overall F1 score for that class drops, though, because while improving recall, SMOTE severely harms precision, which goes from 0.7500 to 0.4781, indicating that with the oversampling executed by SMOTE many tickets that are not part of the Operation class get labeled as such, and as it throws a bigger amount of tickets into the Operation bin, the algorithm also recovers a bigger portion of the tickets of that category.

Table 33 – L1 (Experiment III – Best Results – SMOTE)

Class	Precision		Recall		F1		Number of Records
	PC	SMOTE	PC	SMOTE	PC	SMOTE	
Backup	0.9305	**0.9390**	**0.9564**	0.9104	**0.9433**	0.9244	826
Control	0.8103	**0.8233**	0.8650	**0.8773**	0.8368	**0.8480**	163
Job	0.7784	**0.7818**	**0.6681**	0.6261	**0.7195**	0.6943	238
Monitoring	0.9358	**0.9508**	**0.9557**	0.9245	**0.9456**	0.9373	1,602
Operation	**0.7500**	0.4781	0.5753	**0.7877**	**0.6512**	0.5892	146
Average / Total	**0.9058**	0.9038	**0.9092**	0.8874	**0.9065**	0.8923	2,975

Classifier L2

Table 34 shows the overall results obtained for classifier L2. The best result for each class is highlighted in bold. Given that in Experiment I classifier L2 did not present any signs that it was suffering from performance problems related to class imbalance, it is not surprising to see that all data level solutions fail to produce – for any of the classes – results that are superior to those of PC in relation to the F1 score. SMOTE-ENN performs particularly poorly, having (by a good margin) the worst results for all classes. The other algorithms have similar results, staying generally close to the performance of PC but never overcoming it.

The negative results of SMOTE-ENN can be attributed to its poor performance in precision for classes Database and Internal Network (0.7485 and 0.7574 respectively) and in recall for classes Application and Platform (0.8200 and 0.8030 respectively). As Table 35, which displays the detailed results of the best-performing algorithm for classifier L2 (ROS), shows, those numbers are far below those obtained by both PC and ROS.

Table 34 – L2 (Experiment III – Overall Results)

Class	PC	SMOTE	ROS	SMOTE-ENN	SMOTE-TL	Number of Records
Application	**0.8847**	0.8821	0.8812	0.8447	0.8814	8,952
Database	**0.8497**	0.8462	0.8479	0.8058	0.8425	4,077
Internal Network	**0.9035**	0.8980	0.8983	0.8348	0.9013	1,761
Platform	**0.8609**	0.8551	0.8570	0.8304	0.8561	8,549
Average / Total	**0.8713**	0.8672	0.8678	0.8319	0.8668	23,339

Table 35 reveals the only observed gains of ROS in relation to PC appear in the precision of the Database class and in the recall of the Internal Network tickets; in both cases, the numbers rise slightly. The latter improvement, though, comes at the cost of a bigger decrease in the precision of the classification of that class, which falls from 0.9116 to 0.8871. Everywhere else, there is no considerable disparity between the scores: ROS is consistently worse than PC, but the gap between the performances is small, which reflects on metrics whose weighted averages are always close to each other.

Table 35 – L2 (Experiment III – Best Results – ROS)

Class	Precision		Recall		F1		Number of Records
	PC	ROS	PC	ROS	PC	ROS	
Application	**0.8754**	0.8730	**0.8941**	0.8896	**0.8847**	0.8812	8,952
Database	0.8601	**0.8621**	**0.8396**	0.8344	**0.8497**	0.8479	4,077
Internal Network	**0.9116**	0.8871	0.8955	**0.9097**	**0.9035**	0.8983	1,761
Platform	**0.8641**	0.8614	**0.8578**	0.8526	**0.8609**	0.8570	8,549
Average / Total	**0.8713**	0.8679	**0.8714**	0.8679	**0.8713**	0.8678	23,339

Overview

In Experiment III, it was possible to see that, in some specific cases, the data level solutions contributed to the achievement of better results in the classification of some classes. For classifier L1 vs. L2, the former class as well as the whole classifier itself reached a higher level of F1 score when SMOTE was applied; for classifier L0, that improvement happened for classes Cloud and External Network (which is, surprisingly, a majority class), and that better performance ended up increasing the overall result of the classifier when SMOTE was applied; finally, for classifier L1, the class Control was also better classified with SMOTE than when the pure classifier of Experiment I was employed.

Therefore, in all classifiers where a gain in performance occurred, whether in one or more classes or in relation to the classifier itself, SMOTE was involved. For classifiers L0 vs. Non L0, and L2, where improvements in the F1 score were not seen, SMOTE was not the top-performing algorithm, but in both cases it came extremely close to the top. In the case of L0 vs. Non L0, the best score, achieved by SMOTE-TL, was 0.9649; and SMOTE reached 0.9647. Meanwhile, in the case of L2, the best score, achieved by ROS, was 0.8678; and SMOTE reached 0.8672.

The sole data-balancing algorithm that did not come out on top for any of the classifiers was also a considerable negative highlight in the experiment as a whole. SMOTE- ENN was the worst-performing algorithm for all classifiers, and in some cases (L1 and L2) the gap between its achieved F1 score and that of the second-to-last algorithm was big.

At the end of Experiment III, when considering even the cases where the execution of data-balancing did not make classifiers outperform the results they had achieved Experiment I, the selected configuration of each one of them was:

- **L0 vs. Non L0:** TF and RF; Binary IFDR at 25%; SMOTE-TL.
- **L1 vs. L2:** TF and RF; Frequency IFDR at 25%; SMOTE.
- **L0:** TF-IDF and SVM; Binary IFDR at 25%; SMOTE.
- **L1:** TF-LOG and SVM; Binary IFDR at 25%; SMOTE.
- **L2:** TF and NN; Frequency IFDR at 25%; ROS.

5.3.5 Experiment IV – Imbalanced Data (Ensemble Solution)

Experiment IV, like Experiment III, tries to address the problem of class-imbalance for all classifiers of Incident Routing's classification tree. However, while Experiment III tries to do so with data level solutions, Experiment IV looks to the ensembles bagging and boosting, which have been reported to achieve good results in situations where class-imbalance occurs (GALAR *et al.*, 2014) (WANG; YAO, 2012) and, in the case of AdaBoost, to improve the effects of data level solutions when applied in conjunction with those (BARUA *et al.*, 2014).

However, boosting and bagging are usually employed with weak classifiers (MAYR *et al.*, 2014) and research indicates performance degradation can occur when they are built with strong learning models (DONG; HAN, 2005), such as SVM.

Yet, as it stands, after the results of Experiment I, all classifiers of Incident Routing's classification tree are configured with strong learning models: NN, RF, and SVM. For the purpose of Experiment IV, then, all classifiers are replaced with DTs. The setup of each classifier is, therefore, as follows:

- **L0 vs. Non L0:** TF and DT; Binary IFDR at 25%; SMOTE-TL.
- **L1 vs. L2:** TF and DT; Frequency IFDR at 25%; SMOTE.
- **L0:** TF-IDF and DT; Binary IFDR at 25%; SMOTE.
- **L1:** TF-LOG and DT; Binary IFDR at 25%; SMOTE.
- **L2:** TF and DT; Frequency IFDR at 25%; ROS.

Similarly to what happened in Experiment III, for both boosting and bagging, the procedure that was followed was the same. The 5-fold cross-validation was executed, and for each fold – that is, in every one of its five iterations – the data-balancing approaches selected in Experiment III were applied on the four pieces used for training (as shown in Figure 20); the data contained in the one piece of each iteration that is used for testing (or validation) was, then, ignored during the data-balancing procedure.

The results of the experiment are reported for each class of the five classifiers. The F1 scores obtained through all ensemble solutions are presented as well as their average as weighted according to the number of records of each class. For comparative purposes, the F1 scores obtained for all classes on Experiment I are also shown; in those cases, the scores are those of the best classifiers that were selected at the end of the experiment. Those scores are denoted by PC, which stands for pure classifiers; in other words, the classifiers without the feature selection of Experiment II and the data-balancing of Experiment III.

Classifier L0 vs. Non L0

Table 36 shows the overall results obtained for classifier L0 vs. Non L0. The best result for each class is highlighted in bold. As it happened in Experiment III, no improvements in terms of the F1 score for each class can be noticed. For both classes, boosting outperforms bagging.

Table 36 – L0 vs. Non L0 (Experiment IV – Overall Results)

Class	PC	SMOTE-TL Boosting	SMOTE-TL Bagging	Number of Records
L0	**0.8879**	0.8732	0.8695	4,832
Non L0	**0.9799**	0.9772	0.9760	26,314
Average / Total	**0.9655**	0.9610	0.9595	31,146

Table 37 shows the detailed results obtained with SMOTE-TL and boosting, the best-performing combination, for classifier L0 vs. Non L0. The best results for each metric for each class are highlighted in bold. Differently from what was seen in Experiment III, even when drilling down on the results and analyzing precision and recall, the mixture of a data level solution for class imbalance and an ensemble does not bring any noticeable gains.

Table 37 – L0 vs. Non L0 (Experiment IV – Best Results – SMOTE-TL / Boosting)

Class	Precision		Recall		F1		Number of Records
	PC	SMOTE-TL Boosting	PC	SMOTE-TL Boosting	PC	SMOTE-TL Boosting	
L0	**0.9061**	0.8889	**0.8704**	0.8582	**0.8879**	0.8732	4,832
Non L0	**0.9764**	0.9741	**0.9834**	0.9803	**0.9799**	0.9772	26,314
Average / Total	**0.9655**	0.9609	**0.9659**	0.9614	**0.9655**	0.9610	31,146

Classifier L1 vs. L2

Table 38 shows the overall results obtained for classifier L1 vs. L2. The best result for each class is highlighted in bold. In this case, the least-represented class (L1) does achieve a gain of performance when both SMOTE and boosting are applied in conjunction. Differently from what happened in Experiment III, though, that increase is not enough to make the classifier's weighted F1 score be bigger when data balancing occurs than when it does not. It is also worthy of note how that gain does not happen when bagging is applied. Boosting once more achieves the best result among both configurations even if it does not surpass PC.

Table 38 – L1 vs. L2 (Experiment IV – Overall Results)

Class	PC	SMOTE Boosting	SMOTE Bagging	Number of Records
L1	0.7280	**0.7307**	0.7194	2,975
L2	**0.9687**	0.9669	0.9642	23,339
Average / Total	**0.9415**	0.9402	0.9365	26,314

Table 39 shows the detailed results obtained with SMOTE and boosting, the best-performing combination, for classifier L1 vs. L2. The best results for each metric for each

class are highlighted in bold. SMOTE and boosting are able to improve the F1 score of class L1 by increasing the recall from 0.6639 to 0.7062 and sacrificing the precision, which falls from 0.8058 to 0.7578. Moreover, there is a slight gain in the precision metric for class L2, a result that does not reflect on a better F1 score because there is a bigger drop in the recall metric for that class when boosting and SMOTE are used.

Table 39 – L1 vs. L2 (Experiment IV – Best Results – SMOTE / Boosting)

Class	Precision		Recall		F1		Number of Records
	PC	SMOTE Boosting	PC	SMOTE Boosting	PC	SMOTE Boosting	
L1	**0.8058**	0.7578	0.6639	**0.7062**	0.7280	**0.7307**	2,975
L2	0.9581	**0.9629**	**0.9796**	0.9710	**0.9687**	0.9669	23,339
Average / Total	**0.9409**	0.9397	**0.9439**	0.9411	**0.9415**	0.9402	26,314

Classifier L0

Table 40 shows the overall results obtained for classifier L0. The best result for each class is highlighted in bold. While in Experiment III the data-balancing approaches had brought better results for the Cloud and External Network classes, the mixture of SMOTE with either boosting or bagging does not do the same in this experiment. In fact, with the exception of the majority class External Network, for which the F1 scores of the three classifiers are even, the drops in performance are considerable; moreover, they happen for the minority classes, categories which the data level solutions and ensembles should have theoretically benefited. The most glaring degradation occurs for the Production Support class, for while classifier PC reaches 0.9091, bagging and boosting fail in getting to 0.8.

Table 40 – L0 (Experiment IV – Overall Results)

Class	PC	SMOTE Boosting	SMOTE Bagging	Number of Records
Cloud	**0.9143**	0.8558	0.8883	114
External Network	**0.9987**	0.9954	0.9976	4,239
Incident Management	**0.9368**	0.8519	0.8914	338
Production Support	**0.9091**	0.7931	0.7941	141
Average / Total	**0.9898**	0.9762	0.9817	4,832

Table 41 shows the detailed results obtained with SMOTE and bagging, the best-performing combination, for classifier L0. The best results for each metric for each class are highlighted in bold. The only gain, and one that is very marginal, happens in the recall metric of the Cloud class. It is possible to see that the massive degradation suffered by the Production Support class comes from both metrics, as the class' precision and recall when SMOTE and bagging are used are both significantly worse than the values of those metrics for PC. Additionally, save for the External Network class, it can be seen how the fall in performance is pretty much widespread: the precision and recall for all categories become much worse when SMOTE and bagging are applied. In the case of classifier L0, then, not only did the combination of a data-balancing approach with an ensemble method fail to improve results for minority classes, it also made the classification of those least-represented categories much worse.

Table 41 – L0 (Experiment IV – Best Results – SMOTE / Bagging)

Class	Precision		Recall		F1		Number of Records
	PC	SMOTE Bagging	PC	SMOTE Bagging	PC	SMOTE Bagging	
Cloud	**1.0**	0.9338	0.8421	**0.8509**	**0.9143**	0.8883	114
External Network	**0.9981**	0.9972	**0.9993**	0.9981	**0.9987**	0.9976	4,239
Incident Management	**0.9106**	0.8792	**0.9645**	0.9053	**0.9368**	0.8914	338
Production Support	**0.9328**	0.8091	**0.8865**	0.7801	**0.9091**	0.7941	141
Average / Total	**0.9901**	0.9819	**0.9899**	0.9818	**0.9898**	0.9817	4,832

Classifier L1

Table 42 shows the overall results obtained for classifier L1. The best result for each class is highlighted in bold. Repeating the pattern of Experiment III, most of the classes had their best F1 score in the pure classifier of Experiment I; meaning that data-balancing in conjunction with ensembles did not improve classification. While in Experiment III the only marginal improvement happened for class Control, in Experiment IV the only marginal improvement emerges for class Operation and is achieved by the combination of SMOTE and boosting, which once again outperforms bagging. For the class Job, both ensembles present a considerable performance degradation, with the one yielded by bagging being more significant. For all other classes, SMOTE and boosting are consistently close to PC, falling behind by small margins in F1 score, and consistently

ahead of SMOTE and bagging. That combination is a particular negative highlight on the Control class, where it reaches an F1 score of 0.7844 while the other two configurations operate above 0.82.

Table 42 – L1 (Experiment IV – Overall Results)

Class	PC	SMOTE Boosting	SMOTE Bagging	Number of Records
Backup	**0.9433**	0.9333	0.9246	826
Control	**0.8368**	0.8208	0.7844	163
Job	**0.7195**	0.6822	0.6392	238
Monitoring	**0.9456**	0.9387	0.9338	1,602
Operation	0.6512	**0.6586**	0.6451	146
Average / Total	**0.9065**	0.8965	0.8853	2,975

Table 43 shows the detailed results obtained with SMOTE and boosting, the best-performing combination, for classifier L1. The best results for each metric for each class are highlighted in bold. For classes Backup, Control, and Monitoring, SMOTE and boosting bring improvements to the precision metric; however, they also bring losses to recall, which causes the F1 scores for those classes to be inferior to those obtained by PC. For the Job class, there is a big improvement in terms of recall, but an even bigger drop regarding precision, which negatively impacts the F1 score for that class. The only category in which SMOTE and boosting outperform PC in classifier L1, the class Control, gets a gain in performance because while recall is enhanced, precision does not suffer that much, dropping from 0.7500 to 0.7328.

Table 43 – L1 (Experiment IV – Best Results – SMOTE / Boosting)

Class	Precision		Recall		F1		Number of Records
	PC	SMOTE Boosting	PC	SMOTE Boosting	PC	SMOTE Boosting	
Backup	0.9305	**0.9336**	**0.9564**	0.9334	**0.9433**	0.9333	826
Control	0.8103	**0.8305**	**0.8650**	0.8160	**0.8368**	0.8208	163
Job	**0.7784**	0.6541	0.6681	**0.7185**	**0.7195**	0.6822	238
Monitoring	0.9358	**0.9401**	**0.9557**	0.9376	**0.9456**	0.9387	1,602
Operation	**0.7500**	0.7328	0.5753	**0.6164**	0.6512	**0.6586**	146
Average / Total	**0.9058**	0.8992	**0.9092**	0.8965	**0.9065**	0.8965	2,975

Classifier L2

Table 44 shows the overall results obtained for classifier L2. The best result for each class is highlighted in bold. As it happened in Experiment III, no improvements in terms of the F1 score for each class can be noticed. Bagging outperforms boosting by a small margin.

Table 44 – L2 (Experiment IV – Overall Results)

Class	PC	ROS Boosting	ROS Bagging	Number of Records
Application	**0.8847**	0.8687	0.8722	8,952
Database	**0.8497**	0.8369	0.8438	4,077
Internal Network	**0.9035**	0.8999	0.8786	1,761
Platform	**0.8609**	0.8471	0.8466	8,549
Average / Total	**0.8713**	0.8580	0.8583	23,339

Table 45 shows the detailed results obtained with ROS and bagging, the best-performing combination, for classifier L2. The best results for each metric for each class are highlighted in bold. The only gains achieved by ROS and bagging come in the precision of class Platform and in the recall of class Internal Network. None of those improvements, however, are enough to increase the classes' F1 score, because the precision and recall achieved by ROS and bagging for, respectively, classes Internal Network and Platform are much worse than the ones reached by PC.

Table 45 – L2 (Experiment IV – Best Results – ROS / Bagging)

Class	Precision		Recall		F1		Number of Records
	PC	ROS Bagging	PC	ROS Bagging	PC	ROS Bagging	
Application	**0.8754**	0.8545	**0.8941**	0.8906	**0.8847**	0.8722	8,952
Database	**0.8601**	0.8573	**0.8396**	0.8308	**0.8497**	0.8438	4,077
Internal Network	**0.9116**	0.8517	0.8955	**0.9074**	**0.9035**	0.8786	1,761
Platform	0.8641	**0.8659**	**0.8578**	0.8283	**0.8609**	0.8466	8,549
Average / Total	**0.8713**	0.8589	**0.8714**	0.8586	**0.8713**	0.8583	23,339

Overview

In Experiment IV, none of the classifiers had their overall performance improved by the use of combinations of bagging and boosting with data-balancing methods, contrasting with Experiment III, where two classifiers had gains in terms of their weighted F1 score. In fact, those combinations were so unsuccessful that only two ticket categories – L1 for classifier L1 vs. L2 and Operation for classifier L1 – had better results when compared to the pure configurations of Experiment I. Moreover, for some classes like Cloud, Incident Management, Production Support, and Job, there was considerable performance degradation.

If, for each classifier, the best configurations obtained in Experiment IV are considered, whether or not they report improvements over previously used setups, the experiment ends with the following parameters:

- L0 vs. Non L0: TF and DT; Binary IFDR at 25%; SMOTE-TL and Boosting.
- L1 vs. L2: TF and DT; Frequency IFDR at 25%; SMOTE and Boosting.
- L0: TF-IDF and DT; Binary IFDR at 25%; SMOTE and Bagging.
- L1: TF-LOG and DT; Binary IFDR at 25%; SMOTE and Boosting.
- L2: TF and DT; Frequency IFDR at 25%; ROS and Bagging.

5.3.6 Experiment V – Testing Dataset

With the best text transformation techniques and learning models identified (Experiment I); the feature selection done for the five classifiers tasked with routing incident tickets to the correct areas (Experiment II); and the imbalanced nature of the datasets treated through data level and ensemble solutions (Experiment III and IV, respectively), the resulting best configurations for each of the classifiers in each experiment were applied to the test dataset.

The test dataset consists of 7,787 incident tickets. Since the goal is to evaluate the improvement brought by the use of classifiers in the routing of incident tickets in relation to the current manual setting, we use the forwarding cost (FC) metric to present the final results. For the current scenario, the test dataset would cause a forwarding cost of 7,787, as all tickets require one action of analysis and forwarding (as shown by Figure 3); for

the automatic scenario, meanwhile, the forwarding cost is given by the total number of misclassified tickets multiplied by 2, which is how many times tickets that do not reach the specialized teams fed by classifiers L0, L1, and L2 need to be forwarded to reach the area that is actually responsible for them (as shown by Figure 19).

In addition to the resulting classification trees of the four previous experiments, another three classification methods were employed. Firstly, a baseline classifier was constructed; that baseline classifier is an NB model that tries to simultaneously categorize tickets of all thirteen classes; that is, it does not employ the proposed tree structure shown in Figure 4. Secondly, the configurations obtained in Experiment IV were tested without the ensembles. The goal of this particular classification tree is to evaluate the gains (if any) the use of boosting and bagging brought to the balanced datasets. Finally, a super classification tree was put together by configuring each classifier with the setup that, through the course of the first four experiments, yielded the best results in terms of the overall F1 score. Table 46 shows how the F1 scores for each classifier progressed as the experiments went along and highlights the best configuration for each classifier in bold.

Table 46 – F1 Scores (Experiments I-IV)

Classifier	Experiment I	Experiment II	Experiment III	Experiment IV
L0 vs. Non L0	**0.9656**	0.9649	0.9649	0.9610
L1 vs. L2	0.9415	**0.9441**	0.9424	0.9402
L0	0.9898	0.9891	**0.9899**	0.9817
L1	**0.9065**	0.9052	0.8901	0.8965
L2	**0.8713**	0.8697	0.8678	0.8583

With that, in the super classification tree, L0 vs. Non L0, L1, and L2 were configured as the pure classifiers of Experiment I; L1 vs. L2 was configured as the classifier with feature selection of Experiment II; and L0 was configured as the classifier with feature selection and data balancing of Experiment III.

Table 46 shows the final results. The column Total has the number of tickets for each class in the testing dataset. The gain is reported according to how much the of the workload (in terms of forwarding cost) is lifted from the service desk if that classification tree is employed. For each class, the best result is highlighted in green; the worst result is highlighted in red. The baseline classifier was not included in the coloring scheme.

The columns of the table represent the classification methods, which learned patterns for the classes by using the training portion of the dataset (80% of the full set of

tickets) and were then used to classify the testing tickets (20% of the full set). These classification methods are:

- **BL:** The baseline Naïve Bayes classifier with TF-IDF that does not use a tree-like structure.
- **PC:** The pure classifiers of Experiment I.
- **FS:** The classifiers with feature selection of Experiment II.
- **DB:** The classifiers with feature selection and data-balancing of Experiment III.
- **EN:** The ensembles of decision trees with feature selection and data balancing of Experiment IV.
- **DTB:** The decision trees of Experiment IV with balancing but without the ensembles.
- **ST:** The super tree of classifiers built with the best overall result (selected by F1 Score) of each classifier in all experiments.

For BL, it is clear the learning model struggles with classes that are poorly represented in the dataset. For Cloud, Production Support, Control, Job, and Operation, BL is unable to rout a single incident ticket to the correct teams. And for Incident Management, just 1 ticket out of 37 successfully makes its way to the technical team responsible for solving tickets of the sort. Therefore, the choice of making a Naïve Bayes algorithm choose from one among thirteen classes (without the proposed tree-like classification structure) to file the ticket under proves to be quite harmful to minority categories. BL achieves surprising results for the Internal Network class, where it outperforms all other classification trees, being able to identify 26 tickets more than the second-best approach (PC). Everywhere else, though, BL is either outright the worst-performing approach or lies close to the worst one. Unsurprisingly, therefore, it is the classification method with the worst reported gain.

PC, FS, and DB, meanwhile, report gains that are similar to one another: 65.6%, 65.2%, and 63.5%, respectively. The difference rests in how they are achieved. PC does so with the whole set of attributes present in the dataset; FS reaches a result that is just 0.4% worse but by eliminating 75% of the attributes of the dataset, showing that the feature selection performed in Experiment II yielded the expected effects. In other words, it allowed classifiers to manage less attributes while keeping classification performance

on a similar level. One particular highlight of FS is how it improves the classification of the L1 class, from 425 correctly classified tickets in PC (the worst-performing approach) to 438. Given that class is part of an imbalanced classification problem (L1 vs. L2), it is possible to say the reduction in the number of attributes reduced the bias of the classifier to a certain degree.

The results of the classification trees for class L1 are, in fact, revealing. Results in Experiment III had indicated that the use of data-balancing techniques had brought noticeable gains in terms of F1 score for that group of tickets when compared to the numbers of Experiment I. And, here, they materialize. DB, DTB, and EN all perform data level solutions for class imbalance on the training dataset, and they perform better for that class than all other classification trees. Moreover, when it comes to the minority classes, the best results are usually found among those three classification trees. DB has the best results for L1, Backup, Control, and Operation, and is close to the best in Incident Management and Job; DTB has the best results for Cloud, Production Support, Incident Management, and Operation, and is close to the best in L1, Backup, Control, and Job; and EN has the best results for Cloud, Production Support, and Operation, and is close to the best in L1, Incident Management, Control, and Job.

DB and EN get to results that are close to the best ones by performing consistently better than the other classifiers in those minority classes. They usually fall behind PC and FS in relation to classes that are well-represented, but they make up for that gap by regularly outperforming those classification trees in the least-represented categories. And, alongside DTB, they show the data-balancing that was executed produced positive results. DB is a particularly solid classification tree: it is the worst-performing tree for none of the classes, and it feeds with a certain degree of success all of the teams handling incident tickets. It may not present the largest gain, but it gets close to that in a balanced way.

DTB was deployed so that the gain in using the ensembles bagging and boosting alongside data-balancing approaches could be evaluated in its comparison with EN, which uses the same configuration as DTB but with those ensembles being executed. And, in that regard, as Experiment IV had indicated, the results show no noticeable gain, as DTB gets similar numbers to EN in many minority classes. The negative effect that the lack of boosting and bagging causes for DTB is that the decision trees struggle for the

classes of L2 when they are not working inside an ensemble: DTB is the worst classification tree for all classes of L2.

Table 47 – Testing Dataset Results

		BL	PC	FS	DB	DTB	EN	ST	Total
L0 vs. Non L0	**L0**	-	1,082	1,074	1,078	1,071	1,078	1,081	**1,237**
	Non L0	-	6,454	6,465	6,450	6,407	6,441	6,452	**6,550**
L1 vs. L2	**L1**	-	425	438	464	458	460	433	**763**
	L2	-	5,660	5,653	5,591	5,465	5,573	5,662	**5,787**
L0	**Cloud**	0	7	10	8	11	11	5	**26**
	External Network	1,077	1,051	1,037	1,040	1,017	1,023	1,048	**1,094**
	Incident Management	1	15	18	19	21	19	18	**73**
	Production Support	0	9	8	11	17	17	10	**44**
L1	**Backup**	91	158	147	161	153	154	148	**222**
	Control	0	22	22	25	23	23	24	**42**
	Job	0	21	23	27	26	30	32	**65**
	Monitoring	101	200	217	211	209	213	193	**403**
	Operation	0	8	7	13	13	13	11	**31**
L2	**Application**	1,946	1,980	1,974	1,943	1,883	1,964	2,004	**2,204**
	Database	651	836	828	819	803	830	828	**1,050**
	Internal Network	309	375	377	373	357	366	380	**426**
	Plataform	1,576	1,764	1,763	1,717	1,645	1,682	1,756	**2,107**
Results	**Forwarding Cost**	**4,050**	**2,682**	**2,712**	**2,840**	**3,218**	**2,888**	**2,642**	-
	Gain (%)	**47.9**	**65.6**	**65.2**	**63.5**	**58.7**	**62.9**	**66.1**	-

The created super classification tree (ST) achieves, by a margin of 0.5% over PC, the largest gain in terms of forwarding cost. However, that classification tree presents the problem that its classification is biased towards the level with the most numerous set of tickets: L2. For all classes of that level, ST is either the classification tree with the best results or is close to the best-performing ones, a success that is responsible for its superior gain in terms of workload reduction, given that for many of the other classes ST is among the worst-performing methods. In fact, for minority classes, ST struggles. The only minority class for which it excels is that of L2: Internal Network. Therefore, if it were chosen to rout tickets to their correct destinations, it would fall behind DB when it comes to keeping all specialized teams well-fed with tickets.

Outside of class L1, the gain observed for the minority classes may not have been very significant when data-balancing algorithms were executed; the improvements were steady, but small. By looking at Table 48 and Table 49, which are the confusion matrixes for the classification tree DB when it comes to levels L0 and L1, the reason behind that small improvement is revealed.

Table 48 – Confusion Matrix (DB – L0)

	Cloud	**External Network**	**Incident Management**	**Production Support**
Cloud	8	0	0	0
External Network	1	1,040	0	0
Incident Management	0	0	19	0
Production Support	0	0	0	11

Table 49 – Confusion Matrix (DB – Ll)

	Backup	**Control**	**Job**	**Monitoring**	**Operation**
Backup	161	1	2	0	9
Control	0	25	0	0	0
Job	2	3	27	0	0
Monitoring	1	1	4	211	1
Operation	3	0	0	0	13

Both classifiers are very successful in the labeling of the tickets of L0 and L1. If a ticket arrives to the classifier, it will most likely be sent to the correct team. That means that even though the total of tickets that is correctly sent to the teams of those levels is

not very high, the problem does not lie in those classifiers, but in L0 vs. Non L0 and L1 vs.L2, which are the points where improvements can be achieved in terms of classification so that tickets from those two levels are more accurately identified and sent to their respective classifiers, since the proportion of tickets recovered for them is low compared to that of L2.

5.4 Concept Drift Treatment

5.4.1 Methodology

With the refinement experiments that sought to improve the classification of the tickets to its maximum level and achieve the best-possible results in terms of workload reduction done, the goal shifts to the evaluation of the existence of concept drift in the field of IT incidents. With that target in mind, the scope of this experiment is to measure how learning with more recent tickets (and ignoring those from a more distant past) brings different results when compared to learning with all tickets.

As so, 7 classification trees as the one shown in Figure 4 were created and trained. Following the procedure described in the FLORA algorithm, the training of each one of the classifiers that compose them was done by only considering tickets from distinct time windows that preceded the month that was being evaluated. The names given to the classification trees as well as the data within their training windows are:

- **K1:** 1 Month Prior to Month Being Evaluated.
- **K2:** 2 Months Prior to Month Being Evaluated.
- **K3:** 3 Months Prior to Month Being Evaluated.
- **K4:** 4 Months Prior to Month Being Evaluated.
- **K5:** 5 Months Prior to Month Being Evaluated.
- **K6:** 6 Months Prior to Month Being Evaluated.
- **KN:** All Months Prior to Month Being Evaluated.

Therefore, the cumulative classification tree KN serves as the experiment's baseline, for by learning with all tickets that were created before the start of the current

month, it ignores the presence of concept drift in the dataset. In addition to those 7 classification trees, the proposed ensembles of Section 3.5.2 were also tested. Given both short and long learning windows have their own advantages and disadvantages, as previously discussed, these ensembles look to balance those characteristics and put together classification strategies that outperform the fixed training windows of the FLORA algorithm by combining their decisions through different voting schemes. The ensembles that will be executed concurrently with the classification trees K1-KN are:

- Unweighted Voting (UV);
- Weighted by Shortest Window (WSW);
- Weighted by Longest Window (WLW);
- Weighted Monthly (WM);
- Weighted by Classifier Accuracy (WCA);
- Weighted by Classifier Total (WCT).

As the data collected for the execution of the concept drift experiment represents a period of fifteen months, and the longest training window tested was of 6 months, the classifier and ensemble trees were compared during the remaining 9 months of the dataset. Their performance was evaluated by considering the total percentage of tickets that were successfully forwarded to the correct areas, which represents a reduction of the workload (measured by forwarding cost) that is dedicated to manually analyzing the tickets and sending them to the technical group that is capable of solving them. The 9 months evaluated in the test were as follows:

- **M1:** 10/25/2016 to 11/24/2016.
- **M2:** 11/25/2016 to 12/24/2016.
- **M3:** 12/25/2016 to 01/24/2017.
- **M4:** 01/25/2017 to 02/24/2017.
- **M5:** 02/25/2017 to 03/24/2017.
- **M6:** 03/25/2017 to 04/24/2017.
- **M7:** 04/25/2017 to 05/24/2017.
- **M8:** 05/25/2017 to 06/24/2017.
- **M9:** 06/25/2017 to 07/24/2017.

It is worthy to highlight that given the total number of months of data that preceded the first month is 6, in the evaluation of that month both K6 and KN (the cumulative classifier) were trained with the same subset.

Additionally, since the configurations of the classifiers that were obtained through the classification refinement experiments considered tickets that are now part of the testing subset, given the dataset is now divided from a temporal standpoint that had not been previously considered, all classifiers were reconfigured to a standard setup. That configuration is: TF-IDF for text transformation; Linear SVM for classification; and ROS for data-balancing.

5.4.2 Experiment VI – Concept Drift

Figure 28 shows, with lines corresponding to each of the 13 classifiers, how they performed for each of the months evaluated. As it is possible to see, with the exception of the third month, all classifier trees are able to reach good percentages of correctly classified tickets.

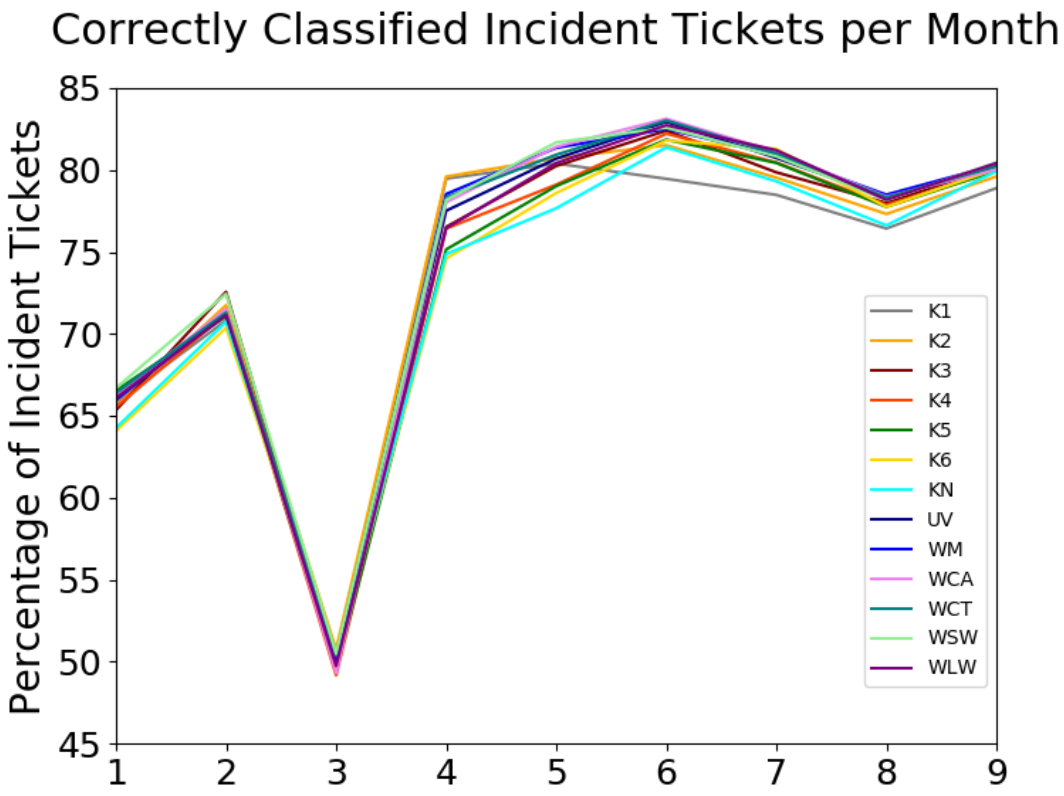

Figure 28 – Monthly Performance Of Classification Trees

The global deterioration of performance that occurs during the third month (the period between 12/25/2016 and 01/24/2017) can be explained by how prior to this period External Network tickets were almost non-existent. Before that month, only 5 External Network incidents had been registered; in the third month, though, a surge occurred and 869 tickets belonging to that category appeared, which caused classifiers not to have enough data to identify the patterns present in the class and misclassify a considerable portion of them. The second month also presented a similar phenomenon, as the number of Cloud tickets rose from 1 (prior to that period) to 29, which also caused classifiers to struggle; however, as the number of Cloud tickets is relatively insignificant compared to the whole dataset such an issue is not visible on the chart.

Figure 28 also reveals that the percentages of correctly classified tickets per month are very similar for all 13 classification trees; nonetheless, there is a certain consistency in how some of them fare better than others. KN, the cumulative tree that serves as the experiment's baseline is constantly among the worst-performing trees; and the same goes for K6, showing that longer learning windows are not beneficial to the classification of the tickets of the dataset and that concept drift is occurring. On the other hand of the spectrum, it is possible to see that the lines representing the created ensembles are usually among the best for any given month, which shows the success of the proposed aggregation strategies.

Table 50 – Monthly Performance Ranking Of Classification Trees

	M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**
1st	WSW	K3	K2	K2	WSW	WCA	K6	WM	WSW
2nd	K5	WSW	WSW	K1	WCA	WCT	WLW	WCA	WLW
3rd	WCT	K2	K1	WM	WM	UV	WCA	UV	WM
4th	WCA	WCA	KN	WCT	WCT	WLW	WCT	WCT	UV
5th	UV	WCT	WCT	WSW	K2	WSW	WM	WLW	K3
6th	WLW	UV	K6	WCA	UV	WM	WSW	WSW	WCT
7th	WM	WM	WM	UV	WLW	K3	UV	K3	K6
8th	K1	K4	UV	K3	K1	K4	K4	K4	WCA
9th	K2	WLW	K3	WLW	K3	K5	K5	K5	K5
10th	K4	K5	WLW	K4	K4	K6	K3	K6	K4
11th	K3	K1	K5	K5	K5	K2	K2	K2	KN
12th	KN	KN	WCA	KN	K6	KN	KN	KN	K2
13th	K6	K6	K4	K6	KN	K1	K1	K1	K1

Table 50 presents the information of Figure 28 in a different way: although percentages cannot be seen, the monthly raking of all trees (according to the number of correctly classified tickets) is clearer and can be further analyzed. Columns represent the 9 months whose tickets were evaluated, while rows represent the rank for that given month. Save for in M3, the worst-performing classification tree is either one with a long window (K6 and KN) or with a very short learning period (K1, which takes the bottom position of the table during the final four months of the experiment); a fact that shows neither extreme is ideal, even if K1 has good performances in M3 and M4. With ensembles excluded from the equation, the best learning windows seem to be those that lie in between the poles: K2 and K3. They are, still, slightly inconsistent: K2 is always among the top-ranked trees from M1 to M5, but it drops down considerably from M6 to M9; and K3 is the best-performing non-ensemble in 4 of the 9 months, but it struggles in M1 and M7 while staying close to the middle of the pack everywhere else.

As the longest windows (K6 and KN) are the least-effective ones, it is not surprising to notice that the ensemble that gives more weight to their votes (WLW) usually ranks among the last of the ensemble solutions; conversely, due to the positive results of K2 and K3, as well as the occasional success of K1 in M3 and M4, the ensemble that gives more weight to the votes of the shortest windows (WSW) is naturally constantly reaching a very good performance level, being the best-performing ensemble during five of the 9 months.

Table 51 shifts the analysis from the month dimension to the evaluation of the five classifiers of the tree through the 9-month period that was tested. It shows which learning windows were the best for the performance (once more evaluated via the number of correctly classified tickets) of each classifier. Through that perspective, among non-ensembles, K2 and K3 are even bigger highlights than they were in the monthly perspective: they are always floating inside the three best-performing trees of that class. Meanwhile, KN is the worst tree for all classifiers save for L1, and – once again – it is joined at the bottom of the list by K6, and also K5; the former never rising above the 11th place and the latter having its best performance for L2, to which it is only the 9th-best classification tree. When it comes to ensembles, WSW repeats its success, always being among the three best-performing voting strategies; with no other ensembles being so consistent in their rank; all of the proposed strategies are, nonetheless, constantly

outperforming the fixed sliding learning windows of the FLORA algorithm, further validating that ensemble strategies are the better option when dealing with concept drift.

Table 51 – 9-Month Performance Ranking Of Classification Trees Per Classifier

	L0 vs. Non L0	**L1 vs. L2**	**L0**	**L1**	**L2**
1st	K2	WSW	WM	K3	WSW
2nd	WCA	WCT	UV	K2	WCT
3rd	WM	WCA	WSW	K4	WM
4th	WSW	WM	WCA	WM	WCA
5th	WCT	UV	WLW	WCA	UV
6th	K1	K3	WCT	WSW	WLW
7th	UV	WLW	K2	WLW	K2
8th	WLW	K2	K1	WCT	K3
9th	K3	K1	K3	UV	K5
10th	K4	K4	K4	K5	K4
11th	K5	K5	K5	K6	K6
12th	K6	K6	K6	KN	K1
13th	KN	KN	KN	K1	KN

Table 52, Table 53, Table 54, and Table 55 drill down on the view by classifier and explore how each employed strategy fared when it comes to all classes of the dataset across the period of the experiment. As expected, based on previous results, shorter windows again tend to be more accurate in the classification of tickets and ensembles are usually better than the standalone learning windows. There are, however, exceptions to the rule. In Table 52, for example, the rankings of the classification trees are consistent for classes L0, Non L0, and L2. For L1, though, results shift. With the exception of KN, which still does pretty poorly and ranks second-to-last (as opposed to last for all other classes), longer windows overcome shorter ones. K4, K5, K6, and WLW (which gives more weight to the votes of longer windows) outperform generally successful classification trees like K2 and WSW.

It is worth noting that L1 is a minority class for classifier L1 vs. L2, and the performance of its classification was improved during the data-balancing experiments. Given L1 tickets are not very common when compared to those of L2, classifiers that work with longer windows have the advantage of being able to learn from a bigger

collection of tickets from that class; when windows are too short, examples of minority class are scarce. Therefore, it is possible K1 and K2 are struggling due to that matter.

Table 52 – 9-Month Performance Ranking Of Classification Trees Per Class I

	L0 vs. Non L0		**L1 vs. L2**	
	L0	**Non L0**	**L1**	**L2**
1st	WM	K2	K4	WSW
2nd	UV	WCA	K6	WCT
3rd	WSW	WCT	WLW	K1
4th	WLW	WSW	K5	WM
5th	WCA	WM	WCA	WCA
6th	WCT	K1	K3	UV
7th	K1	K3	UV	K2
8th	K2	UV	WSW	K3
9th	K3	K4	WM	WLW
10th	K4	WLW	K2	K4
11th	K5	K5	WCT	K5
12th	K6	K6	KN	K6
13th	KN	KN	K1	KN
Total Tickets	**10,051**	**20,979**	**3,392**	**17,587**
Percentage of Dataset (%)	**32.39**	**67.61**	**16.16**	**83.84**

In Table 53, which shows the results for the classes of L0, that pattern reappears. For the most numerous class of that subset, External Network, the shorter the learning window the better it performs, and K1 is the best classifier, correctly labeling 8,355 tickets versus 8,190 of K6 and 8,155 of KN. For the least numerous classes, that reality is inverted: K5, K6, and KN dominate the top positions for classes Incident Management and Production Support. Differences in performance between them and the trees that learn with shorter windows can, in fact, be big when the total number of tickets for each class is considered. For Incident Management, K6, KN, WLW and K5 correctly label 29, 29, 21, and 20 tickets; while K2, WSW, and K1 correctly label 15, 15, and 7 tickets. For Production Support, KN, K6, and K5 correctly label 47, 31, and 28 tickets; while WSW, K2, and K1 correctly label 16, 15, and 14 tickets.

As far as the poorly represented classes of L0 go, the Cloud category is an outlier in that sense and shorter learning windows such as K2, K4, and K3 overcome K5, K6,

and KN; however, it is worth noting how WLW does better than WSW, even if it is by the smallest of margins: 166 versus 165, a closeness in performance that does not reappear in any other class of L0.

Table 53 – 9-Month Performance Ranking Of Classification Trees Per Class II

	L0			
	Cloud	**External Network**	**Incident Management**	**Production Support**
1st	UV	K1	K6	KN
2nd	WM	WSW	KN	K6
3rd	K2	WM	WLW	K5
4th	K4	WCA	K5	K4
5th	WLW	UV	UV	WLW
6th	WSW	WCT	K3	UV
7th	WCA	WLW	K4	WCT
8th	K3	K2	WCA	K3
9th	WCT	K3	WM	WM
10th	K5	K4	WCT	WCA
11th	K6	K5	K2	WSW
12th	K1	K6	WSW	K2
13th	KN	KN	K1	K1
Total Tickets	**311**	**9,489**	**146**	**105**
Percentage of Dataset (%)	**3.09**	**94.40**	**1.46**	**1.05**

In Table 54, which shows the results for the classes of L1, minority classes Job and Operation are also better identified by classification trees that use longer learning windows. For class Operation, while KN correctly labels 31 tickets, K1 only successfully recovers 11 of them. For class Job, meanwhile, the difference between the number of correctly classified tickets achieved by the first and last-placed classification trees is smaller, but it still exists: K5 is able to send 41 tickets to the Job team, whereas K1 forwards 28 of them. For well-represented classes Backup and Monitoring, the opposite happens: classification trees with shorter windows do better. Once again, a minority class acts as an outlier; in this case, the class Control, for although it has a low number of tickets, K2 and K1 are the best non-ensembles. In fact, while K2 correctly classifies 61 control tickets, KN only correctly classifies 25 of them.

These variations in performance of the standalone classification trees with fixed learning windows validate the option to create ensembles. By joining the votes of different-sized learning windows, the good and bad characteristics of each one of them are balanced, and the quality of the classifications are improved. And as Table 52, Table 53, Table 54, and Table 55 show, ensemble strategies are constantly ahead of standalone classification trees.

Table 54 – 9-Month Performance Ranking Of Classification Trees Per Class III

	L1				
	Backup	**Control**	**Job**	**Monitoring**	**Operation**
1st	WSW	K2	K5	K3	KN
2nd	K2	WM	K6	K4	WLW
3rd	UV	WCT	WLW	K5	K6
4th	WM	WSW	UV	K6	K5
5th	WCA	WCA	WCA	K2	K4
6th	K3	UV	WCT	WM	UV
7th	WCT	WLW	KN	WCA	K2
8th	K4	K1	WM	WCT	WSW
9th	WLW	K3	K4	WLW	WM
10th	KN	K4	K3	WSW	WCA
11th	K5	K5	WSW	UV	WCT
12th	K6	K6	K2	K1	K3
13th	K1	KN	K1	KN	K1
Total Tickets	**1,577**	**103**	**168**	**1,427**	**117**
Percentage of Dataset (%)	**46.49**	**3.04**	**4.96**	**42.07**	**3.44**

Although through all experiments conducted L2 has shown itself to be the most balanced set of classes, Table 55 reveals that even the minority class of that group – Internal Network – replicates the phenomenon where longer learning windows are the best choice, with KN, K6, K5, and WLW occupying the top positions. For classes Database and Platform, the ensemble strategies do particularly well, and the Application class marks the sole point on the entire dataset in which the worst-performing classification tree is an ensemble.

Table 55 – 9-Month Performance Ranking Of Classification Trees Per Class IV

	L2			
	Application	**Database**	**Internal Network**	**Platform**
1st	K2	WM	KN	WSW
2nd	WSW	WCT	K6	UV
3rd	KN	WSW	K5	WCT
4th	WCT	UV	WLW	WLW
5th	WM	WCA	WCA	WCA
6th	K1	K5	WM	WM
7th	WCA	K3	UV	K3
8th	K3	WLW	K3	K4
9th	K6	K4	WCT	K1
10th	K5	KN	WSW	K5
11th	K4	K6	K4	K2
12th	UV	K2	K2	K6
13th	WLW	K1	K1	KN
Total Tickets	**6,284**	**3,542**	**1,105**	**6,656**
Percentage of Dataset (%)	**35.37**	**20.14**	**6.29**	**37.84**

All of those results culminate in Table 56, which displays the forwarding cost – the metric utilized to measure the gains in workload reduction achieved by the classifiers – for all classification trees during the period covered by the experiment. Through those 9 months, 31,030 tickets were created; given each ticket represents, in the current service desk structure that is in place, a forwarding cost of 1, the total forwarding cost for the dataset is the same as the number of tickets.

All of the 7 proposed ensemble strategies do better than the classification trees with fixed learning windows, confirming the value gained by aggregating the votes of the individual classifiers in order to make a decision on the class of the ticket. As Figure 28 had indicated, the minimal differences in the percentage of correctly classified tickets for each month produced a final result in which the gap between the strategies is tight: the difference between the best and worst approaches is of 565 tickets, which represent an extra forwarding cost of 1,130. Nevertheless, by seeking to treat concept drift in the scenario of incident ticket management, better results were achieved, for the worst-performing classification tree (KN) represents the case in which concept drift is ignored

and the full dataset is used in the learning. In this case, having more data to build the classifier did not yield a superior result; in fact, the total opposite happened.

Table 56 – 9-Month Overall Performance Ranking Of Classification Trees

Ranking	Classification Tree	Correctly Classified Tickets	Incorrectly Classified Tickets	Forwarding Cost	Gain (%)
1st	WSW	23,466	7,564	15,128	51.24
2nd	WCT	23,430	7,600	15,200	51.01
3rd	WM	23,418	7,612	15,224	50.93
4th	WCA	23,406	7,624	15,248	50.86
5th	UV	23,359	7,671	15,342	50.55
6th	WLW	23,307	7,723	15,446	50.22
7th	K2	23,292	7,738	15,476	50.12
8th	K3	23,241	7,789	15,578	49.79
9th	K4	23,151	7,879	15,758	49.21
10th	K5	23,113	7,917	15,834	48.97
11th	K1	23,065	7,965	15,930	48.66
12th	K6	23,053	7,977	15,954	48.58
13th	KN	22,901	8,129	16,258	47.60

Among non-ensemble strategies, as previous tables had shown, windows that lie between the too short and too big extremes fare better, with K2, K3, and K4 overcoming K5, K1, K6, and KN. Due to that, the ensemble that gave more weight to the votes of shortest windows (WSW) produced the best results, while the one that gave more weight to the votes of longest windows (WLW) ranked the worst among ensemble classification trees. Besides, overall, giving weight to the votes is a better idea than leaving them unweighted, because four of the five approaches that used weighted votes outperformed UV, the Unweighted Voting classifier. Moreover, adjusting the weight of the votes for each classifier individually (which is what WCT and WCA do) did not bring results that

were significantly better than weighting them in the same way throughout the whole tree according to the performance of the different-sized learning windows during the previous month (which is what WM does). Still, weighting votes according to the total of correctly classified tickets of classifiers (WCT) was better than doing so according to their accuracy (WCA).

Appendix A holds more details regarding the numbers of correctly classified tickets that produced the rankings shown in the tables of this experiment.

Chapter 6

Conclusions and Future Works

6.1 Conclusions

By using the combinations of text transformation techniques and learning models obtained through Experiment I, it was possible to report gains in the forwarding cost metric, reducing by 65.6% the analysis and forwarding work the routing of a ticket entails and diminishing the workload of the service desk team tasked with opening the tickets and assigning them. It is important to note the reported gains over the manual scenario currently in place assume service desk members make no mistakes in the forwarding of the tickets, meaning they could be even bigger.

The classifiers of Experiment I were not significantly improved by those built in the 3 subsequent experiments, as the best overall result (the one obtained by ST, the classification tree with the best configuration of each classifier in all experiments) was only 0.5% better. What changed, however, between PC and the others, was how the gains were obtained. FS had similar results to PC, but by using just 25% of the terms in the corpus, showing the effectiveness of IFDR outside the sentiment analysis realm in which it was originally tested. It is a reduction that can mean improvements related to how quickly the classifiers can be trained, something that can be significant as new incident tickets are added to the training dataset.

Meanwhile, both DB and EN – constructed with the goal of better classifying minority classes – tended to get the best results for underrepresented ticket categories. In the construction of those classification trees the data-balancing methods SMOTE, ROS, and SMOTE-TL stood out, while SMOTE-ENN failed to achieve good results. With those 3 selected data level solutions for class imbalance, minority classes such as L1, Incident Management, Production Support, Control, Job, and Operation were better identified by those trees of classifiers than by PC and FS. For most of them Incident Routing failed to recover more than half of the tickets; still, improvements were consistent nevertheless – even if not too big, which indicates there may be room for improvement regarding the treatment of those classes. The most noteworthy gain regarding the classification of

underrepresented categories came in the L1 class of the L1 vs. L2 classifier; out of its 763 tickets, only 425 were recovered by PC; however, when FS, DB, and EN were used, the correctly routed amount raised to 438, 464, and 460 respectively.

Finally, the BC classification tree topped all others in terms of sheer forwarding cost gain; still, it did so by being biased towards the most represented classes, which are those of L2 (Application, Database, Internal Network, and Platform). While it did well in those categories, it contrasted those results by being among the worst classification trees for many other less numerous classes, such as L1, Cloud, Incident Management, Monitoring, and Production Support.

With those results, it is possible to see text classification and machine learning can be used to improve the work of service desks and incident management, which are among the most critical aspects of IT service providing since if not treated in an efficient manner they can damage the image of companies due to outages, slowdowns, and other problems in important applications or other services. Moreover, feature selection (more specifically, the IFDR technique) as well as methods for treating imbalanced datasets proved to be useful in, respectively, reducing dimensionality while preserving F1 Score performance and enhancing the classification of minority classes.

Both ensembles explored in the first set of experiments (boosting and bagging), however, did not produce significant improvements for this dataset when applied over data-balancing approaches; SMOTE, ROS, and SMOTE-TL proved to be enough to deal with such issue. The only class in which the combination of bagging and boosting with data-balancing algorithms resulted in gains when compared to the scenario in which those algorithms were applied on their own was Production Support.

Moreover, when comparing the classification trees DTB (formed by decision trees with data-balancing approaches) and EN (formed by decision trees with data-balancing approaches and ensembles), there are no significant differences for the poorly represented classes, which goes to show the addition of ensembles did not do much to make the data-balancing approaches more effective.

Regardless of the approach used, and even including the baseline classifier that was outperformed by all other classification trees obtained during the classification refinement experiments (hence validating the strategy of using a tree-like classification structure), it was proven, through the forwarding fost metric, that the proposed Incident Routing approach can reduce the workload of service desks by lifting the task of

analyzing recorded incidents out of their shoulders and automatically sending those tickets to the specialized areas that will be capable of solving them. As tickets are solved by teams, they can automatically be labeled by the service desk system as belonging to that incident category and then fed into the training dataset that will be used to update the classifiers; consequently, creating an environment in which classifiers are able to keep up-to-date with the data distribution as time goes along.

Inside that context of time, Experiment VI showed that, like many other fields that have been approached under the perspective of the concept drift problem, incident ticket management – which has grown to be a rather important area in a business scenario where IT service providing is one of the bases of the world's economy – is also affected by hidden contexts that, over time, alter the behavior of the target function classifiers seek to learn. As a consequence, a consideration of the time dimension when addressing the classification of incidents can bring gains; care, however, as the conducted experiment has shown, must be taken when it comes to choosing the most appropriate training windows for the classifiers.

A very short training window that only considers the most recent tickets negatively affects the classification of minority classes, because under a very tight learning period there may not be enough examples of those classes for classifiers to be able to identify them effectively. Meanwhile, windows that are too long may be slow to react to concept drift and fail to perform as well as shorter windows. Therefore, as Experiment VI showed, the combination of shorter and longer windows inside ensembles that use different strategies to aggregate the votes of those classifiers so that items are identified yields better results than standalone windows of fixed size.

During the period of 9 months whose tickets were evaluated, the 6 proposed ensemble strategies outperformed not only all standalone training windows of size varying from 1 to 6 months but also the cumulative classification tree that considered the full set of past data in its training. In fact, that cumulative classifier was the worst-performing of all employed approaches, reporting a gain of 47.6% versus 51.24% achieved by the best classification tree, displaying how concept drift affects incidents that occur in a company's IT infrastructure.

6.2 Future Works

A detail that is rather noticeable in relation to Experiment V and Experiment VI is how the gains that were reported in terms of forwarding cost differ. In Experiment V, the best-performing classification tree (ST) eliminated 65.6% of the workload; in Experiment VI, meanwhile, the best performing classification tree (WSW) eliminated 51.24% of the workload. Such a gap can be attributed to many factors. Treating the available dataset of tickets by considering the time variable causes situations such as the one that occurred for the External Network tickets in M3. Before that month, only 5 tickets of the class had been reported; when that month arrived, though, 869 tickets of that class appeared, and – naturally – given they had almost no examples of tickets from that class, the classification trees failed to correctly identify them, which caused a considerable drop in performance. When the dataset is randomly divided into 2 pieces – one for training and one for testing, as it was done in Experiments I through V – such a problem does not occur, as all ticket patterns are already present in the data, and by extracting a subset of those tickets it is possible all patterns will be captured and classifiers will thus perform better in the testing dataset.

More importantly, however, the 13 classification trees of Experiment VI did not go through any of the refinements done in Experiments I through IV: the text transformation method of their classifiers was defined as TF-IDF; the learning model that was used was SVM; and the data-balancing algorithm employed was ROS. Those selections were not made based on comparisons to alternative methods to see which would be more appropriate; they were made based on the knowledge those techniques tend to work well. Additionally, no feature selection was done for those classifiers.

Consequently, there are two natural progressions to this work. Firstly, when it comes to the classification refinement experiments, more methods could be evaluated. In terms of feature selection, those alternatives could be Log Likelihood Ratio (DUNNING, 1993) and Chi-Squared Test (MANNING *et al.*, 1999). In the field of imbalanced datasets (FERNÁNDEZ *et al.*, 2013), not only are there other data level solutions besides the ones that were used, but algorithmic level and cost-sensitive solutions, two areas that were not approached in this work, also hold plenty of opportunities for improvement. Furthermore, Word2Vec, which did not yield the best results for any of the classifiers, could be more deeply explored, as it has the benefit of producing vectors that represent tickets with a

small number of dimensions and there is research (CHAWLA *et al.*, 2002) indicating high dimensionality creates classifiers that are more biased; particularly, its Doc2Vec variation as well as models that are pre-trained over a larger corpus (like GloVe) could be adapted to be used with the dataset that was explored. If employed in conjunction with data-balancing approaches, it could be possible to analyze if the lower dimensionality of the ticket vectors yielded by Word2Vec helps in the classification of the minority classes, especially those of L0 and L1, service desk levels for which Incident Routing has room for improvement. All of those refinements could, additionally, be conducted in conjunction with the treatment of concept drift; in other words, the work of the two sets of experiments of this work could be combined.

When it comes to concept drift on its own, the FLORA algorithm that uses learning windows of fixed size, ignoring data that is older than a certain threshold, is one of the area's first and simplest approaches to the problem. Thorough surveys on concept drift (ZLIOBAITĖ *et al.*, 2010) (GAMA *et al.*, 2014) have gone to great lengths to list other different solutions for the problem, some of which try to adjust the size of learning windows by detecting the occurrence of concept drift. Hence, not only could they be implemented in the domain of incident ticket management, but ensemble strategies such as the ones proposed in this work could be explored to combine the qualities of the different algorithms in order to produce stronger classification results. In relation to the fixed window sizes of the FLORA algorithm, smaller scales of window-update time could be tried, such as hourly, daily, or weekly updates, as opposed to the monthly time scale used to move the windows forward in the experiment that was executed in this work.

Furthermore, attempts to enhance the classification results themselves could be executed in a number of ways. Alternatives to the bag of words methods for text transformation that were used in this work and that yielded the best F1 Score for all classifiers could be explored, such as feature engineering techniques (SCOTT; MATWIN, 1999). Moreover, given text was always transformed into unigrams, bigrams – which have the advantage of capturing some information about the local word order of the text (JOULIN *et al.*, 2016) – could also be evaluated. Finally, the tree structure of Incident Routing was strongly based on the one employed in a similar work (MENEZES, 2009), starting by separating tickets from the top level of the service desk until the one that stands at the bottom (from L0 to L2, in the case of this work). Different tree configurations, then, could be explored.

References

ABU-MOSTAFA, Yaser S.; MAGDON-ISMAIL, Malik; LIN, Hsuan-Tien. Learning from data. New York, NY, USA:: AMLBook, 2012.

ALBERG, Dima; LAST, Mark. Short-Term Load Forecasting in Smart Meters with Sliding Window-Based ARIMA Algorithms. In: Asian Conference on Intelligent Information and Database Systems. Springer, Cham, 2017. p. 299-307.

ANDERSON, Monica. Technology Device Ownership: 2015. Pew Research Center, 2015. Available online in: http://www.pewinternet.org/2015/10/29/technology-device-ownership-2015/

BAHLI, Bouchaib; RIVARD, Suzanne. The Information Technology Outsourcing Risk: A Transaction Cost and Agency Theory-Based Perspective. In: Outsourcing and Offshoring Business Services. Palgrave Macmillan, Cham, 2017. p. 53-77.

BARUA, Sukarna et al. MWMOTE--majority weighted minority oversampling technique for imbalanced data set learning. IEEE Transactions on Knowledge and Data Engineering, v. 26, n. 2, p. 405-425, 2014.

BATISTA, Gustavo EAPA; PRATI, Ronaldo C.; MONARD, Maria Carolina. A study of the behavior of several methods for balancing machine learning training data. ACM Sigkdd Explorations Newsletter, v. 6, n. 1, p. 20-29, 2004.

BATISTA, Gustavo EAPA; BAZZAN, Ana LC; MONARD, Maria Carolina. Balancing Training Data for Automated Annotation of Keywords: a Case Study. In: WOB. 2003. p. 10-18.

BESSA, Ricardo J. et al. Information theoretic learning applied to wind power modeling. In: Neural Networks (IJCNN), The 2010 International Joint Conference on. IEEE, 2010. p. 1-8.

BLACK, Michaela; HICKEY, Ray. Detecting and adapting to concept drift in bioinformatics. Lecture Notes in Computer Science, p. 161-168, 2004.

BOGOJESKA, Jasmina et al. Impact of HW and OS type and currency on server availability derived from problem ticket analysis. In: Network Operations and Management Symposium (NOMS), 2014 IEEE. IEEE, 2014. p. 1-9.

BOJARSKI, Mariusz et al. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316, 2016.

BOLTON, Richard J.; HAND, David J. Statistical fraud detection: A review. Statistical science, p. 235-249, 2002.

BREIMAN, Leo. Bagging predictors. Machine learning, v. 24, n. 2, p. 123-140, 1996.

BROWN, Iain. An experimental comparison of classification techniques for imbalanced credit scoring data sets using SAS® Enterprise Miner. In: Proceedings of SAS Global Forum. 2012.

CHANDRASHEKAR, Girish; SAHIN, Ferat. A survey on feature selection methods. Computers & Electrical Engineering, v. 40, n. 1, p. 16-28, 2014.

CHAWLA, Nitesh V. et al. SMOTE: synthetic minority over-sampling technique. Journal of artificial intelligence research, v. 16, p. 321-357, 2002.

COWLEY, Benjamin Ultan; CHARLES, Darryl. Adaptive Artificial Intelligence in Games: Issues, Requirements, and a Solution through Behavlets-based General Player Modelling. arXiv preprint arXiv:1607.05028, 2016.

CUNNINGHAM, Padraig; DELANY, Sarah Jane. k-Nearest neighbour classifiers. Multiple Classifier Systems, v. 34, p. 1-17, 2007.

DAHLGAARD-PARK, Su Mi (Ed.). The sage encyclopedia of quality and the service economy. SAGE Publications, 2015.

DINATA, Ida Bagus Putu Peradnya; HARDIAN, Bob. Predicting smart home lighting behavior from sensors and user input using very fast decision tree with Kernel Density Estimation and improved Laplace correction. In: Advanced Computer Science and Information Systems (ICACSIS), 2014 International Conference on. IEEE, 2014. p. 171-175.

DONG, Yan-Shi; HAN, Ke-Song. Boosting SVM classifiers by ensemble. In: Special interest tracks and posters of the 14th international conference on World Wide Web. ACM, 2005. p. 1072-1073.

DUNNING, Ted. Accurate methods for the statistics of surprise and coincidence. Computational linguistics, v. 19, n. 1, p. 61-74, 1993.

ERL, Thomas. Service-oriented architecture (SOA): concepts, technology, and design. 2005.

FREUND, Yoav; SCHAPIRE, Robert E. A desicion-theoretic generalization of on-line learning and an application to boosting. In: European conference on computational learning theory. Springer, Berlin, Heidelberg, 1995. p. 23-37.

GALAR, Mikel et al. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), v. 42, n. 4, p. 463-484, 2012.

GAMA, João et al. A survey on concept drift adaptation. ACM Computing Surveys (CSUR), v. 46, n. 4, p. 44, 2014.

GUERRA, Pedro Calais; MEIRA JR, Wagner; CARDIE, Claire. Sentiment analysis on evolving social streams: How self-report imbalances can help. In: Proceedings of the 7th ACM international conference on Web search and data mining. ACM, 2014. p. 443-452.

HAASDIJK, Evert; BREDECHE, Nicolas; EIBEN, A. E. Combining environment-driven adaptation and task-driven optimisation in evolutionary robotics. PloS one, v. 9, n. 6, p. e98466, 2014.

HAN, Jiawei; PEI, Jian; KAMBER, Micheline. Data mining: concepts and techniques. Elsevier, 2011.

HAREN, Van. TOGAF Version 9.1. Van Haren Publishing, 2011.

HARTMANN, Nathan et al. Portuguese Word Embeddings: Evaluating on Word Analogies and Natural Language Tasks. arXiv preprint arXiv:1708.06025, 2017.

JONES, Rupert. Mobile banking on the rise as payment via apps soars by 54% in 2015. The Guardian, 2016. Available online in: <https://www.theguardian.com/business/2016/jul/22/mobile-banking-on-the-rise-as-payment-via-apps-soars-by-54-in-2015>

IDEN, Jon; EIKEBROKK, Tom Roar. Implementing IT Service Management: A systematic literature review. International Journal of Information Management, v. 33, n. 3, p. 512-523, 2013.

ITIL. The Official Introduction to the ITIL Service Lifecycle. ITIL Foundation, 2011.

JAN, Ea-Ee; CHEN, Kuan-Yu; IDÉ, Tsuyoshi. Probabilistic text analytics framework for information technology service desk tickets. In: Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on. IEEE, 2015. p. 870-873.

JOHNSON, Mark W. et al. Evolving standards for IT service management. IBM Systems Journal, v. 46, n. 3, p. 583-597, 2007.

JOULIN, Armand et al. Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759, 2016.

KIKUCHI, Shinji. Prediction of Workloads in Incident Management Based on Incident Ticket Updating History. In: Utility and Cloud Computing (UCC), 2015 IEEE/ACM 8th International Conference on. IEEE, 2015. p. 333-340.

KINGMA, Diederik; BA, Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

LAINHART IV, John W. COBIT™: A methodology for managing and controlling information and information technology risks and vulnerabilities. Journal of Information Systems, v. 14, n. s-1, p. 21-25, 2000.

LAN, Man et al. Supervised and traditional term weighting methods for automatic text categorization. IEEE transactions on pattern analysis and machine intelligence, v. 31, n. 4, p. 721-735, 2009.

LANE, Terran; BRODLEY, Carla E. Temporal sequence learning and data reduction for anomaly detection. ACM Transactions on Information and System Security (TISSEC), v. 2, n. 3, p. 295-331, 1999.

LEMAITRE, Guillaume; NOGUEIRA, Fernando; ARIDAS, Christos K. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. Journal of Machine Learning Research, v. 18, n. 17, p. 1-5, 2017.

LIFNA, C. S.; VIJAYALAKSHMI, M. Identifying concept-drift in twitter streams. Procedia Computer Science, v. 45, p. 86-94, 2015.

LILLEBERG, Joseph; ZHU, Yun; ZHANG, Yanqing. Support vector machines and word2vec for text classification with semantic features. In: Cognitive Informatics & Cognitive Computing (ICCI* CC), 2015 IEEE 14th International Conference on. IEEE, 2015. p. 136-140.

LIU, Yi-Hung; CHEN, Yen-Ting. Total margin based adaptive fuzzy support vector machines for multiview face recognition. In: Systems, Man and Cybernetics, 2005 IEEE International Conference on. IEEE, 2005. p. 1704-1711.

LÖFSTRÖM, Tuwe. On Effectively Creating Ensembles of Classifiers: Studies on Creation Strategies, Diversity and Predicting with Confidence. 2015. (Doctoral dissertation, Department of Computer and Systems Sciences, Stockholm University).

MANNING, Christopher D. et al. Foundations of statistical natural language processing. Cambridge: MIT press, 1999.

MARRONE, Mauricio et al. IT service management: A cross-national study of ITIL adoption. Communications of the association for information systems, v. 34, 2014.

MAYR, Andreas et al. The evolution of boosting algorithms. Methods of information in medicine, v. 53, n. 6, p. 419-427, 2014.

MAZHELIS, Oleksiy; PUURONEN, Seppo. Comparing classifier combining techniques for mobile-masquerader detection. In: Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on. IEEE, 2007. p. 465-472.

MAZUROWSKI, Maciej A. et al. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. Neural networks, v. 21, n. 2, p. 427-436, 2008.

MENEZES, João. Classificação de Texto para Melhoria no Atendimento de Help Desk. M.S. thesis, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, Brazil, 2009.

MIAO, Gengxin et al. Reliable ticket routing in expert networks. In: Reliable Knowledge Discovery. Springer, Boston, MA, 2012. p. 127-147.

MIKOLOV, Tomas et al. Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. 2013a. p. 3111-3119.

MIKOLOV, Tomas et al. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781, 2013b.

MIKOLOV, Tomas; YIH, Wen-tau; ZWEIG, Geoffrey. Linguistic regularities in continuous space word representations. In: hlt-Naacl. 2013c. p. 746-751.

MIRTALAIE, Monireh Alsadat et al. A decision support framework for identifying novel ideas in new product development from cross-domain analysis. Information Systems, v. 69, p. 59-80, 2017.

MITCHELL, Tom Michael. The discipline of machine learning. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

MOHARRERI, Kayhan; RAMANATHAN, Jayashree; RAMNATH, Rajiv. Motivating dynamic features for resolution time estimation within IT operations management. In: Big Data (Big Data), 2016a IEEE International Conference on. IEEE, 2016a. p. 2103-2108.

MOHARRERI, Kayhan; RAMANATHAN, Jayashree; RAMNATH, Rajiv. Probabilistic sequence modeling for trustworthy it servicing by collective expert networks. In: Computer Software and Applications Conference (COMPSAC), 2016b IEEE 40th Annual. IEEE, 2016. p. 379-389.

MURTHY, Sreerama K. Automatic construction of decision trees from data: A multi-disciplinary survey. Data mining and knowledge discovery, v. 2, n. 4, p. 345-389, 1998.

OSTERWALDER, Alexander et al. Value proposition design: How to create products and services customers want. John Wiley & Sons, 2014.

PALSHIKAR, Girish Keshav et al. Streamlining service levels for it infrastructure support. In: Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on. IEEE, 2012. p. 309-316.

PANG, Bo; LEE, Lillian. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In: Proceedings of the 43rd annual meeting on association for computational linguistics. Association for Computational Linguistics, 2005. p. 115-124.

PANG, Bo; LEE, Lillian; VAITHYANATHAN, Shivakumar. Thumbs up?: sentiment classification using machine learning techniques. In: Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10. Association for Computational Linguistics, 2002. p. 79-86.

PECHENIZKIY, Mykola et al. Online mass flow prediction in CFB boilers with explicit detection of sudden concept drift. ACM SIGKDD Explorations Newsletter, v. 11, n. 2, p. 109-116, 2010.

PEDREGOSA, Fabian et al. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, v. 12, n. Oct, p. 2825-2830, 2011.

PENNINGTON, Jeffrey; SOCHER, Richard; MANNING, Christopher D. Glove: Global vectors for word representation. In: EMNLP. 2014. p. 1532-1543.

POH, Norman et al. Challenges and research directions for adaptive biometric recognition systems. Advances in Biometrics, p. 753-764, 2009.

POLLARD, Carol; CATER-STEEL, Aileen. Justifications, strategies, and critical success factors in successful ITIL implementations in US and Australian companies: an exploratory study. Information systems management, v. 26, n. 2, p. 164-175, 2009.

RENNIE, Jason D. et al. Tackling the poor assumptions of naive bayes text classifiers. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03). 2003. p. 616-623.

RESNICK, Paul; VARIAN, Hal R. Recommender systems. Communications of the ACM, v. 40, n. 3, p. 56-58, 1997.

REHUREK, Radim; SOJKA, Petr. Software framework for topic modelling with large corpora. In: In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. 2010.

ROELLEKE, Thomas. Information retrieval models: foundations and relationships. Synthesis Lectures on Information Concepts, Retrieval, and Services, v. 5, n. 3, p. 1-163, 2013.

ROSENTHAL, Sara et al. SemEval-2015 Task 10: Sentiment Analysis in Twitter. In: SemEval@ NAACL-HLT. 2015. p. 451-463.

SALAH, Saeed et al. A model for incident tickets correlation in network management. Journal of Network and Systems Management, v. 24, n. 1, p. 57-91, 2016.

SCOTT, Sam; MATWIN, Stan. Feature engineering for text classification. In: ICML. 1999. p. 379-388.

SHARMA, Anuj; DEY, Shubhamoy. Performance investigation of feature selection methods and sentiment lexicons for sentiment analysis. IJCA Special Issue on Advanced Computing and Communication Technologies for HPC Applications, v. 3, p. 15-20, 2012.

SHEU, Jyh-Jian et al. An efficient incremental learning mechanism for tracking concept drift in spam filtering. PloS one, v. 12, n. 2, p. e0171518, 2017.

SMITH, Aaron; ANDERSON, Monica. Online Shopping and E-Commerce, 2016. Available online in: <http://www.pewinternet.org/2016/12/19/online-shopping-and-e-commerce/>

SANDVINE. Global Internet Phenomena, 2016.

STEINBACH, Michael et al. A comparison of document clustering techniques. In: KDD workshop on text mining. 2000. p. 525-526.

SUNG, Tae Kyung; CHANG, Namsik; LEE, Gunhee. Dynamics of modeling in data mining: interpretive approach to bankruptcy prediction. Journal of Management Information Systems, v. 16, n. 1, p. 63-85, 1999.

TANG, Jiliang; ALELYANI, Salem; LIU, Huan. Feature selection for classification: A review. Data Classification: Algorithms and Applications, p. 37, 2014.

TOMEK, Ivan. Two modifications of CNN. IEEE Trans. Systems, Man and Cybernetics, v. 6, p. 769-772, 1976.

TSYMBAL, Alexey et al. Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. In: Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on. IEEE, 2006. p. 679-684.

TSYMBAL, Alexey. The problem of concept drift: definitions and related work. Computer Science Department, Trinity College Dublin, v. 106, n. 2, 2004.

UNITED NATIONS. E-Government Survey, 2014.

VAN SELM, Leo. ISO/CEI 20000-Introduction. Van Haren, 2009.

WANG, Suge et al. A feature selection method based on improved fisher's discriminant ratio for text sentiment classification. Expert Systems with Applications, v. 38, n. 7, p. 8696-8702, 2011.

WANG, Shuo; YAO, Xin. Multiclass imbalance problems: Analysis and potential solutions. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), v. 42, n. 4, p. 1119-1130, 2012.

WIDMER, Gerhard; KUBAT, Miroslav. Learning in the presence of concept drift and hidden contexts. Machine learning, v. 23, n. 1, p. 69-101, 1996.

YANG, Yiming; PEDERSEN, Jan O. A comparative study on feature selection in text categorization. In: Icml. 1997. p. 412-420.

YAP, Bee Wah et al. An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In: Proceedings of the First International Conference on Advanced Data and Information Engineering (DaEng-2013). Springer, Singapore, 2014. p. 13-22.

ZHOU, Wubai et al. Recommending ticket resolution using feature adaptation. In: Network and Service Management (CNSM), 2015 11th International Conference on. IEEE, 2015. p. 15-21.

ZHOU, Z. The development of service economy: a general trend of the changing economy. Development Research Center of Shanghai, Shanghai, 2015.

ZHU, Ji et al. Multi-class adaboost. Statistics and its Interface, v. 2, n. 3, p. 349-360, 2009.

ZLIOBAITĖ, Indrė. Learning under concept drift: an overview. arXiv preprint arXiv:1010.4784, 2010.

Appendix A – Experiment VI (Details)

The tables in this appendix give more details regarding the results obtained in Experiment VI. More specifically, they complement the tables shown in that experiment by adding the number of correctly classified tickets to the rankings of the classification trees.

Table 57, Table 58, and Table 59 show the monthly totals. The bottom line of each table displays the number of tickets that were generated during a specific month.

Table 57 – Monthly Ranking Of Classification Trees I (Details)

	M1		**M2**		**M3**	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	WSW	1,780	K3	1,785	K2	1,502
2nd	K5	1,776	WSW	1,782	WSW	1,494
3rd	WCT	1,772	K2	1,765	K1	1,491
4th	WCA	1,767	WCA	1,759	KN	1,490
5th	UV	1,766	WCT	1,754	WCT	1,490
6th	WLW	1,764	UV	1,753	K6	1,489
7th	WM	1,761	WM	1,753	WM	1,482
8th	K1	1,757	K4	1,752	UV	1,478
9th	K2	1,750	WLW	1,750	K3	1,476
10th	K4	1,749	K5	1,748	WLW	1,472
11th	K3	1,744	K1	1,742	K5	1,466
12th	KN	1,715	KN	1,740	WCA	1,459
13th	K6	1,710	K6	1,731	K4	1,455
Monthly Total	**2,669**		**2,460**		**2,960**	

Table 58 – Monthly Ranking Of Classification Trees II (Details)

	M4		M5		M6	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	K2	3,133	WSW	2,725	WCA	3,198
2nd	K1	3,128	WCA	2,716	WCT	3,194
3rd	WM	3,091	WM	2,714	UV	3,191
4th	WCT	3,083	WCT	2,700	WLW	3,183
5th	WSW	3,077	K2	2,693	WSW	3,177
6th	WCA	3,071	UV	2,692	WM	3,174
7th	UV	3,051	WLW	2,683	K3	3,170
8th	K3	3,012	K1	2,682	K4	3,163
9th	WLW	3,009	K3	2,677	K5	3,149
10th	K4	3,008	K4	2,639	K6	3,147
11th	K5	2,958	K5	2,636	K2	3,136
12th	KN	2,947	K6	2,622	KN	3,131
13th	K6	2,936	KN	2,591	K1	3,057
Monthly Total	**3,936**		**3,336**		**3,847**	

Table 59 - Monthly Ranking Of Classification Trees III (Details)

	M7		M8		M9	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	K6	3,249	WM	3,267	WSW	2,947
2nd	WLW	3,246	WCA	3,262	WLW	2,946
3rd	WCA	3,242	UV	3,261	WM	2,942
4th	WCT	3,238	WCT	3,260	UV	2,940
5th	WM	3,234	WLW	3,254	K3	2,939
6th	WSW	3,231	WSW	3,253	WCT	2,939
7th	UV	3,227	K3	3,246	K6	2,933
8th	K4	3,216	K4	3,241	WCA	2,932
9th	K5	3,215	K5	3,236	K5	2,929
10th	K3	3,192	K6	3,236	K4	2,928
11th	K2	3,179	K2	3,218	KN	2,928
12th	KN	3,170	KN	3,189	K2	2,916
13th	K1	3,137	K1	3,181	K1	2,890
Monthly Total	**3,997**		**4,162**		**3,663**	

Table 60 and Table 61 show the detailed rankings for each of the 5 classifiers of Incident Routing during the 9-month period of the experiment. The bottom line of each table displays the number of tickets, during that timespan, that belonged to the classes covered by that classifier.

Table 60 – Ranking Of Classification Trees Per Classifier I (Details)

	L0 vs. Non L0		**L1 vs. L2**	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	K2	28,753	WSW	18,180
2nd	WCA	28,746	WCT	18,131
3rd	WM	28,740	WCA	18,124
4th	WSW	28,739	WM	18,118
5th	WCT	28,735	UV	18,061
6th	K1	28,701	K3	18,047
7th	UV	28,700	WLW	18,043
8th	WLW	28,687	K2	18,041
9th	K3	28,639	K1	18,007
10th	K4	28,603	K4	17,974
11th	K5	28,553	K5	17,948
12th	K6	28,492	K6	17,888
13th	KN	28,226	KN	17,613
Total	**31,030**		**20,979**	

Table 61 – Ranking Of Classification Trees Per Classifier II (Details)

	L0		L1		L2	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	WM	8,545	K3	1,850	WSW	13,100
2nd	UV	8,542	K2	1,839	WCT	13,084
3rd	WSW	8,540	K4	1,836	WM	13,043
4th	WCA	8,539	WM	1,830	WCA	13,042
5th	WLW	8,537	WCA	1,826	UV	13,004
6th	WCT	8,530	WSW	1,826	WLW	12,953
7th	K2	8,522	WLW	1,817	K2	12,931
8th	K1	8,521	WCT	1,816	K3	12,914
9th	K3	8,477	UV	1,813	K5	12,878
10th	K4	8,438	K5	1,812	K4	12,877
11th	K5	8,423	K6	1,805	K6	12,841
12th	K6	8,407	KN	1,725	K1	12,838
13th	KN	8,362	K1	1,706	KN	12,814
Total	**10,051**		**3,392**		**17,587**	

Table 62, Table 63, Table 64, Table 65, Table 66, Table 67, and Table 68 show the detailed rankings for each class during the 9-month period of the experiment. The bottom line of each table displays the number of tickets, during that timespan, that belonged to each class.

Table 62 – Ranking Of Classification Trees Per Class I (Details)

	L0		**Non L0**	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	WM	8,557	K2	20,225
2nd	UV	8,554	WCA	20,198
3rd	WSW	8,550	WCT	20,193
4th	WLW	8,550	WSW	20,189
5th	WCA	8,548	WM	20,183
6th	WCT	8,542	K1	20,170
7th	K1	8,531	K3	20,153
8th	K2	8,528	UV	20,146
9th	K3	8,486	K4	20,142
10th	K4	8,461	WLW	20,137
11th	K5	8,439	K5	20,114
12th	K6	8,423	K6	20,069
13th	KN	8,387	KN	19,839
Total	**10,051**		**20,979**	

Table 63 – Ranking Of Classification Trees Per Class II (Details)

	L1		L2	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	K4	2,007	WSW	16,212
2nd	K6	2,002	WCT	16,178
3rd	WLW	1,995	K1	16,162
4th	K5	1,989	WM	16,153
5th	WCA	1,986	WCA	16,138
6th	K3	1,979	UV	16,082
7th	UV	1,979	K2	16,078
8th	WSW	1,968	K3	16,068
9th	WM	1,965	WLW	16,048
10th	K2	1,963	K4	15,967
11th	WCT	1,953	K5	15,959
12th	KN	1,918	K6	15,886
13th	K1	1,845	KN	15,695
Total	**3,392**		**17,587**	

Table 64 – Ranking Of Classification Trees Per Class III (Details)

	Cloud		External Network		Incident Management	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	UV	167	K1	8,355	K6	29
2nd	WM	167	WSW	8,344	KN	29
3rd	K2	166	WM	8,343	WLW	21
4th	K4	166	WCA	8,339	K5	20
5th	WLW	166	UV	8,337	UV	19
6th	WSW	165	WCT	8,331	K3	18
7th	WCA	164	WLW	8,328	K4	18
8th	K3	163	K2	8,326	WCA	18
9th	WCT	163	K3	8,278	WM	17
10th	K5	161	K4	8,230	WCT	17
11th	K6	157	K5	8,214	K2	15
12th	K1	145	K6	8,190	WSW	15
13th	KN	131	KN	8,155	K1	7
Total	**311**		**9,489**		**146**	

Table 65 – Ranking Of Classification Trees Per Class IV (Details)

	Production Support		Backup		Control	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	KN	47	WSW	1,234	K2	61
2nd	K6	31	K2	1,231	WM	56
3rd	K5	28	UV	1,229	WCT	56
4th	K4	24	WM	1,228	WSW	56
5th	WLW	22	WCA	1,227	WCA	54
6th	UV	19	K3	1,225	UV	52
7th	WCT	19	WCT	1,216	WLW	52
8th	K3	18	K4	1,213	K1	51
9th	WM	18	WLW	1,212	K3	51
10th	WCA	17	KN	1,188	K4	44
11th	WSW	16	K5	1,185	K5	43
12th	K2	15	K6	1,181	K6	40
13th	K1	14	K1	1,166	KN	25
Total	**105**		**1,577**		**103**	

Table 66 – Ranking Of Classification Trees Per Class V (Details)

	Job		Monitoring		Operation	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	K5	41	K3	529	KN	31
2nd	K6	38	K4	524	WLW	28
3rd	WLW	38	K5	521	K6	25
4th	UV	37	K6	521	K5	22
5th	WCA	37	K2	499	K4	21
6th	WCT	36	WM	494	UV	20
7th	KN	35	WCA	493	K2	19
8th	WM	35	WCT	493	WSW	18
9th	K4	43	WLW	487	WM	17
10th	K3	33	WSW	486	WCA	15
11th	WSW	32	UV	475	WCT	15
12th	K2	29	K1	450	K3	12
13th	K1	28	KN	446	K1	11
Total	**168**		**1,427**		**117**	

Table 67 – Ranking Of Classification Trees Per Class VI (Details)

	Application		Database	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	K2	5,294	WM	2,428
2nd	WSW	5,243	WCT	2,422
3rd	KN	52,37	WSW	2,421
4th	WCT	5,233	UV	2,420
5th	WM	5,226	WCA	2,420
6th	K1	5,225	K5	2,419
7th	WCA	5,214	K3	2,415
8th	K3	5,182	WLW	2,411
9th	K6	5,168	K4	2,406
10th	K5	5,165	KN	2,406
11th	K4	5,163	K6	2,405
12th	UV	5,143	K2	2,392
13th	WLW	5,107	K1	2,381
Total	**6,284**		**3,542**	

Table 68 – Ranking Of Classification Trees Per Class VII (Details)

	Internal Network		Platform	
	Classification Tree	**Correctly Classified Tickets**	**Classification Tree**	**Correctly Classified Tickets**
1st	KN	835	WSW	4,670
2nd	K6	795	UV	4,669
3rd	K5	794	WCT	4,662
4th	WLW	791	WLW	4,644
5th	WCA	782	WCA	4,626
6th	WM	774	WM	4,615
7th	UV	772	K3	4,550
8th	K3	767	K4	4,543
9th	WCT	767	K1	4,526
10th	WSW	766	K5	4,500
11th	K4	765	K2	4,497
12th	K2	748	K6	4,473
13th	K1	706	KN	4,336
Total	**1,105**		**6,656**	