UNIVERSIDADE
BEIRA INTERIOR

Automatização de Processos por meio de Inteligência Artificial

Versão final após defesa

João Pedro Machado Lindeza

Relatório de Estágio para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Sebastião Augusto Rodrigues Figueiredo Pais

julho de 2024

Automatização de Processos por meio de Inteligência Artificial

Declaração de Integridade

Eu, João Pedro Machado Lindeza, que abaixo assino, estudante com o número de inscrição M11995 de Engenharia Informática da Faculdade de Informática, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridades da Universidade da Beira Interior**.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 29 / 07 / 2024

Dedicatória

Quero dedicar a finalização deste ciclo de estudos a, além de todas as pessoas que me acompanharam e me apoiaram ao longo desta jornada, a mim mesmo por ser capaz de recolher todas essas peças e moldá-las a mim, tendo então sido capaz de manter a minha resiliência perante as adversidades, manter a dedicação e o foco de o terminar com o maior sucesso possível. Nunca ninguém saberá o que cada pessoa passa nesta fase senão quem as conclui, e mesmo essas diferem entre si.

Agradecimentos

Agradecimentos são devidos a todas as pessoas que de alguma forma contribuíram para que eu prosseguisse este ciclo, aos os meus pais e à minha avó, que me formaram para que eu fosse capaz de encarar este desafio, à minha irmã, que me manteve esse espírito juvenil, de que tanto gosto, vivo, com o seu lado artístico sempre me motivou a ser como ela e a deixá-la orgulhosa. À minha namorada, Margarida, que, muitas vezes sem saber bem o que fazer ou como o fazer, me conseguiu aturar e responder aos meus devaneios e irritações, não me deixando desistir. Aos meus amigos, todos, mas com especial atenção ao Kaynã, que todos os dias me motivou mantendo-me a par do progresso real que eu estava a fazer e do trabalho que eu tive todos os dias para chegar aqui, o que foi extremamente importante para eu ser capaz de ver o progresso e não recuar nele. A um dos meus mestres de Jiu-Jitsu Brasileiro, Mestre Gabriel Figueiredo, que durante os treinos foi capaz de me elevar quando estava mais em baixo e de me chamar à atenção quando eu queria correr sem saber andar, ensinando-me a manter o controlo e a perceber que mesmo dentro do caos, é possível encontrar a calma. Um grande obrigado a todos.

Resumo

A gestão eficiente de tarefas a serem realizadas dentro de um projeto ou empresa é um dos pontos cruciais para garantir a prontidão operacional e a satisfação do cliente, principalmente em empresas de telecomunicações, como a Altice, uma vez que uma baixa desses serviços pode implicar muitas consequências negativas. Este projeto teve como objetivo desenvolver um módulo de **Inteligência Artificial (IA)** que automatizasse o processo de alocação de tarefas aos técnicos de forma eficiente e rápida.

Até ao momento, a Altice encaminha os diferentes tipos de problemas reportados às várias empresas parceiras, cujo trabalho é resolvê-los, por meio de *tickets* que os descrevem. O *backoffice* da empresa aloca, depois, manualmente, as tarefas à equipa e técnico do terreno que mais sentido faça, face ao cenário a solucionar. Uma das empresas é a Excell Communications, cliente da Code 495 Solutions, que proporciona o sistema necessário para a primeira entidade gerir o trabalho. Este processo pode tornar-se demorado e ineficiente, aumentando os tempos de resposta e resolução, levando à insatisfação dos clientes.

O Projeto visou, então, desenvolver um módulo da IA que utilize algoritmos de ***Machine Learning (ML)***, para analisar o tipo de problemas e sua localização, e alocar as tarefas aos técnicos de forma inteligente e eficiente, maximizando os recursos disponíveis, o que leva à redução dos tempos de resposta, tornando o fluxo de trabalho mais rápido e preciso.

Este documento apresenta os vários objetivos do projeto, conceitos necessários à sua compreensão, métodos e tecnologias utilizados para os atingir, cronograma de trabalho, tarefas executadas, explicação da implementação e testes, assim como a análise dos resultados, com final previsto em junho de 2024.

Palavras-chave

dados;análise;aprendizagem;automática;gestão;automação;machine;learning;inteligência; artificial;processo;eficiência;algoritmos;modelo;treino;automatização;python;pyspark

Abstract

The efficient management of tasks to be carried out within a project or company is one of the crucial points to guarantee operational readiness and customer satisfaction, especially in telecommunications companies such as Altice, since a downturn in these services can have many negative consequences. This project aimed to develop an Artificial Intelligence (AI) module that would automate the process of allocating tasks to technicians efficiently and quickly.

So far, Altice forwards the different types of problems reported to the various partner companies, whose job it is to resolve them, by means of tickets describing them. The company's back office then manually allocates the tasks to the team and technician in the field who makes the most sense, given the scenario to be solved. One of the companies is Excell Communications, a Code 495 Solutions client, which provides the system needed for the first entity to manage the work. This process can become time-consuming and inefficient, increasing response and resolution times and leading to customer dissatisfaction.

The project therefore aimed to develop an AI module that uses Machine Learning (ML) algorithms to analyze the type of problems and their location, and allocate tasks to technicians intelligently and efficiently, maximizing the available resources, which leads to a reduction in response times, making the workflow faster and more accurate. This document presents the various objectives of the project, the concepts needed to understand them, the methods and technologies used to achieve them, the work schedule, the tasks tasks performed, an explanation of the implementation and tests, as well as an analysis of the results, which are expected to end in June 2024.

Keywords

data;analysis;machine;learning;management;automation;machine;learning;artificial;intelligence;process;efficiency;algorithms;model;training;automation;python;pyspark

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos e Siglas

AUC	Area Under the ROC Curve
CSV	Comma-Separated Value
IA	Inteligência Artificial
KNN	K-Nearest Neighbors
ML	Machine Learning
PLN	Processamento de Linguagem Natural
RPA	Robotic Process Automation
UBI	Universidade da Beira Interior

Capítulo 1

Introdução

Neste capítulo é feita uma introdução à hipótese do trabalho a realizar, apresentando o seu contexto na secção 1.1 e qual a motivação para a escolha deste tema na secção seguinte, 1.2. É então exposto o problema abordado e quais os objetivos do projeto, que levarão à solução teórica do mesmo, na secção 1.3 e, por fim, na secção 1.4 é apresentada a estrutura deste documento.

1.1 Contexto

Este documento descreve o planeamento efetuado para o Estágio Curricular necessário para a conclusão do 2º Ciclo de Estudos em Engenharia Informática na Universidade da Beira Interior (UBI). O trabalho desenvolvido prevê o desenvolvimento de um modelo de **Machine Learning**, capaz de distribuir, de forma automática, os vários trabalhos a realizar pela empresa.

Atualmente, a empresa faz esta alocação **manualmente**, podendo este processo tornar-se demorado e ineficiente, o que leva a atrasos na resolução. Além disso, o processo manual possui uma margem de falha maior durante a análise do problema, sendo, por vezes, trabalhos alocados a técnicos menos capazes de as resolver.

O objetivo deste projeto é então implementar um algoritmo de aprendizagem automática que irá automatizar esse fluxo, analisando o tipo de tarefa, a localização, entre outros fatores que se achem relevantes, para então a atribuir ao colaborador mais adequado.

1.2 Motivação

Nos últimos anos, a área de ML tem apresentado um rápido crescimento [1], tanto a nível de capacidades, tanto de acessibilidade, consequências do avanço tecnológico e da disponibilidade crescente de bibliotecas e ferramentas *open-source*.

Esta evolução permite a empresas como a Code 495 Solutions, utilizar estas técnicas nos seus sistemas, de modo a criar um **fator diferencial e competitivo**, como no caso do projeto com a Excell Communications. Entre os algoritmos existentes encontram-se os de reconhecimento de padrões e de tomada de decisão, que apresentam mais-valias, sendo estes capazes de se adaptar ao longo do tempo, conforme mais dados vão estando disponíveis para treino, tornando-se a automatização das tarefas cada vez mais precisa e eficiente à medida que for utilizada.

1.3 Problema Abordado e Objetivos

Esta secção, dividida em duas partes, identifica de forma concisa o Problema Abordado, na subsecção 1.3.1, e os objetivos do trabalho a realizar, de modo a obter os resultados o mais aproximados ao esperado possível, na subsecção 1.3.2.

1.3.1 Problema Abordado

A Excell Communications utiliza um **processo manual** para distribuir as tarefas aos trabalhadores do terreno, sendo este modo moroso e ineficiente, levando, por vezes, a atrasos nas resoluções, e consequentemente, insatisfação dos clientes. O facto de a distribuição ser feita manualmente implica também falhas de julgamento, atribuindo-se tarefas a colaboradores menos capacitados para resolver aquele tipo de situação. Além disso, com o passar do tempo e o aumento de *tickets*, pode tornar-se complicado para um humano lidar com tanta carga de trabalho.
Este problema é importante de ser solucionado, uma vez que a eficiência da empresa afeta diretamente a satisfação do cliente, podendo levar à perda dos mesmos e, por sua vez, diminuição de receita, seja por atrasos ou demoras, seja por má qualidade da resolução. Pode também ser por vezes necessário retornar ao local para fazer uma segunda reparação, gastando-se recursos desnecessários.

1.3.2 Objetivos

O objetivo deste projeto é **automatizar a distribuição de tarefas dentro do sistema desenvolvido pela Code 495 Solutions**.
Assim, os objetivos específicos são:

1. Desenvolver um algoritmo de aprendizagem automática capaz de analisar o tipo de tarefa, localização, entre outros fatores, de modo a atribuí-la ao colaborador mais adequado;
2. Identificar os dados históricos necessários para treinar o algoritmo;
3. Avaliar a eficácia do algoritmo no sistema.
4. Planear a implementação do modelo no sistema de produção da empresa;

   Os objetivos serão revistos durante os testes, sendo testada a qualidade das previsões e o impacto no sistema.

1.4 Estrutura do Documento

Este documento encontra-se organizado da seguinte forma:

- Capítulo 1 - Introdução - Apresenta uma introdução ao projeto, dando contexto do mesmo, qual a motivação para o desenvolver, o problema abordado e os seus objetivos. Por fim, é dada a estrutura do documento;
- Capítulo 2 - Estado da Arte - Introduz os conceitos e terminologia relevantes para a pesquisa, sendo depois feita a análise do trabalho relacionado, exemplos existentes e ferramentas que demonstram utilidade ao projeto;
- Capítulo 3 - Caracterização da Organização - Apresentação da empresa na qual será realizado o Estágio em Engenharia Informática, reforçando o contexto do projeto. São também identificadas as oportunidades oferecidas por desenvolver um projeto em contexto empresarial;
- Capítulo 4 - Plano de Trabalho - Descreve as atividades previstas para o sucesso do projeto, o cronograma estimado, com o prazo de cada tarefa especificado e a análise de riscos identificados;
- Capítulo 5 - Recolha e Pré-Processamentos dos Dados - Identifica a fonte de dados e a forma como este foram trabalhados para estarem aptos a ser processados pelos algoritmos;
- Capítulo 6 - Implementação - Descreve todo o processo de seleção do melhor algoritmo para o projeto e a forma como este foi trabalhado no sentido de atingir os melhores resultados possíveis.;
- Capítulo 7 - Análise de Resultados - Após os testes, é feita uma análise dos resultados recolhidos, de forma a aprofundar a viabilidade do modelo e futura implementação;;
- Capítulo 8 - Conclusão - Por fim é feita uma análise de todo o projeto, discutindo os resultados finais, algumas considerações finais e possíveis melhoramentos para o futuro.

Capítulo 2

Estado da Arte

Com o avanço das tecnologias de IA e a sua aplicação em diversas áreas, tais como o setor financeiro, da saúde e da própria área tecnológica [2], a utilização destas ferramentas passou a ser uma nova área de pesquisa e desenvolvimento, que cresce de dia para dia.
Isto vem do facto de tarefas repetitivas poderem ser substituídas por processos automáticos, libertando assim os recursos humanos para outras funções. Tal já existe, sob o nome de Robotic Process Automation (RPA), que se define como a "técnica que resulta da execução automática de tarefas administrativas, científicas ou industriais" [3], no entanto, estas são pré-programadas e estáticas, executando as tarefas para que foram programados, sem capacidade de adaptação.
Com o aparecimento da IA, é possível juntar as capacidades das duas ferramentas e chegar a um processo simbiótico entre ambas as partes, em que, através de algoritmos de ML, se possa melhorar a precisão, eficácia, eficiência e capacidade de ajuste dos processos automatizados. Este seria o cenário ideal para o sistema, uma vez que se pretende reduzir custos e tempo de resposta por parte dos trabalhadores da Excell Communications, assim como aumentar a sua eficiência ao automatizar o processo de atribuição das tarefas aos técnicos do terreno através de Machine Learning, que analise e defina os problemas contidos nos *tickets* recebidos, alocando-os depois ao colaborador que mais faça sentido, com base na localização e análise do problema.

2.1 Conceitos Prévios e Terminologia

2.1.1 Robotic Process Automation

A Robotic Process Automation, ou RPA, é, como o próprio nome indica, uma automatização de processos por intermédio robótico, no entanto, isto não significa que se trate de um robô físico, mas sim um software que replique ações humanas num computador, por exemplo [4]. Em processos repetitivos ou de tomada de decisão, é uma ferramenta de mais-valia, uma vez que permite alocar recursos humanos a outras tarefas, deixando estas tarefas a cargo do RPA. Existem 3 tipos de RPA, o Modo Assistido (Attended Mode), Modo Não Assistido (Unattended Mode) e Híbrido, sendo este uma junção dos 2 referidos anteriormente.
Respetivamente, o primeiro tipo tem como objetivo ser usado de forma supervisionada, ou seja, o utilizador inicia o processo de forma manual, este executa as tarefas pré-programadas no computador, indo consultando o trabalhador acerca das fases do processo, tal como mostrar dados e esperar confirmação, até que todas sejam executadas e o processo termine.
Já o Modo Não Assistido corre fora do ambiente local, sendo a sua iniciação em *real-time* ou em intervalos de tempo definidos, sem necessitar de intervenção manual. Além disso, é capaz de executar várias tarefas em simultâneo.

Quanto ao tipo Híbrido, faz sentido utilizar-se para lidar com processos longos, dividindo estes em duas partes: tomada de decisão e processo automático. Tome-se o exemplo de um empréstimo num banco, o gestor indica ao sistema que é necessário recolher informações do cliente, preencher formulários de pedido de empréstimo, gerar resultados de avaliação de elegibilidade e, manualmente, o trabalhador deve então fazer a tomada de decisão.
O projeto em questão acaba por ser uma RPA, sendo o objetivo automatizar um processo, em específico, de alocação de *tickets* de trabalho, nomeadamente em Modo Não Assistido, já que se pretende que as tarefas sejam automaticamente alocadas, sem intervenção de pessoal de *backoffice*, sendo apenas necessária vigilância. Pode também considerar-se o Modo Híbrido, apresentando uma sugestão automática de que técnico escolher para resolver o problema, sendo depois confirmada manualmente por um colaborador do *backoffice*.

2.1.2 Machine Learning

Machine Learning, ou Aprendizagem Automática, em português, é um ramo da IA que utiliza algoritmos e modelos computacionais capazes de analisar dados e de identificar padrões nestes, para mais tarde serem utilizados para realizar previsões e lidar com problemas relacionados com dados [5]. Além disso, durante os processos, o modelo é aprimorado, apresentando resultados cada vez mais precisos e eficientes.
Estas ferramentas, aliadas ao ponto anterior, permitirão a análise automática de *tickets* e automatização completa do processo.
O paradigma que mais se aplica ao projeto a desenvolver é a Aprendizagem Supervisionada, uma vez que se possuem dados de histórico para o treino do algoritmo, sendo depois este treinado também ao longo da utilização. Além disso, os algoritmos de categorização aparentam ser o ideal, já que os problemas devem ser categorizados para um técnico.

2.1.2.1 Aprendizagem Supervisionada

Um dos paradigmas mais conhecidos é o de **Aprendizagem Supervisionada**, em que, a partir de um conjunto de dados, se divide uma parte para treino e outra para teste, o modelo é então treinado, identificando padrões nos dados e afinado conforme os resultados dos testes, sendo depois utilizado para fazer previsões.
Entre os algoritmos mais utilizados de Aprendizagem Supervisionada estão: **Regressão Linear**, muito utilizado ao trabalhar com valores numéricos contínuos, uma vez que utiliza uma reta num gráfico para realizar as previsões; **K-Nearest Neighbors (KNN)**, utilizado principalmente em casos de categorização de dados, classificando os dados em grupos, com base nas suas semelhanças, identificando depois a qual pertencem os dados em análise; e **Random Forest**, que gera árvores de decisão.

2.1.2.2 Aprendizagem Semi-Supervisionada

Entre os outros paradigmas encontram-se a **Aprendizagem Semi-Supervisionada**, em que o conjunto de treino contém dados rolutados e não rotulados, ou seja, com resultado definido ou não, sendo então identificados os padrões e rotulados os dados em que falta previsão.

2.1.2.3 Aprendizagem por Reforço

A **Aprendizagem por Reforço** utiliza algoritmos que recebem *feedback* acerca dos valores gerados, na forma de recompensas ou penalidades, procurando este maximizar a recompensa total do processo. O modelo aprende a chegar ao resultado mais aproximado ao pretendido através de tentativa e erro.

2.2 Trabalho Relacionado

2.2.1 UiPath

A UiPath fornece uma plataforma sólida para automatização de processos empresariais, sendo líder no mercado de RPA. Possui capacidades de Inteligência Artificial, estando estas integradas nas poderosas soluções que fornece, tais como: classificação de documentos, extração de dados de vários tipos de fontes, validação e verificação de dados, gravação de ações, entre muitas outras. É capaz de se integrar com um vasto leque de software, aplicações e sistemas, de modo a estar presente durante todo o *workflow*, exponenciando assim o desempenho dos processos. Tem uma interface intuitiva, de modo a ser simples e direto de utilizar, já que a sua capacidade de escalabilidade permite que a ferramenta seja implementada em pequenos negócios ou grandes empresas.

2.2.2 Automation Anywhere

A Automation Anywhere integra Inteligência Artificial com os seus sistemas RPA, possuindo componentes de Processamento de Linguagem Natural e leitura de dados não estruturados, sendo capaz de automatizar processos **end-to-end**. Oferece opções de ferramentas que incluem gravações de *macros*, reconhecimento de imagem, extração de dados, dentre outros, aliadas à implementação de IA, são capazes de tomar decisões mais complexas de forma mais precisa. Possui outros serviços, tais como Automatização de Processos de Negócio e Gestão de Workforce Digital, permitindo trabalhar processos de forma completa.

2.2.3 Blue Prism

A Blue Prism é outra força global no ramo da RPA, integrando IA nos seus serviços, permite aos utilizadores destacar a plataforma para lidar com tarefas mais complexas, oferecendo melhores resultados. Além desta capacidade, oferecem soluções relacionadas com Automatização de Processos de Negócio, Automatização Inteligente e Gestão de Workforce Digital, destancando-se pela sua escalabilidade e flexibilidade em relação a outras opções do mercado. Possui recursos de análise de processos, gestão de bots e mineração de dados, por exemplo, permitindo fazer implementações em vários tipos de negócio.

2.2.4 SparkML

A SparkML é uma biblioteca de Machine Learning que suporta um vasto número de algortimos para classificação, regressão, entre outros [6], o que leva a que seja uma das bibliotecas mais utilizadas por empresas como a Amazon e a Microsoft.

Além da variedade de algoritmos disponíveis, foi pensada para ser escalável sobre *datasets* extensos e de fácil utilização devido à documentação que possui, explicando de forma detalhada como utilizar as capacidades da biblioteca e como estas funcionam.

É também capaz de automatizar vários processos, desde a preparação dos dados até à avaliação dos resultados da fase de testagem, algo que é extremamente valioso para o projeto, criando a possibilidade de além de automatizar o processo de alocamento das tarefas, automatizar também o próprio processo de treino do modelo.

Suporta várias linguagens de programação populares, Python, Java, Scala e R, sendo a primeira a mais interessante a explorar, uma vez que o sistema utilizado pela Excell Communications está desenvolvido sobre PHP, simplificando o processo de comunicação do módulo devido à existência de bibliotecas como **requests**, que permite fazer pedidos HTTP.

2.2.5 TensorFlow

O Tensorflow é uma biblioteca *open-source* de Machine Learning desenvolvida pela Google [7]. Atualmente, é uma das mais utilizadas no campo de *Deep Learning*, tendo capacidades de destaque como Reconhecimento de Imagem, Processamento de Linguagem Natural (PLN), entre outras.

Possuí uma ampla variedade de algoritmos, relacionados a *Deep Learning* e a Redes Neurais. É capaz de processar grandes quantidades de dados, de forma simples devido à extensa documentação própria e apoio da comunidade.

Devido à sua versatilidade, pode vir a ser útil ao projeto, já que possui capacidades de escalabilidade, importantes para o objetivo pretendido, principalmente a longo prazo.

Possui suporte para um vasto leque de linguagens de programação, sendo as principais Python e C++, além de outras como Java, Javascript e Go, no entanto, assim como na opção anterior, Python revela-se ser a que mais faz sentido escolher, pela simplicidade e popularidade da mesma.

2.3 Conclusão

Este capítulo abordou o estado da arte da Inteligência Artificial e a Automatização Robótica de Processos, analisando ferramentas que podem ser relevantes para o projeto, nomeadamente os diferentes tipos de RPA, os vários algoritmos disponíveis em Machine Learning e aplicações práticas da junção dos dois conceitos, que servirão como exemplo.

Assim, a solução pensada para a alocação de tarefas trata-se de uma RPA, quer em Modo Não Assistido, quer em Modo Híbrido, automatizando por completo o processo, ou introduzindo o fator de confirmação por parte de um colaborador.

As capacidades de aprendizagem oferecidas pela Machine Learning apresentam-se úteis, nomeadamente a Aprendizagem Supervisionada, pois há dados históricos disponíveis, com a definição de problemas anteriores e o técnico que os resolveu, sendo possível treinar o modelo. Após análise, os algoritmos do tipo K-Nearest Neighbors (KNN) ou Random Forest aparentam ser os mais adequados para o problema, sendo capazes de categorizar os *tickets* com base no tipo de problema e localização, identificando então o técnico mais próximo desse conjunto.

Ambas as bibliotecas apresentam capacidades de suportar o projeto, no entanto, a **SparkML** possui melhor capacidade de escalabilidade e, uma vez que se prevê um crescimento grande e contínuo dos dados, terá melhor capacidade de lidar com estes no futuro, excluindo desde já problemas relacionados com a quantidade de informações a processar.

Os sistemas oferecidos pelas UiPath, Automation Anywhere e BluePrism apresentam exemplos práticos da junção de RPA com Inteligência Artificial, sendo exemplos práticos dos objetivos pretendidos, servindo como exemplo. São ferramentas extremamente capazes, no entanto, para as desenvolver e manter, é necessária mão de obra especializada, uma vez que, além da dificuldade de composição do sistema em si, é necessário considerar os fatores de risco associados à tomada de decisão automática, como resultados ou processos inválidos.

Pode então considerar-se que a combinação de RPA e ML compõe a solução ideal para automatizar o processo de alocamento de tarefas aos técnicos da Excell Communications, tendo sido apresentadas as ferramentas que oferecem recursos para a sua implementação.

Capítulo 3

Caracterização da organização

A Code 495 Solutions é uma empresa de Desenvolvimento de Software portuguesa, fundada em 2023, sob os valores de criatividade, experiência e dinâmica. Possui escritórios em vários pontos do país, Covilhã, Vila Real e Lisboa, e mais recentemente em Nova Iorque, nos Estados Unidos da América.
Oferece soluções na área digital, tais como Desenvolvimento de Software personalizado, Design UX/UI, serviços de Marketing Digital, entre outros, sendo capaz de servir empresas de diversos setores.
Possui vários sistemas para venda, total ou de utilização, assim como manutenção dos mesmos, sendo um deles em colaboração com a Excell Communications, no qual vai ser implementado o módulo desenvolvido ao longo deste projeto, beneficiando ambas as partes, uma vez que o sistema da Code 495 Solutions será mais capaz, eficiente e moderno, permitindo à Excell Communications alocar recursos humanos para outras tarefas.
A supervisão por parte da empresa foi feita pelos Engenheiros **Valter Vale** e **António Santos**, que acompanharam o desenvolvimento e planeamento do projeto.
Além disso, uma vez que durante o período de estágio estive também empregado na Code 495 Solutions, estive presente em atividades de desenvolvimento Full Stack, utilizando tecnologias baseadas em JavaScript, PHP e MySQL.

Assim, elaborar o projeto na Code 495 Solutions permitiu:

1. Aplicar os conhecimentos num projeto real, em contexto de trabalho;
2. Trabalhar com pessoas experientes e qualificadas, capazes de me acompanhar ao longo do projeto;
3. Contribuir com um desenvolvimento inovador;
4. Solidificar os conhecimentos profissionais:
5. Reforçar as capacidades dos sistemas da Code 495 Solutions.

Capítulo 4

Plano de Trabalho

4.1 Introdução

O Plano de Trabalho foi elaborado após definição do problema e objetivos na secção 1.3, realização de pesquisa e seleção das ferramentas adequadas no capítulo 2, com o objetivo de definir de forma concisa os diferentes passos a tomar de modo a chegar ao resultado pretendido, a automatização do processo de alocação de tarefas aos técnicos da Excell Communications. Desta forma, este capítulo descreve as diferentes fases do planeamento da seguinte forma:

4.2 Definição e Planeamento de Tarefas

De acordo com os objetivos definidos na secção 1.3 e com as informações recolhidas no capítulo 2, Estado da Arte, é possível elaborar as tarefas que serão necessárias percorrer para chegar ao objetivo do projeto: **A alocação automática de tarefas** aos técnicos da Excell Communications. Para tal, ter-se-á em consideração o **Modelo em Cascata**, uma vez que se pretende definir os requisitos o melhor possível antes da etapa de Desenvolvimento, já que não se prevêm grandes alterações destes ao longo do projeto, permitindo estruturar e planear o projeto detalhadamente antes das próximas etapas, o que tornará o trabalho a realizar mais claro.
Desta forma foram identificadas as seguintes tarefas a realizar:

- **Projeto de Estágio** (16 semanas)
  - **T1** - Identificação do problema e Motivação (2 semana);
  - **T2** - Pesquisa de Trabalho e Ferramentas relacionadas (5 semanas);
  - **T3** - Definição dos objetivos do Projeto (4 semanas);
  - **T4** - Elaboração do documento de Projeto de Estágio (5 semanas).
- **Estágio** (23 semanas)
  - **T5** - Definição do Dataset (1 semanas);
  - **T6** - Pré-processamento do Dataset (1 semana);
  - **T7** - Elaboração do relatório de Estágio (21 semanas);
  - **T8** - Testes aos Algoritmos (2 semanas);
  - **T9** - Desenvolvimento do módulo (3 semanas);
  - **T10** - Testes e validação do modelo (2 semanas);
  - **T11** - Implementação do módulo em ambiente de produção (2 semanas).

4.2.1 Projeto de Estágio em Engenharia Informática

No primeiro semestre, no âmbito da Unidade Curricular **Projeto de Dissertação ou de Estágio em Engenharia Informática**, pretendia-se que fosse feito o planeamento do projeto a desenvolver durante o Estágio, aplicando desta forma conhecimentos adquiridos acerca de Gestão de Projetos e Qualidade de Software, no sentido de planear corretamente as diferentes fases do projeto, desde a identificação de um problema real que necessita de uma melhoria que faça sentido, até ao modo como esta ideia ia ser colocada em prática.
Tendo tido uma duração de 16 semanas para esta fase, encontra-se em seguida uma breve descrição de cada parte que a compõe.

4.2.1.1 T1 - Identificação do problema e Motivação

Nesta fase pretendeu-se identificar de forma concisa o problema a resolver e de que forma a sua solução iria ter um impacto positivo na empresa. Foi depois então efetuada uma reunião com a Code 495 Solutions para discussão da ideia, tendo-a recebido de bom grado e ajudado depois a identificar melhor o processo atual, para definir o plano de trabalho.

4.2.1.2 T2 - Pesquisa de Trabalho e Ferramentas relacionadas

Após perceber de forma precisa o problema a resolver, foi possível realizar pesquisa sobre as tecnologias e ferramentas que poderiam suportar o projeto, analisando as vantagens e desvantagens de cada uma, assim como de serviços semelhantes, já existentes, que pudessem servir de exemplo para o desenvolvimento. Identificar os pontos mais relevantes permitiu também que se iniciassem as próximas fases em concorrência, sendo possível começar a elaborar os objetivos e o documento de Projeto de Estágio.

4.2.1.3 T3 - Definição dos objetivos do Projeto

Foi então feita uma reunião com os orientadores do estágio de ambas as partes, de modo a identificar objetivos específicos, para perceber as necessidades da empresa e de que forma estas se podem colmatar. Este alinhamento facilitou a organização do trabalho com o professor doutor Sebastião Pais, já que, ao saber a forma como os dados estavam estruturados e de que forma precisavam de ser trabalhados, poderia elaborar-se o plano de trabalho.

4.2.1.4 T4 - Elaboração do documento de Projeto de Estágio

Ao longo desta fase, foi escrito este documento, que serve para concluir o 2º Ciclo de estudos em Engenharia Informática, e descreve o planeamento de atividades e do projeto a desenvolver.

4.2.2 Estágio em Engenharia Informática

Ao fazer o planeamento das tarefas durante este semestre facilitou o início e decorrimento das atividades práticas, a seleção e preparação dos dados que foram utilizados durante o desenvolvimento, validação de resultados, medição de impacto e implementação em ambiente de produção. Era pretendido que fossem aplicados os conhecimentos práticos adquiridos ao longo deste ciclo de estudos, adaptando-os ao problema a solucionar e caminho delineado, assim como às regras de negócio da empresa. O período de estágio propriamente dito iniciou-se a 1 de novembro de 2023 e terminou a 1 de março de 2024, tendo sido necessário entregar o relatório final até 11 de junho. Estas datas foram a base do planeamento das tarefas descritas de seguida, prevendo-se uma duração de 23 semanas para todo o projeto, incluindo elaboração do relatório.

4.2.2.1 T5 - Definição do Dataset

A primeira fase foi de análise, são identificados os dados necessários ao projeto, construindo depois o *dataset* sobre o qual foi feito o desenvolvimento. Este alimentou o algoritmo de ML, sendo dividido em duas partes, 70% dos dados foram utilizados para a fase de treino do modelo e os restantes 30% para teste. Depois de identificados foram então limpos e pré-processados, normalizando-os da melhor forma possível.

4.2.2.2 T6 - Pré-processamento do Dataset

Para facilitar o manuseamento e processamento dos dados, primeiro fez-se uma conversão de todos os valores categóricos para valores numéricos únicos, através do método de Codificação Numérica simples, em que se atribui um número inteiro único aos valores distintos de cada coluna categórica, criando identificadores únicos. Foram também limpos todos os registos com valores nulos, tendo sido avaliada a possibilidade de trabalhar dois tipos semelhantes de *ticket*, os UTT e os DSTX, devido a uma quantidade mais reduzida do primeiro tipo.

4.2.2.3 T7 - Elaboração do relatório de Estágio

Sendo que existia conteúdo suficiente para o início da elaboração do relatório de Estágio, pôde iniciar-se, sendo elaborado em paralelo com as seguintes tarefas.
Começou a haver conteúdo a relatar a partir da primeira semana de janeiro, deixando um intervalo de tempo suficiente para a escrita. Algumas partes do desenvolvimento podem tornar-se mais demoradas que o esperado sendo importante adiantar este tipo de trabalhos ao longo de todo o processo, também para evitar perdas de informação com o desenrolar do projeto, daí o tempo previsto de 23 semanas para a tarefa. Foram deixadas 3 semanas de margem, para o caso de algum dos passos planeados tomar mais tempo que o previsto.

4.2.2.4 T8 - Testes aos Algoritmos

Após ter o *dataset* pronto para processamento, puderam iniciar-se os testes aos vários algoritmos, de forma a identificar qual oferecia resultados melhores inicialmente, para depois ser aprimorado. Para isso, urilizaram-se as seguintes métricas: ***Area Under the ROC Curve***, ***Accuracy***, ***Precision***, ***Recall*** e **F1 *Score***, todos possíveis de medir com métodos do **SparkML**. Com estes valores, foi possível avaliar o desempenho dos vários modelos, assim como identificar possíveis problemas com o conjunto de dados, através das últimas 3 métricas, que se focam em avaliar se o modelo tem capacidade de identificar corretamente os técnicos mais aptos para os trabalhos, se está a perder muitos casos verdadeiros positivos ou se há um desequilíbrio entre as classes.

4.2.2.5 T9 - Desenvolvimento do módulo

Depois de identificado o melhor algoritmo, o **Random Forest**, fizeram-se alterações aos seus parâmetros numa tentativa de melhorar o seu desempenho. O SparkML oferece uma série de campos passíveis de receber valores, para encontrar a combinação que retorne previsões de maior qualidade. Além disso, fez-se também *feature selection*, analisando a influência de cada campo na previsão, de modo a identificar possíveis origens de *noise* e processamento desnecessário. Para comparação de resultados, testaram-se dois métodos de validação dos resultados das previsões, o ***Cross Validator*** e o ***Train Validation Split***, que funcionam de forma diferente na hora de dividir o conjunto de dados de treino. O primeiro divide o *dataset* de treino em todas as iterações que executa, enquanto que o segundo apenas divide os dados de treino uma vez. Isto resulta num processamento mais intensivo, mas expectavelmente mais preciso, ou mais rápido, mas com resultados, à partida, piores, respetivamente.

4.2.2.6 T10 - Testes e Validação do modelo

Caso se obtivesse um modelo que satisfizesse as condições propostas, com métricas cuja maioria dos valores rondasse os 70%, exceto o AUC, que se aceitavam valores a rondar os 0.8, deviam fazer-se testes com dados recebidos após a recolha inicial, uma vez que o modelo nunca os viu, servindo como um bom exemplo de capacidade de generalização. Com valores rondem os 90%, por exemplo, poderia considerar-se um RPA Não Assitido. No entanto, como discutido mais à frente nos capítulos 7 e 8, tal não foi possível.

4.2.2.7 T11 - Implementação do módulo em ambiente de produção

Ao recolher os resultados, fez-se uma reunião com a Code 495 Solutions e a Excell Communications, para estruturar um plano para se fazer a implementação no sistema, sendo, numa fase inicial, implementado o modelo num ambiente de desenvolvimento e realizados testes com dados reais. Depois o mesmo deveria ser aplicado em ambiente de qualidade, que possui uma aproximação maior ao ambiente de produção, de modo a despoletar situações não previstas no ambiente anterior. Por fim, se todas as validações fossem bem sucedidas, podia então proceder-se à aplicação do novo fluxo no ambiente de produção.
Esta implementação seria discutida com as empresas, para ser aplicada pós-estágio, uma vez que, devido à alteração de algumas regras empresariais a decorrer de momento, era difícil prever as condições a ser trabalhadas, mas, o plano inicial seria: ao receber os dados no sistema, o ficheiro PHP que os processa faria uma chamada ao ficheiro python com estes. Por sua vez, este passa-los-ia ao modelo treinado, que devolveria a sua previsão do técnico mais apto, retornando-o depois de volta para o PHP, registando-o na base de dados, para futuras análises e para o mostrar como recomendação no *text-end* do sistema.

4.3 Calendarização

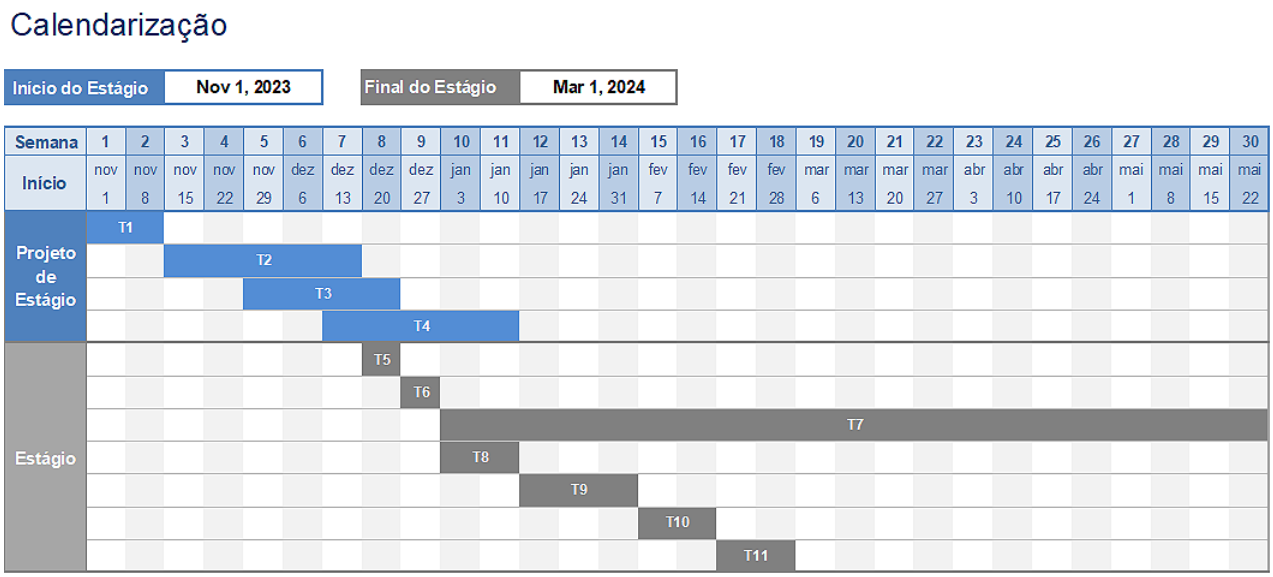


Figura 4.1: Calendarização do 2º Ciclo de Estudos

De maneira a tornar o encadeamento e período das tarefas mais explícito, foi elaborado um diagrama, mostrado na figura 4.1, no qual são descritas as atividades do estágio, do Projeto, do Estágio propriamente dito e da elaboração dos documentos necessários.
Nestes podem ver-se as várias tarefas a realizar ao longo do tempo, assim como as previsões de datas de início e término de cada uma, com o início do Projeto de Estágio em Engenharia Informática no dia 1 de novembro de 2024, terminando dia 28 de janeiro de 2024, data em que se teve de entregar um relatório com o planeamento.
Terminado o Projeto do Estágio, pôde iniciar-se o desenvolvimento, que começou no dia 20

de dezembro de 2023 e terminou no dia 25 de maio de 2024, considerando que todo o trabalho foi entregue a dia 11 de junho de 2024. É importante considerar que apesar do relatório só ser entregue em junho, o **período de estágio** acordado com a Code 495 Solutions terminou no dia 1 de março de 2024, pelo que todo o desenvolvimento teve de ser feito até essa data, utilizando o restante para a elaboração do relatório de Estágio, que necessitou de consolidar e analisar toda a informação recolhida.

Algumas tarefas, como é o caso das T2 e T3, respetivamente, a pesquisa relacionada e a definição de objetivos, precisaram de ser um pouco trabalhadas antes de iniciar a escrita do relatório de Projeto, no entanto, foram realizadas paralelamente. Já na segunda fase, durante o período de Estágio, as tarefas dependeram sempre da anterior, à exceção da escrita do documento, tendo estas de ser terminadas completamente para seguir para a próxima tarefa, de modo a evitar problemas, além de simplificar e organizar o projeto, sendo preciso fazer a identificação e tratamento dos dados antes do desenvolvimento e só quando este estava terminado se efetuaram os devidos testes e validações.

4.4 Análise de Risco

De modo a solidificar ainda mais a probabilidade de sucesso do projeto, foi feita uma análise de risco que prevê os potenciais problemas a ser enfrentados, qual a sua chance de ocorrer, como lidar com cada um e qual a sua origem. Isto serve para se poder mitigar situações indesejadas, antes de se tornarem um problema real, principalmente depois da implementação do projeto em ambiente de produção, já que uma falha neste processo pode afetar a resolução de problemas por parte dos técnicos, e, consequentemente, levar à insatisfação do cliente.

Foram então identificados alguns riscos possíveis, descritos na tabela 4.1.

Risco	Prob.	Mitigação	Origem
R1 - Identificação incorreta do Problema	Média	Comunicação correta com a Code-495 Solutions e a Excell Communications é crucial na definição do problema e do processo atual.	Análise do Problema
R2 - Definição Incorreta dos Objetivos	Baixa	Devem definir-se objetivos específicos e mensuráveis, sendo este documentados de forma a poder acompanhar o progresso do projeto.	Identificação de Objetivos
R3 - Pesquisa do Estado da Arte e ferramentas relacionadas insuficiente ou incorreta	Média	Devem consultar-se fontes confiáveis e completas, consultando a informação em mais que uma origem, de modo a confirmá-la, analisando-a cuidadosamente.	Pesquisa do Estado da Arte
R4 - Falta de coesão entre o Projeto de Estágio e o Estágio	Baixa	O planeamento do projeto deve ser feito tendo em consideração o tempo e necessidades do Estágio, definido corretamente os objetivos e atividades a desenvolver, assim como efetuar uma pesquisa de qualidade, mantendo comunicação com os orientadores de forma a mitigar uma desconexão ou ideia irrealista do plano de trabalho em relação às atividades práticas.	Planeamento do Projeto de Estágio
R5 - Definição Incorreta do Dataset	Alta	Deve definir-se corretamente o objetivo do projeto e os dados necessários para o atingir, mantendo-se a comunicação com a Code-495 Solutions e a Excell Communications, fazendo uma correta seleção e processamento dos dados para posterior trabalho.	Análise de Dados
R6 - Quantidade de Dados Insuficiente	Alta	Uma vez que o serviço de *ticketing* oferecido pelo sistema da Code-495 Solutions é recente, podem não haver dados suficientes para o treino de um modelo aceitável. É necessário realizar testes e analisar corretamente as métricas e dados recolhidos, de modo a perceber se já existem *tickets* de exemplo suficientes ou se é necessário juntar mais alguns trabalhos para o treino ser possível.	Testes aos algoritmos

Risco	Prob.	Mitigação	Origem
R7 - Desempenho Insatisfatório do Módulo	Média	A biblioteca e algoritmo escolhidos devem ser adequados ao problema a resolver e os dados a utilizar devem ser de qualidade para a fase de treino ser o mais eficiente possível, desenvolvendo um modelo consistente e útil.	Eficiência e precisão do módulo insatisfatórios
R8 - Dificuldade de Integração do Módulo no Sistema	Média	Uma vez que se esperam mudanças estruturais em ambas as empresas e sistemas, a implementação é difícil de se planear e prever possíveis erros, pelo que pode originar um problema, se houver mudanças grandes. No entanto, a linguagem de programação do sistema deverá manter-se o PHP e o modelo desenvolvido em Python, o que permite abordar alguns cenários, como a comunicação através de *requests*, a analisar após validação do modelo.	Integração do Módulo no Sistema da empresa
R9 - Testes falhados	Alta	A presença de bugs não detetados ou incompatibilidade com o sistema da Code-495 Solutions pode gerar problemas imprevistos no sistema de produção, sendo necessário realizar testes rigorosos e específicos, para mitigar o máximo de situações indesejadas possível	Testes e Validação
R10 - Validação Incorreta do Módulo	Média	Os critérios de validação devem ser claros e específicos, fazendo análises aos resultados dos testes, para identificar e corrigir os pontos não concordantes antes de implementação em ambiente de produção	Validação do Módulo
R11 - Implementação incorreta do Módulo em Produção	Baixa	A forma de implementação deve ser bem definida e, após introdução do módulo no ambiente de produção, deve fazer-se um acompanhamento do processo durante, pelo menos, uma semana, por garantia, caso hajam problemas não previstos	Implementação do Módulo

Tabela 4.1: Descrição dos riscos previstos durante o Projeto e Estágio

4.5 Conclusão

Após fazer uma análise mais aprofundada do Projeto que visa a alocação automática de tarefas aos técnicos, concluiu-se que se trata de uma ideia viável, existindo ferramentas capazes de auxiliar o seu desenvolvimento, tendo sido escolhida a SparkML especificamente, devido às suas capacidades e por ser uma biblioteca atual. A implementação deste módulo prevê um impacto positivo no negócio, otimizando o processo, o que resultaria numa redução de tempo gasto e ineficiência da alocação manual, alcançando maior satisfação do cliente e melhor alocação e gestão de recursos, quer humanos, quer materiais. O desenvolvimento apresentava alguns riscos associados, tendo estes sido explorados o mais detalhadamente possível, para aumentar a probabilidade de sucesso, o que permitiu realizar cada atividade prática com estes já em mente, além de permitir um planeamento mais preciso dos testes e validações. Destes riscos foram de facto enfrentados alguns deles, como a Quantidade de Dados Insuficiente e o Desempenho Insatisfatório do módulo, sendo isto discutido mais adiante no documento, nos capítulos 7 e 8. Espera-se que o módulo seja uma mais-valia para a Code 495 Solutions, já que oferece um ponto de destaque no seu sistema, aumentando a competitividade no mercado e o próprio sucesso do cliente, a Excell Communications. Pessoalmente, foi uma oportunidade de aplicar conhecimentos adquiridos ao longo do período de estudos, sentindo confiança e utilidade por desenvolver um projeto moderno e com um grande potencial.

Capítulo 5

Recolha e Pré-Processamento dos Dados

5.1 Introdução

Para treinar os vários modelos, foi necessário recolher dados que definissem de forma concreta e o mais precisamente possível os vários problemas, assim como a identificação do técnico que os resolveu, sendo esse o objetivo que os algoritmos pretendiam atingir.
Foi feita uma análise aos dados disponíveis, e, após exportação para um ficheiro **csv**, estes foram limpos e codificados de modo a reduzir ao máximo a informação desnecessária, para reduzir os custos computacionais e poder realizar o maior número de testes possível, o mais rápido possível, para tornar o processo mais eficiente e reduzir o impacto no sistema.
Este capítulo descreve estes processos, assim como uma análise final do conjunto de dados após o processamento, que serviu para o treino dos modelos.

5.2 Recolha de Dados

Os dados foram recolhidos do sistema da Code 495 Solutions, vindos do registo histórico dos *tickets* recebidos até à data presente. Destes puderam retirar-se as informações que definem os problemas e os técnicos que os resolveram, para alimentar os vários algoritmos de Machine Learning, sendo importante rever os dados possíveis de trabalhar, uma vez que, devido às políticas de privacidade, os dados dos clientes da Altice devem manter-se privados.
Os dados foram então exportados para um documento em formato **"csv"**, de modo a ser utilizado pelos vários métodos da biblioteca PySpark, sendo contabilizados 22467 *tickets* no final.

5.2.1 Caracterização e Seleção dos Dados

Após uma análise dos dados disponíveis para seleção, foram filtrados os campos utilizados pelos trabalhadores do *backoffice* para chegar ao técnico que julgam ser o mais adequado para resolver determinada tarefa, chegando-se aos campos que definem o local, sendo esses **o estado, a cidade e o identificador do nó da rede**, e o problema, com base em **três códigos, o do tópico do problema, o de falha e o de detalhe**, sendo o primeiro um identificador geral e o segundo e terceiro, como os próprios nomes indicam, detalhamentos do ocorrido. Foram também recolhidos os identificadores únicos dos trabalhadores que resolveram os ocorridos, supondo que, mesmo que ao longo do tempo de vida do *ticket* este tenha sido alocado a diferentes técnicos, assume-se que o final terá sido o técnico mais apto para o serviço.
Desta forma, os trabalhadores do *backoffice* conseguem identificar o problema, havendo uma filtragem dos possíveis técnicos, já que trabalhos relacionados com cabos elétricos ou de rede serão entregues a pessoal especializado nessa área, enquanto algo relacionado com falhas de

rede, que precisem de testes específicos, necessitarão de outro tipo de mão de obra, por exemplo. Depois, uma vez que a Excell Communications possui várias equipas espalhadas pelas suas áreas de atuação dentro dos Estados Unidos da América, dependendo do estado e da cidade, é escolhido o técnico que mais faça sentido com base na distância.
Foi também feita **feature selection** aos dados, para identificar quais são mais relevantes para os modelos, de modo a simplificar o dataset e reduzir ao máximo o *noise* causado pela presença de *features* pouco impactantes na escolha do técnico.

5.3 Pré-Processamento dos Dados

Depois de recolhidas as informações disponíveis, que mais faziam sentido para identificar o problema, com base nos dados a que os trabalhadores do *backoffice* têm acesso à chegada dos *tickets*, recolheram-se 21388 registos. Foi então necessário fazer um pré-processamento dos dados, para deixar o dataset o mais limpo possível, isto é, sem linhas irrelevantes ou cujos dados são fora do normal, como *tickets* de teste.
Além disso, existem dois tipos de *ticket*, UTT e DSTX, destinados estes a diferentes tipos de problemas, sendo também preciso avaliar a disponibilidade de dados suficientes após o processamento de ambos os tipos para a fase de treino, uma vez que existiam apenas 3018 registos do primeiro tipo, sendo os restantes 19444 do segundo.

5.3.1 Limpeza dos Dados

Para começar a limpeza dos dados começou-se por excluir *tickets* que foram utilizados durante testes, uma vez que estes possuíam alguns valores anormais ao processo, assim como alguns que foram transmitidos à Excell Communications com erros.
Por fim, foram retirados os registos com valores nulos, já que, devido à dificuldade de prever que valores seriam possíveis de preencher esses campos, era impossível aplicar técnicas de *data repair*.
Após estas etapas de pré-processamento dos dados, sobraram 18604 conjuntos de valores válidos do tipo DSTX e 675 do tipo UTT, pelo que não existiam dados suficientes do segundo, sendo esses descartados do processo.

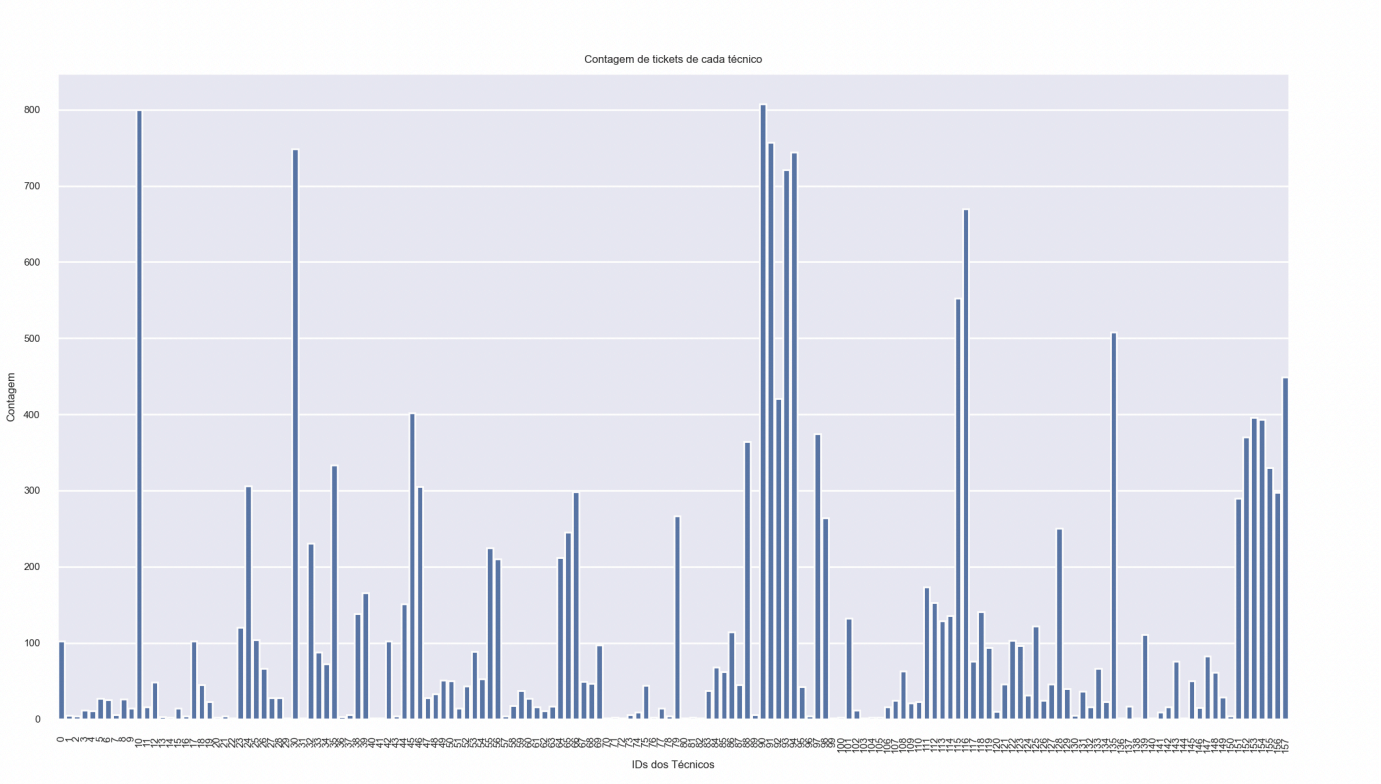
5.3.2 Codificação e Normalização das Variáveis Categóricas

De modo a alimentar os algoritmos, foi também necessário converter os valores categóricos em discretos, tais como o estado, a cidade, o identificador do nó da rede, e os códigos de falha e detalhe, tendo-se identificado os valores únicos de cada um e associado um valor inteiro, começando no 1, até haver referências para todos.
Apesar da coluna dos identificadores dos técnicos já conter valores discretos, estes foram também remapeados, uma vez que os seus valores estavam compreendidos no intervalo de 20 a 1003, inclusive, e, após serem todos referenciados, chegou-se a apenas 157 técnicos diferentes, o que permitiu trabalhar com números inferiores e depois fazer o remapeamento para o número identificador do técnico.

5.4 Visualização dos Dados

De modo a tornar a análise mais visual, fizeram-se gráficos da distribuição do número de *tickets* de cada técnico, para analisar a existência de classes minoritárias que possam estar muito pouco representadas. Desta forma obteve-se o gráfico da imagem 5.1.

Figura 5.1: Distribuição do número de tickets por cada técnico



Desta forma, o desequilíbrio das classes tornou-se mais evidente, o que pode vir do facto de certos problemas serem mais raros que outros, ou de haver técnicos novos que possuam poucos *tickets* à data atual. Previa-se que isto afetasse as previsões devido a haver muitos técnicos com poucos dados, o que levaria a que o modelo não seja capaz de analisar padrões para as previsões ou que não fizesse uma previsão correta quando em situações de casos pouco representados.

5.5 Conclusão

Com os dados recolhidos foi possível identificar um potencial problema no conjunto de dados, que é a falta de exemplos de *tickets* de cada técnico. Um *dataset* desequilibrado a nível de classes, como é o caso, pode originar problemas de generalização ou de métricas com valores enganosos, levando a confiar mais no modelo do que ele é realmente capaz de oferecer, o que, em cenário real levará a uma proporção de previsões de técnicos não aptos demasiado elevada.

Desta forma, é importante ter esta estrutura do conjunto de dados em mente na hora de avaliar as métricas recolhidas, assim como se foi também interessante explorar técnicas de *oversampling* de modo a reduzir a discrepância entre o número de dados de cada técnico e realizar testes sobre esses dados para comparar com os originais.

Em teoria, os campos recolhidos devem permitir a identificação dos técnicos, uma vez que são os mesmos a que os trabalhadores do *backoffice* têm acesso quando o *ticket* entra no sistema para atribuir a um técnico.

Capítulo 6

Implementação

6.1 Introdução

Havendo vários algoritmos possíveis de utilizar, foi necessário realizar testes para identificar o que melhores resultados apresenta, para depois ser trabalhado. De modo a tornar esses testes o mais eficientes possível, decidiu montar-se uma Pipeline que processa o dataset e testa os vários modelos na mesma execução, comparando as várias métricas, *accuracy*, *Area Under the ROC Curve*, ou AUC, Precisão, *Recall* e F1 utlizadas para avaliar o desempenho dos vários modelos treinados.
Ao ter o modelo escolhido, foi então avaliada a importância das *features*, de modo a retirar os campos desnecessários e reduzir o *noise* e tornar os modelos mais simples.
Por fim, fizeram-se testes com vários conjuntos de valores de parâmetros do algoritmo, chamado "*Hyperparameter Tuning*" para fazer os últimos ajustes e tentar aumentar a percentagem de acerto dos resultados finais.

6.2 Seleção do Modelo

Para montar a pipeline, foi preciso primeiro analisar que tipo de modelos fazia mais sentido testar, de acordo com o tipo de dados do *dataset* e do objetivo do modelo, o que, de acordo com o planeamento efetuado na Unidade Curricular anterior, se resume aos seguintes algoritmos: Regressão Logística, *Random Forest*, *Decision Tree* e Naive Bayes.
Ao ter definidos os modelos, pôde montar-se um ciclo contínuo de testes e treinos, que a partir do mesmo sub-conjunto de dados, que é retirado do *dataset* original a cada ciclo, pudessem testar-se os vários algoritmos e analisar os seus resultados de forma mais precisa.
Já tendo o tipo de dados de todos os campos, foi possível montar o esquema de dados, de forma a evitar problemas de dualidade de dados, ou destes não estarem corretamente definidos no ficheiro. O PySpark possui uma classe que define a estrutura, a **StructType()** [8], a que foi passado um conjunto de objetos do tipo **StructField()** [9], um por cada campo do ficheiro, recebendo estes o nome da coluna a definir, uma função que define o tipo, e o último valor **True** serve para permitir a existência de valores nulos, para evitar erros durante os testes, uma vez que o *dataset* já tinha sido pré-processado e retiradas todas as linhas com valores nulos.

```
schema = StructType([
    StructField("tk_thread_id", StringType(), True),
    StructField("tk_subject", StringType(), True),
    StructField("tk_topic_id", IntegerType(), True),
    StructField("tk_details_id", IntegerType(), True),
    StructField("tk_topic_sla", IntegerType(), True),
    StructField("tk_state_id", IntegerType(), True),
    StructField("tk_city_id", IntegerType(), True),
    StructField("tk_node_id", IntegerType(), True),
    StructField("tk_failure_code_id", IntegerType(), True),
    StructField("tk_user_id_on", IntegerType(), True),
    StructField("tk_user_id_on_min", FloatType(), True)
])
```

Após ter a estrutura e os dados carregados num *dataframe*, utilizaram-se vários métodos de manipulação de *features* do PySpark para preparar o dataset de forma a ser consumido pelos vários modelos da pipeline. O primeiro método aplicado foi o **StringIndexer** [10], que adiciona uma coluna ao *dataframe* com os *labels* indexados. É passado outro método à pipeline que irá então fazer a conversão inversa, o **IndexToString** [11], de modo a ser mais fácil analisar os dados no conjunto de dados de previsão.
Depois, fez-se o ajuntamento das *features* numa única coluna, com o método **VectorAssembler** [12], que agrega todos os valores das várias colunas numa única, com o vetor resultante.
Com os dados prontos para o treino, dividiu-se aleatoriamente os dados em dois grupos, de treino, com 70% dos valores, e de teste, com os restantes 30%. Esta divisão foi feita cada vez que um ciclo de treino dos modelos era concluído, tendo sido executados 500 ciclos com os valores *default* dos métodos disponibilizados pelo PySpark.
Em cada ciclo era verificado se já existia um modelo treinado anteriormente, sendo este avaliado, para depois se poder comparar com as métricas dos novos modelos. É montada então a pipeline, da seguinte forma:

```
pipeline = Pipeline(stages=[labelIndexer, featuresAssembler, model,
labelConverter])
newTrainedModel = pipeline.fit(trainingData)
newModelPredictions = newTrainedModel.transform(testData)
```

Depois de treinados e testados os novos modelos, as métricas são comparadas com os modelos anteriores, sendo depois guardados, caso apresentem valores melhores.

6.2.1 Testes Aos Modelos

Para testar o desempenho dos vários modelos, recolheram-se métricas que permitem avaliar algoritmos de classificação, ao longo dos 500 ciclos: *accuracy*, *Area Under the ROC Curve*, F1, *Weighted Precision* e *Weighted Recall*, com base nelas, foi possível comparar de forma mais precisa a diferença entre os resultados dos vários algoritmos, o que permitiu escolher o mais apto, face ao problema a resolver.
A bilbioteca PySpark possui vários métodos de avaliação dos modelos, tendo sido escolhidos os mais aptos para testar algoritmos de classificação, sendo esses o **BinaryClassificationEvaluator** [13] e **MulticlassClassificationEvaluator** [14], sendo o primeiro utilizado para analisar os resultados dum ponto de vista binário, ou seja, se as previsões foram ou não corretas, enquanto que o segundo oferece várias métricas recolhidas num cenário onde há mais de duas classes possíveis, o que é o casso, já que existem vários técnicos possíveis para resolver determinado problema, mas pretende-se escolher o melhor.
O avaliador binário recebe também o tipo de métrica a avaliar, tendo sido utilizado o ***areaUnderRoc***.
Estes foram então inicializados da seguinte forma:

```
multiEvaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel",
predictionCol="prediction")

accEvaluator = BinaryClassificationEvaluator(labelCol="indexedLabel",
rawPredictionCol="prediction", metricName="areaUnderROC")
```

Estes foram usados para, mais à frente, testar o **AUC**, por parte do avaliador binário e as restantes métricas, a ***Accuracy***, a **Precisão**, o **Recall** e o **F1**, por parte do avaliador múltiplo:

```
newModelACC = multiEvaluator.evaluate(modelPredictions,
{multiEvaluator.metricName: "accuracy"})

newModelAUC = accEvaluator.evaluate(modelPredictions)

newModelF1 = multiEvaluator.evaluate(modelPredictions,
{multiEvaluator.metricName: "f1"})

newModelWPREC = multiEvaluator.evaluate(modelPredictions,
{multiEvaluator.metricName: "weightedPrecision"})

newModelWREC = multiEvaluator.evaluate(modelPredictions,
{multiEvaluator.metricName: "weightedRecall"})
```

6.2.2 Accuracy

Começando pela *accuracy*, esta mede a proporção de previsões corretas num valor enquadrado entre 0 e 1, sendo que quanto maior, mais previsões foram bem feitas. É uma métrica simples e direta, no entanto, é necessário cruzá-la com as restantes métricas, para garantir que não retorna valores melhores que os reais, devido a, por exemplo, existir um técnico que tenha resolvido a maioria dos tickets, se o modelo classificar todos os tickets com esse identificador, irá obter uma taxa de acerto relativamente elevada.
Assim, reviu-se o dataset de forma a fazer a contagem de tickets de cada técnico, obtendo-se 795 tickets para a classe mais repetida e apenas 1 para várias outras. Além disso, apenas 49 técnicos possuíam mais de 100 tickets resolvidos, restando 108 com menos, de um total de 157, o que se traduz em apenas 31,2% dos dados se poder considerar em quantidade suficiente. Tendo isso em consideração, obtiveram-se os resultados descritos na tabela 6.1.

Accuracy		
Modelo	Média	Maior Valor
Regressão Logística	0.1157	0.1283
Random Forest	0.1420	0.1564
Decision Tree	0.1296	0.1408
Naive Bayes	0.0485	0.0570

Tabela 6.1: Resultados de Accuracy recolhidos durante a fase de testes

Pode ver-se que o modelo que obteve melhores resultados foram o *Random Forest* e o *Decision Tree*, o que é interessante do ponto de vista de funcionarem de forma semelhante, e o que obteve pior resultado foi o Naive Bayes, com um valor muito inferior aos outros algoritmos testados.

6.2.3 Area Under the ROC Curve

A *Area Under the ROC Curve*, ou AUC, mede a área do espaço existente por baixo da curva ROC (Receiver Operating Characteristic) na área compreendida entre os pontos (0,0) e (1,1). A linha em si é desenhada comparando as taxas de verdadeiros positivos e falsos positivos o que leva que o valor da AUC represente uma taxa de previsões corretas quanto mais aproximado de 1 for o seu valor.
Assim como na *accuracy*, este valor é afetado pela distribuição desigual de classes, neste caso, do número de tickets que existem no dataset para cada técnico, tendo sido revisto na subsecção anterior, **Accuracy**, que existe de facto uma quantidade desigual de tickets associados aos vários técnicos, sendo isso considerado na análise.
Após os vários ciclos de teste, obtiveram-se os resultados apresentados na tabela 6.2.

AUC		
Modelo	Média	Maior Valor
Regressão Logística	0.6759	0.7665
Random Forest	0.6989	0.8464
Decision Tree	0.6066	0.6926
Naive Bayes	0.3549	0.4077

Tabela 6.2: Resultados de AUC recolhidos durante a fase de testes

Os valores recolhidos foram bastante bons para uma primeira fase, tendo sido o algoritmo *Random Forest* a atingir os melhores valores, indicando que é capaz de atingir valores de 0.8464 sem *tuning*, o que, à partida, significaria que é capaz de distinguir verdadeiros positivos, os técnicos com as capacidades de resolver o problema, dos falsos positivos, os que seriam menos adequados, em cerca de 85% das vezes.
No entanto, é necessário analisar o valor de AUC em conjunto com outras métricas, pelas razões discutidas anteriormente, de distribuição de classes, e devido à discrepância entre estes resultados e os da *accuracy*, sendo estes relativamente elevados, com os melhores resultados compreendidos entre os 0.4077 e 0.8464, dos algoritmos Naive Bayes e *Random Forest*, respetivamente, e os anteriores muito baixos, entre 0.057 e 0.1564, para os mesmos métodos. Isto pode indicar que a diferença entre as quantidades existentes de tickets resolvidos por cada técnico esteja realmente a afetar os resultados. Podem também tratar-se de problemas de generalização, que irão ser despistados durante a fase de treino, ao utilizar diferentes métodos, de modo a que o modelo seja testado em dados não vistos anteriormente.

6.2.4 Precisão

A precisão é calculada com base no número de exemplos bem previstos, em relação ao número total de previsões positivas, ou seja, calcula a proporção de positivos verdadeiros de entre o total de positivos.
É importante analisar esta métrica pois, apesar de ter uma taxa de acerto alta, esta pode envolver muito falsos positivos, o que, na hora de efetuar previsões com o modelo, irá implicar o retorno de técnicos inválidos para os serviços.
Recolheram-se os resultados apresentados na tabela 6.3.

Precisão		
Modelo	Média	Maior Valor
Regressão Logística	0.0520	0.0876
Random Forest	0.0855	0.1173
Decision Tree	0.0957	0.1197
Naive Bayes	0.0443	0.0614

Tabela 6.3: Resultados de Precisão recolhidos durante a fase de testes

Com estes resultados, o desequilíbrio das classes torna-se mais evidente, uma vez que os modelos estão a ser capazes de distinguir as diferentes classes, devido aos valores de AUC elevados, no entanto, não consegue fazer previsões corretas, o que indica que em diferentes cenários não estão a ser capazes de identificar o técnico mais apto para as situações, o que leva a que as restantes métricas tenham valores muito baixos.
A precisão, em conjunto com a métrica analisada na subsecção seguinte, a **Recall**, dão-nos a possibilidade de calcular o **F1**, discutido também mais à frente, que indica melhor a proporção de acerto do modelo, sendo possível fazer uma análise mais completa.

6.2.5 Recall

A *recall*, ou taxa de deteção, mede a proporção que, de entre todos os valores que devia ter previsto corretamente, sejam verdadeiros positivos ou falsos negativos, quantos realmente são verdadeiros positivos.
Assim, serve para perceber quantos técnicos o modelo é capaz de prever, de entre todos os que devia ser possível este prever. É importante cruzar estes dados com os da secção **Precisão**, de modo a analisar o equilíbrio entre as previsões corretamente efetuadas e as analisadas por esta métrica. Tal será feito na secção seguinte, a secção **F1**.
Os dados recolhidos de *recall* estão descritos na tabela 6.4.

Recall		
Modelo	Média	Maior Valor
Regressão Logística	0.1157	0.1283
Random Forest	0.1420	0.1564
Decision Tree	0.1296	0.1408
Naive Bayes	0.0485	0.0570

Tabela 6.4: Resultados de Recall recolhidos durante a fase de testes

Os valores baixos de *recall*, em conjunto com as métricas anteriores, indicam também que um desequilíbrio nas classes, não sendo os modelos capazes de identificar os verdadeiros positivos do dataset de teste. Ainda será calculada também a F1, apesar dos valores de precisão e *recall* baixos, de modo a confirmar que estas estão equilibradas, no sentido de despistar a correta identificação de verdadeiros e falsos positivos.

6.2.6 F1

Esta métrica é importante para consolidar as métricas anteriores, uma vez que ambas são importantes, é necessário haver um equilíbrio entre elas, sendo esse o objetivo da métrica **F1**, medir o quão capaz o modelo é de distinguir as previsões positivas e negativas.
É uma métrica harmónica, o que equilibra as duas métricas anteriores, tendo em consideração os valores baixos, calculada da seguinte forma: **F1 = 2 * (Precisão * Recall) / (Precisão + Recall)**. Esta fórmula evita desequilíbrios que não seriam detetados se fosse feita uma média entre os dois valores anteriores, pois uma precisão alta e um *recall* baixo, uma situação indesejável, pois ambas as métricas devem ter valores altos para garantir que o modelo prevê o maior número de técnicos corretamente possível, iria gerar um F1 mediano, que indicaria um acerto e deteção de verdadeiros positivos razoável, o que não seria verdade.
Os métodos de avaliação do PySpark oferecem os valores desta métrica automaticamente, tendo sido recolhidos durante os testes e organizados na tabela 6.5.

F1		
Modelo	Média	Maior Valor
Regressão Logística	0.063	0.0748
Random Forest	0.0843	0.1026
Decision Tree	0.0807	0.0950
Naive Bayes	0.0325	0.0387

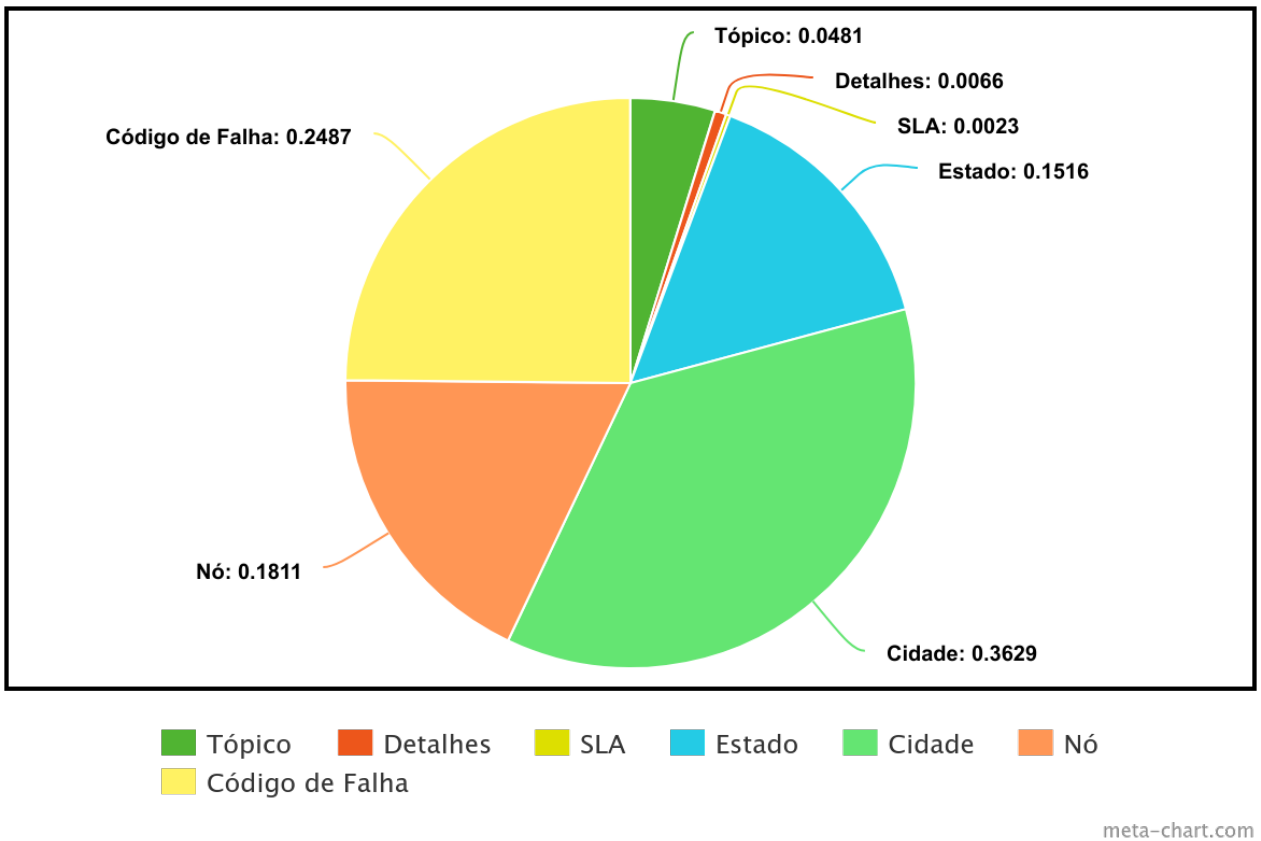
Tabela 6.5: Resultados de F1 recolhidos durante a fase de testes

Os valores de F1 confirmam que os modelos não estão a ser capazes de fazer previsões corretamente, provavelmente devido ao desequilíbrio das classes. Poderá ser necessário aplicar técnicas de *over* ou *undersampling*, de modo a tentar equilibrar a quantidade de tickets de cada técnico, numa tentativa de mitigar este problema.
Estes resultados fazem sentido no contexto das métricas anteriores, uma vez que os valores da precisão estão entre 0.04 e 0.09, aproximadamente, e os valores de *recall* entre os 0.04 e o 0.14, sendo ambos os valores mais baixos obtidos ao testar o algoritmo Naive Bayes e os mais altos do *Random Forest* e da Árvore de Decisão.

6.3 Feature Selection

Durante os ciclos da fase de testagem, foi também verificada a importância das *features*, para se perceber quais as mais impactantes e necessárias, de modo a retirar as outras além de reduzir o *noise* causado, reduzir também o tempo de execução, já que o dataset se torna mais simples.
Sendo apenas possível recolher valores dos algoritmos *Decision Tree* e *Random Forest*, obtiveram-se os resultados representados na imagem 6.1, emn média e arredondados aos décimos de milésimos:

Figura 6.1: Valores médios recolhidos durante a Feature Selection



Estes valores fazem sentido em situação real uma vez que a cidade e o código de falha são os que definem o local e o tipo de problema da forma mais específica, dentre todos os campos, sendo expectável que fossem os mais determinantes na hora de decidir o técnico mais apto para efetuar as tarefas no terreno.

Assim, tendo o tópico, os detalhes e o sla valores muito baixos de importância, foram removidos do dataset de futuros testes e treinos.

6.4 Algoritmo Random Forest

O algoritmo escolhido foi então o **Random Forest** pois, após os testes, os métodos que obtiveram melhores resultados de métricas foram este e o *Decision Tree*, e, uma vez que estes têm aplicações semelhantes ao outro método, mas apresenta mais robustez para lidar com dados mais complexos, além de possuir uma maior resistência a *overfitting*.

Os valores das métricas pareceram ser os mais promissores, ainda que não tenham sido propriamente altos, no cenário a trabalhar, será o algoritmo mais apto, pois o objetivo do modelo é substituir a tomada de decisões para chegar a um técnico.

6.5 Oversampling dos Dados

Decidiu-se também fazer testes com um *dataset* a que foi aplicados métodos de *oversample* dos dados, multiplicando as várias linhas existentes, até haver pelo menos 300 *tickets* de cada técnico. Não foi possível adicionar linhas com dados teoricamente possíveis, pois era difícil prever um conjunto de técnicos e problemas provável.

Assim, para manter a variedade de tipos de problemas de cada técnico o máximo possível, todos os *tickets* dos técnicos com menos que o número pretendido foram multiplicados da seguinte forma:

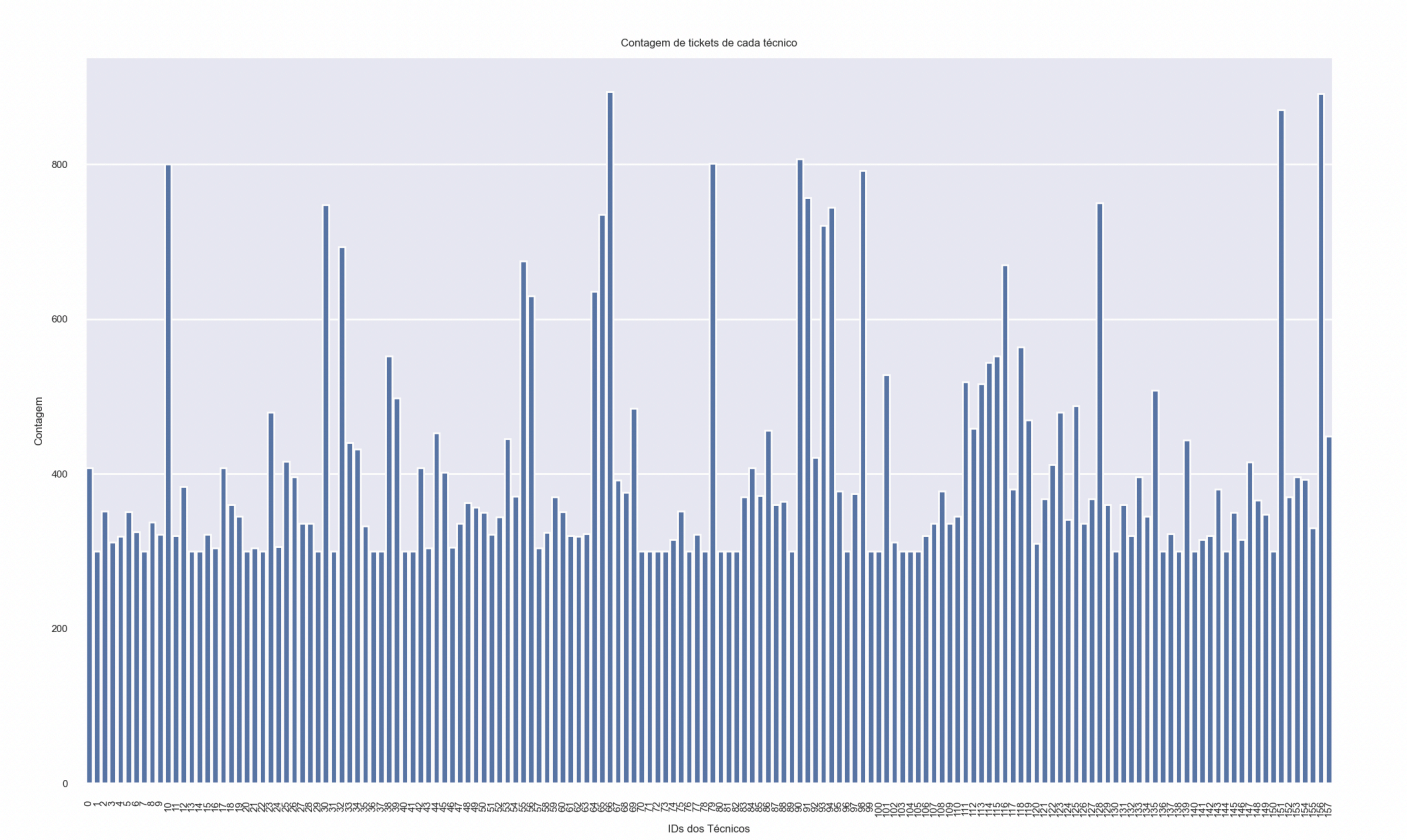
```
n_mult_tickets = (300 - n_existente_tickets) / n_existente_tickets
```

Assim, **n_mult_tickets** representa o número de vezes que cada *ticket* tem de ser multiplicado para que, no total, somando os duplicados e os já existentes previamente, dê pelo menos 300. Os valores foram arredondados às unidades e por excesso, de modo a garantir o mínimo pretendido.

Após este processo o novo *dataset* ficou com 64095 linhas, o que apresenta um número muito superior ao do conjunto anterior, com uma melhor distribuição das classes.

Para comparar com o gráfico de distribuição dos dados do *dataset* original, fez-se também um gráfico deste conjunto de dados, representada na figura 6.2.

Figura 6.2: Distribuição do número de tickets por cada técnico do conjunto de dados oversampled



6.6 Hyperparameter Tuning

Estando escolhido o algortimo *Random Forest*, foram feitos testes numa tentativa de melhorar os resultados obtidos e a qualidade das previsões efetuadas pelo modelo.
Para isso fizeram-se testes com várias modificações nos parâmetros do modelo, analisando as métricas resultantes dessas alterações, de modo a tentar atingir os melhores valores possíveis.
Além do *hyperparameter tuning*, foram também testados dois métodos de treino do PySpark, descritos nas sub-secções **Train Validation Split** e **Cross Validation**, devido a funcionarem de forma diferente, é possível que os resultados dos treinos sejam diferentes.

Entre os parâmetros que o PySpark permite modificar no algoritmo *Random Forest* [15], foram trabalhados os seguintes:

- **numTrees** - Número de árvores a treinar;
- **maxDepth** - Profundidade máxima da árvore, que afeta o número de nós;
- **maxBins** - Granularidade da discretização de recursos contínuos no modelo;
- **minInstancesPerNode** - Número mínimo de instâncias por nó, antes do *split* da árvore;
- **impurity** - Critério utilizado pelo modelo para escolher as melhores *features* e pontos de *split*;
- **subsamplingRate** - Define uma proporção aleatória de exemplos de treino para cada árvore;
- **seed** - Valor que gera aleatoriedade.

Para definir os valores de cada um foi utilizada a biblioteca **numpy**, que possui os métodos ***randint()***, para gerar valores inteiros aleatórios num dado intervalo. e ***uniform()***, para gerar valores decimais.
Assim, foi possível montar uma grelha de parâmetros e utilizar a classe ***ParamGridBuilder()*** do PySpark para gerar todas as combinações únicas possíveis de valores para os parâmetros. Tal foi feito definindo os diferentes campos com *arrays* gerados pelos métodos do numpy, sendo depois invocado o método ***build()*** para calcular as combinações desses valores.

```
paramGrid = ParamGridBuilder()
  .addGrid(modelToTrain.numTrees, [int(x) for x in np.random.randint(100, 201, size=2)])
  .addGrid(modelToTrain.maxDepth, [int(x) for x in np.random.randint(2, 31, size=1)])
  .addGrid(modelToTrain.maxBins, [int(x) for x in np.random.randint(50, 100, size=1)])
  .addGrid(modelToTrain.minInstancesPerNode, [int(x) for x in np.random.randint(2, 16,
  size=1)])
  .addGrid(modelToTrain.impurity, ['gini', 'entropy'])
  .addGrid(modelToTrain.subsamplingRate, [round(np.random.uniform(0, 1), 1)])
  .addGrid(modelToTrain.seed, [18])
  .build()
```

A variável **paramGrid** pode depois ser passada às classes de treino que irão então processar cada combinação única.

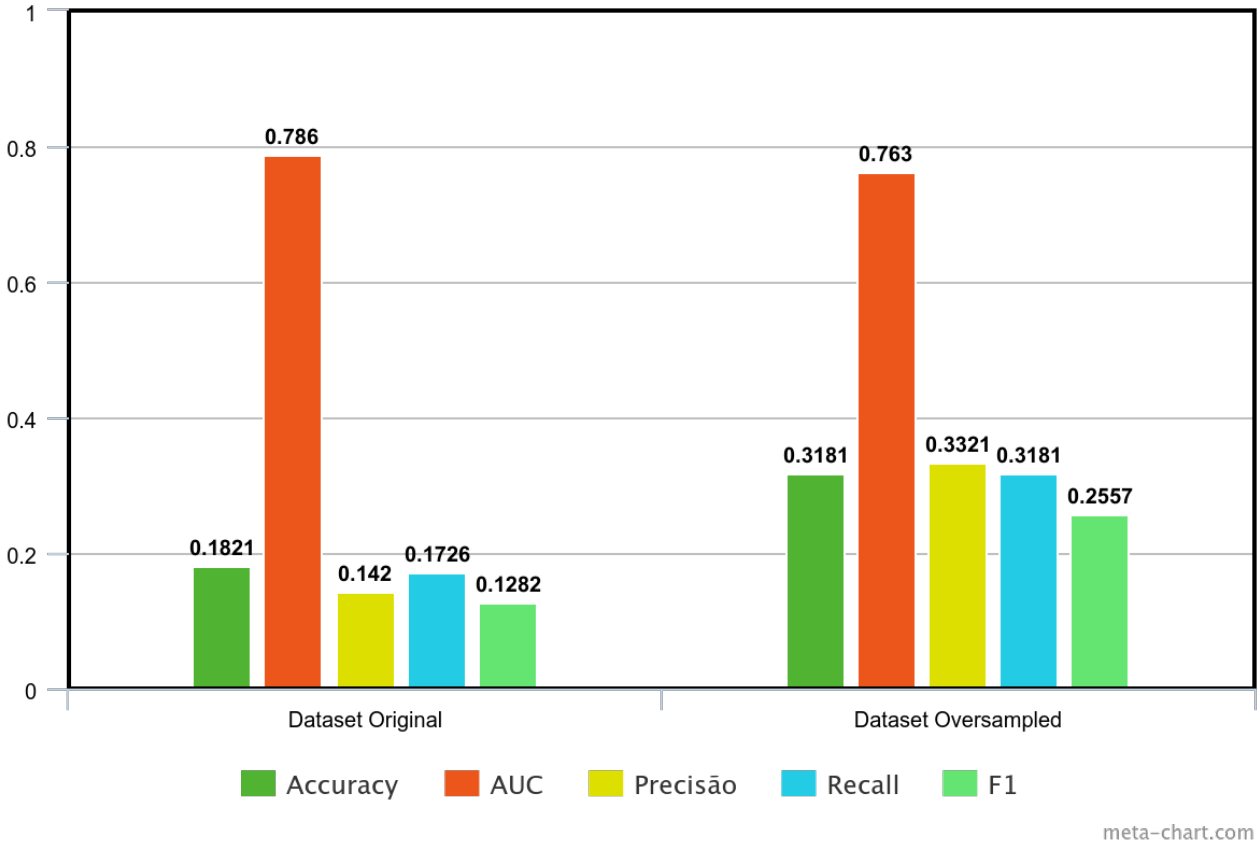
6.6.1 Train Validation Split

O **Train Validation Split** é uma classe do PySpark que serve para realizar *hyperparameter tuning* [16]. Foi-lhe passada a pipeline do modelo a treinar e testar, a grelha de parâmetros gerada pelo **ParamGridBuilder()**, a classe que gera as métricas de avaliação usadas para identificar o melhor modelo, o número de *threads* que desejamos correr em paralelo e a percentagem do dataset original que desejamos repartir em conjunto de treino.

```
trainValidationSplit = TrainValidationSplit(estimator=pipeline,
evaluator=multiEvaluator, estimatorParamMaps=paramGrid, trainRatio=0.8)
```

Este faz a divisão aleatória do *dataset* em dois conjuntos, o de treino e o de validação, no caso testado, 80% devido à existência de poucos dados de origem, é necessário deixar um grupo de treino maior. Desta forma, durante o treino garante-se que o modelo nunca vê os dados que irão ser utilizados para o validar e avaliar, reduzindo problemas de *overfitting* e generalização.
Os melhores valores das métricas obtidos ao fim de testar 11200 modelos diferentes com o *dataset* original e com o *oversampled* foram os representados na figura 6.3.

Figura 6.3: Resultados recolhidos durante os testes ambos os datasets e Train Validation Split

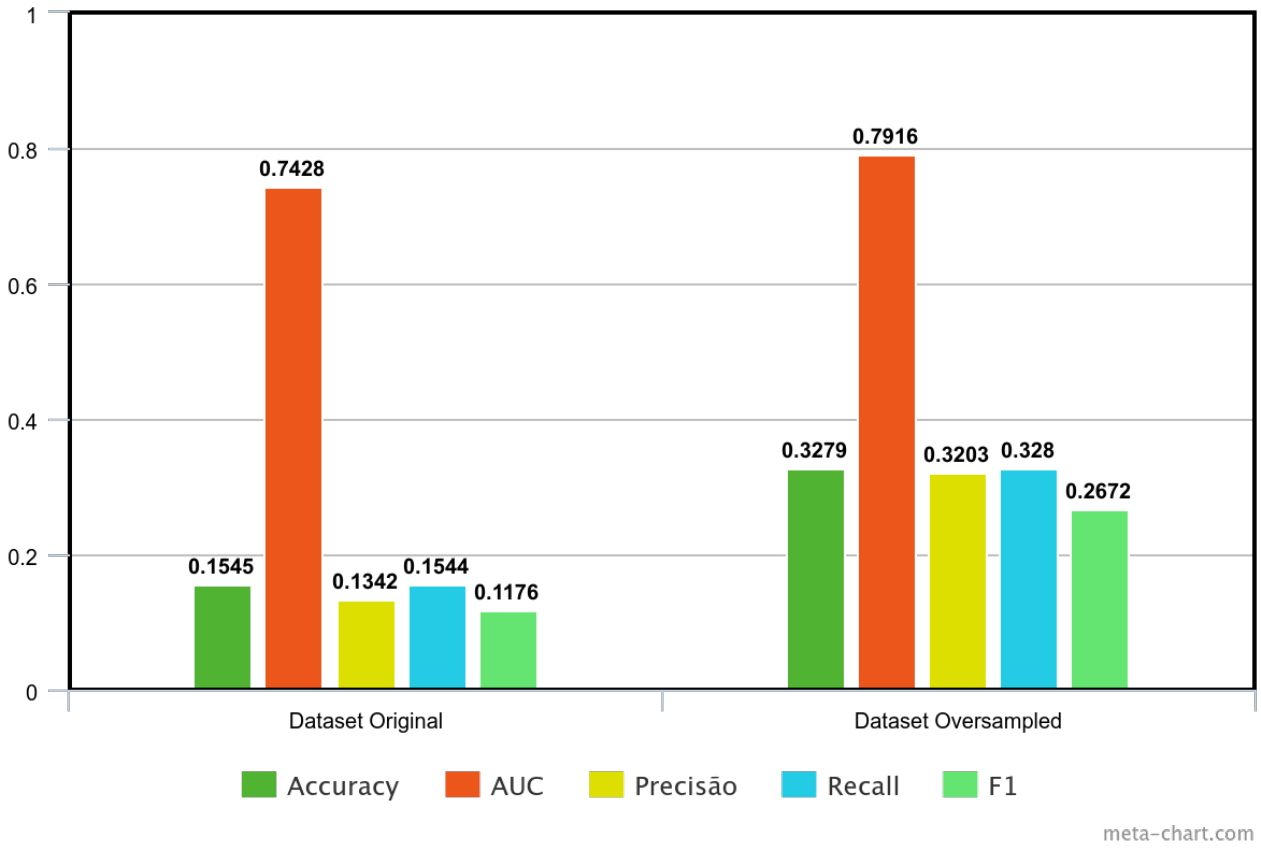


Isto indica que uma quantidade maior de dados permite que o modelo faça previsões melhores, no entanto, como se repetiram muitas vezes os tickets, o aumento dos valores das métricas pode também dever-se à previsão facilitada por isso, o que ao tentar utilizar o modelo para fazer previsões no futuro com dados novos, pode não ser capaz de fazer generalizações e falhar mais que o indicado pelos resultados dos testes.
Ainda assim, as métricas continuam a não ser altas o suficiente para que o modelo seja já utilizado para realizar previsões dos técnicos em ambiente de produção.

6.6.2 Cross Validation

O PySpark oferece também outra classe de treino e *hyperparemeter tuning*, o **Cross Validation** [17], que, ao contrário do método anterior, divide o *dataset* em vários conjuntos aleatórios e não sobrepostos, ou seja, sem que a mesma de linha exista em mais que um conjunto, sendo depois utilizados dois terços dos *datasets* resultantes para o treino e o restante para efetuar os testes, alternadamente.
Desta forma, garante resultados de generalização melhores que os do *Train Validation Split*, uma vez que os modelos são testados com vários conjuntos de dados não vistos. No entanto, os custos computacionais são muito maiores, apesar de também permitir utilizar várias *threads*, existe mais processamento de dados, para as divisões e durante as fases de testagem, o que aumenta bastante a complexidade dos treinos.
Com este método os melhores valores de métricas obtidas foram os descritos na figura 6.4

Figura 6.4: Resultados recolhidos durante os testes ambos os datasets e Cross Validation



6.7 Conclusão

Pode dizer-se que a implementação foi bem sucedida, no sentido de ter sido possível testar os vários algoritmos no sentido de identificar o que melhores resultados ofereceria em termos de qualidade das previsões.
Tendo sido trabalhado o **Random Forest**, este apresentou resultados promissores, no entanto, com a quantidade de dados disponível e a variedade que estes apresentam não permitiu que se obtivesse um modelo com mais de 33% de *accuracy*, além dos valores baixos das outras métricas, o que não é satisfatório para que este seja utilizado com o objetivo pretendido para já. Os testes efetuados sobre o *dataset* que sofreu multiplicação das linhas dos técnicos que possuíam poucos *tickets* indicaram que uma maior quantidade e melhor distribuição dos dados poderá permitir atingir melhores previsões.
Os treinos efetuados através de **Cross Validation** retornaram melhores resultados, o que indica que este será o método preferencial a utilizar, apesar dos custos computacionais. Isto vem do facto de o **Train Validation Split** dividir os dados apenas uma vez o que gera problemas de generalização, e ao efetuar os testes leva a métricas mais baixas, enquanto esta forma de treinar os modelos implica mais divisões dos dados e testes com dados não vistos anteriormente, reduzindo-os.

Capítulo 7

Análise dos Resultados

7.1 Introdução

Neste capítulo são analisados os resultados recolhidos durante os testes, de modo a perceber a viabilidade do modelo para utilização por parte da Code-495 Solutions. Analisam-se as diferentes métricas obtidas com os dois métodos discutidos, Train Validation Split e Cross Validation, com ambos os *datasets*, original e *oversampled*, de modo a poder fazer-se uma comparação e chegar a conclusões mais detalhadas.

7.2 Análise das Métricas

Ao analisar as várias métricas recolhidas durante os testes, pode chegar-se a várias conclusões acerca dos dados.
Em termos de *accuracy*, os baixos valores de 0.1821 e 0.1545 com ambos os métodos **Train Validation Split** e **Cross Validation**, respetivamente, indicam que os modelos não estão a ser capazes de realizar previsões dos técnicos adequados aos problemas. Ao comparar com os resultados de 0.3181 e 0.3279, dos mesmos métodos, obtidos ao utilizar o *dataset oversampled*, leva a crer que se trata de um problema de sub-representação das classes minoritárias, em que o modelo rotula a maioria dos *tickets* com os identificadores dos técnicos mais comuns numa tentativa de atingir os melhores valores, reduzindo a *accuracy*.
Apesar dos segundos resultados terem sido superiores, é importante analisar as outras métricas, como o *Recall* e o F1, pois pode ser uma subida artificial, consequência do *oversampling*.
Os valores das métricas recolhidos com os primeiros testes, precisões de 0.1420 e 0.1342, *recalls* de 0.1726 e 0.1544, f1 de 0.1282 e 0.1176, do *Train Validation Split* e do *Cross Validation*, respetivamente.
Os dois primeiros apontam para uma incapacidade dos modelos de representar a classe minoritária não sendo capaz de identificar esses casos como positivos, identificando apenas 17,26%, no máximo, dos casos da classe minoritária como positivos, ou seja, bem atribuídos.
Os valores de F1 reforçam essa ideia, já que ao equilibrar as duas métricas anteriores resultou numa métrica muito baixa.
O aumento da quantidade dos dados ao executar os testes sobre o *dataset* que foi *oversampled*, levou a um aumento dos valores das três métricas, o que apresenta um avanço em relação aos dados originais, no entanto, assim como na *accuracy*, pode ser uma subida não real, e em cenário de produção o modelo pode não estar apto para discernir corretamente sobre o técnico mais apto para uma situação, além de que o aumento da classe minoritária durante o *oversampling* pode também não representar a realidade e, no futuro, continuar a haver classes pouco representadas, por exemplo, em casos de serviços muito específicos e não recorrentes.

Em termos de AUC os valores mantiveram-se relativamente altos ao longos dos testes tendo valores de 0.7860 e 0.7428, dos métodos *Train Validation Split* e *Cross Validation*, respetivamente, recolhidos durante os testes com o *dataset* original, não sido muito afetados, chegando-se a valores de 0.7630 e 0.7916 com os dados *oversampled*, indicando que apesar das previsões não serem de qualidade, o modelo é capaz de distinguir as diferentes classes satisfatoriamente.

7.3 Conclusão

Após analisar os resultados dos vários testes, pode afirmar-se que não existem dados suficientes para obtenção de um modelo capaz de realizar previsões em contexto real de atribuição de técnicos para solucionar problemas de *tickets*, sendo 18604 *tickets* não suficientes para definir as tomadas de decisão face à quantidade de combinações possíveis entre os possíveis problemas e locais onde estes ocorrem. Com mais *tickets* de histórico será possível analisar melhor os diferentes cenários, pois além de uma representação mais fiável da realidade, reduz-se também o ruído aleatório.
O *oversampling* do *dataset* original levou a melhores resultados, mas é difícil definir se em contexto real essas métricas se iriam manter, pois o aumento do número de representantes das classes minoritárias pode não se verificar na realidade e causar novo desequilíbrio nas previsões.

Capítulo 8

Conclusão

Sendo o objetivo principal do projeto o desenvolvimento de um módulo de ***Machine Learning*** capaz de realizar previsões sobre técnicos da Excell Communications adequados para resolver vários tipos de problemas a eles reportados com base em dados de *tickets* anteriormente resolvidos.
Assim, o objetivo não foi atingido com sucesso, já que, ao realizar os treinos e testes sob o algoritmo **Random Forest**, não foi possível atingir resultados satisfatórios. Isto vem do facto de, devido a ser um recurso recente no sistema da **Code-495 Solutions**, o que leva a que ainda não exista uma quantidade de dados suficiente que permita aplicar este tipo de técnicas.
Os modelos treinados não apresentaram aptidão para realizar previsões corretamente, apesar de serem capazes de identificar de forma algo correta as diferentes classes (cerca de 75% das vezes), ou seja, os diferentes técnicos.

8.1 Desafios Enfrentados

Inicialmente pensou-se trabalhar dois tipos de *tickets*, UTT e DSTX, tendo sido feito planeamento nesse sentido, no entanto, a existência de apenas 675 exemplos do primeiro tipo, seria impossível realizar este tipo de trabalho sobre eles, pelo que foi necessário readaptar algumas tarefas, no caso do Pré-Processamento dos Dados, já que depois disso foram descartados.
O trabalho sobre um *dataset* reduzido apresentou bastantes desafios, principalmente a nível de análise, já que quando existe uma quantidade grande de dados, as métricas tornam-se mais fiáveis, representando melhor o problema que estará a afetar os modelos e, neste caso, como todas as métricas de análise das previsões eram baixas, poderia ter várias origens, tendo-se tentado fazer o maior número de despistes possível, desde a introdução de *feature selection*, ao *oversampling* dos dados e utilização de diferentes métodos de treino, numa tentativa de reduzir ao máximo as possibilidades.

8.2 Considerações Finais

Apesar de não ter sido possível concluir o projeto com um modelo capaz de realizar as previsões pretendidas, o processo de planeamento e desenvolvimento foi bastante importante no sentido de solidificar os conhecimentos relacionados com a área de **Inteligência Artificial**. A necessidade de aplicar diversas técnicas relacionadas com o pré-processamento dos dados para, de seguida, estes serem processados corretamente durante a etapa de treino e testes ao algoritmo levou a que estes trabalhos fossem feitos praticamente, em contexto real, levando-me a aprofundar os conhecimentos adquiridos de forma bastante enriquecedora. A análise dos resultados implicou julgamento crítico de várias métricas independentes e complementares, o que também levou a uma melhor compreensão do objetivo destas e da importância do estudo dos resultados no sentido de tentar melhorá-los.

8.3 Trabalho Futuro

No futuro seria interessante realizar novos testes com um *dataset* maior, com mais exemplos reais de *tickets*, podendo fazer-se um treino regular do modelo, por exemplo, até se obterem resultados melhores e aí, então, introduzir a previsão automática no sistema da Code-495 Solutions.
Outra possível melhoria seria o desenvolvimento de vários modelos, um para cada tópico, por exemplo, de modo a separar os problemas regulares dos menos comuns, resultado de diferentes tipos de situações, criando um melhor equilíbrio na distribuição do número de *tickets* de cada técnico. Isto também permite uma implementação faseada, conforme cada tópico possuía dados suficientes para trabalhar, além que depois de implementados, cada modelo levaria menos tempo a ser treinado e testado com os novos dados, pois cada *dataset* teria muito menos ruído e linhas a analisar, sendo este mais específico.

Bibliografia

[1] "Artificial intelligence market size, share trends analysis report by solution, by technology (deep learning, machine learning, nlp, machine vision, generative ai), by function, by end-use, by region, and segment forecasts, 2024 - 2030." [Online]. Available: https://www.grandviewresearch.com/industry-analysis/artificial-intelligence-ai-market 1

[2] Y. Duan, J. S. Edwards, and Y. K. Dwivedi, "Artificial intelligence for decision making in the era of big data – evolution, challenges and research agenda," *International Journal of Information Management*, vol. 48, pp. 63–71, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0268401219300581 5

[3] J. Ribeiro, R. Lima, T. Eckhardt, and S. Paiva, "Robotic process automation and artificial intelligence in industry 4.0 – a literature review," *Procedia Computer Science*, vol. 181, pp. 51–58, 2021, cENTERIS 2020 - International Conference on ENTERprise Information Systems / ProjMAN 2020 - International Conference on Project MANagement / HCist 2020 - International Conference on Health and Social Care Information Systems and Technologies 2020, CENTERIS/ProjMAN/HCist 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921001393 5

[4] B. Axmann and H. Harmoko, "Robotic process automation: An overview and comparison to other technology in industry 4.0," pp. 559–562, 2020. 5

[5] B. Mahesh, "Machine learning algorithms -a review," 01 2019. 6

[6] "Main guide - spark 3.5.1 documentation." [Online]. Available: https://spark.apache.org/docs/latest/ml-guide.html 8

[7] "Why tensorflow." [Online]. Available: https://www.tensorflow.org/about?hl=en 8

[8] "Structtype — pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.types.StructType.html#pyspark.sql.types.StructType. 27

[9] "Structfield — pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.sql.types.StructField.html#pyspark.sql.types.StructField. 27

[10] "Stringindexer - pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.StringIndexer.html 28

[11] "Indextostring - pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.IndexToString.html 28

[12] "Vectorassembler - pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.VectorAssembler.html 28

[13] "Binaryclassificationevaluator - pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.evaluation.BinaryClassificationEvaluator.html 29

[14] "Multiclassclassificationevaluator - pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.evaluation.MulticlassClassificationEvaluator.html 29

[15] "Randomforestclassifier — pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/3.1.3/api/python/reference/api/pyspark.ml.classification..html 36

[16] "Trainvalidationsplit — pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.tuning.TrainValidationSplit.html 37

[17] "Crossvalidator — pyspark 3.1.3 documentation. apache spark tm." [Online]. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.tuning.CrossValidator.html 38