



Malware Detection – Email & Network Analysis

Gecad

2022 / 2023

1200627 - Tiago Marante

Malware Detection – Email & Network Analysis

2022/2023

1200627 - Tiago Marante



Licenciatura em Engenharia Informática

Setembro 2022

Orientador ISEP: **Isabel Praça**

Supervisor Externo: **Orlando Sousa & Eva Maia**

“Power isn’t determined by your size, but the size of your heart and dreams.”

Someone out there

Agradecimentos

Antes de mais gostaria de agradecer aos meus pais, Olga e José por me terem até ao dia de hoje financiado toda a minha educação, pois se não tivesse sido por eles neste momento não estaria aqui. Também gostaria de agradecer aos meus amigos, que sem dúvida são uma enorme força, pois é graças a eles que os sonhos são formados, e quem sabe são com este que os realizamos. Todos apoio destas pessoas nunca serão esquecidos.

De igual modo, gostaria de agradecer ao GECAD pela oportunidade que me garantiu, pois sem dúvida fiz algo que não estava à espera muito menos habituado, pois passei de uma abordagem de desenvolvimento de software para uma abordagem mais de investigação.

Agradeço também à minha orientadora, Prof. Isabel Praça, que me foi ajudando ao longo do tempo do que fazer para melhor o projeto quando estava mais perdido, bem como agradecimentos ao Prof. Orlando Sousa por ter recomendando novas e melhores formas de fazer o algoritmo.

Em suma também quero agradecer ao restantes colegas do GECAD e do ISEP, pois parecendo que não, toda a ajuda ao longo destes tempos foram fundamentais, e ajuda esta que estarei eternamente agradecido.

Resumo

A cibersegurança tem vindo a ser uma grande preocupação nos dias de hoje. Existe uma necessidade de evoluir e construir novas técnicas que nos permitam ficar mais seguro e nos salvaguardar destas ameaças. Para tal, técnicas preventivas que usam inteligência artificial têm sido cada vez mais exploradas. O objetivo deste projeto será então optar por essas técnicas automáticas e com a ajuda de machine learning e deep learning formar modelos preventivos para análise e deteção de phishing em emails, desta forma para alcançar uma solução mais segura e eficaz.

Efetuada uma revisão da literatura sobre esforços anteriores para encontrar casos de phishing na área da cibersegurança utilizando inteligência artificial, este projeto propõe a criação de um modelo de inteligência artificial, neste caso uma árvore de decisão, que analisando prévios emails de phishing será capaz de inferir se este é ou não um. Vários modelos serão analisados e estudados, bem como outras técnicas, desde Logistic Regression, Redes Neurais, Naive Baeys ... Também é proposto a resolução de uma API para fornecer a qualquer colaborador ou até mesmo a qualquer pessoa uma maneira simples de analisar o conteúdo do mesmo.

Conclui-se a partir deste projeto que o sistema criado é de alta confiança, demonstrado bons resultados na deteção. Como este sistema funciona à base de técnicas de “*machine learning*”, este sistema é bastante flexível e preciso na deteção de phishing em emails com idioma inglês. Caso este sistema comece a perder alguma precisão devido a novas técnicas usadas por atacantes basta somente dar de input mais resultados recentes e este automaticamente se auto adaptará.

Keywords (Context): Inteligência artificial, Cibersegurança, Machine Learning, Phishing

Keywords (Technologies): Python, Docker, Scikit-learn

Índice

1	<i>Introdução</i>	2
1.1	Enquadramento/Contexto	2
1.2	Descrição do Problema	2
1.2.1	Objetivos	3
1.2.2	Abordagem	4
1.2.3	Contributos	5
1.2.4	Planeamento do trabalho	6
1.3	Estrutura do relatório	7
2	<i>Estado da arte</i>	8
2.1	Cibersegurança	8
2.1.1	Inteligência artificial aplicada à segurança	10
2.2	Tecnologias existentes	13
2.2.1	Python	13
2.2.2	Keras vs Tenserflow vs Scikit-learn	14
2.2.3	Numpy vs Pandas	16
2.2.4	FastAPI	17
2.2.5	Docker	18
3	<i>Ambiente de Trabalho</i>	21
3.1	Metodologias de trabalho	21
3.1.1	CRISP-DM	22
1.	<i>Entendimento do negócio – O que o negócio precisa?</i>	22
2.	<i>Compreensão dos dados – Que dados temos/precisamos? Está limpo?</i>	22
3.	<i>Preparação de dados – Como organizamos os dados para modelagem?</i>	22
4.	<i>Modelagem – Quais técnicas de modelagem devemos aplicar?</i>	22
5.	<i>Avaliação – Qual modelo melhor atende aos objetivos do negócio?</i>	22
6.	<i>Implantação – Como as partes interessadas podem ver os resultados?</i>	22

3.2	Versões de Controlo.....	24
4	Implementação da Solução	27
4.1	Descrição da implementação	27
4.1.1	Resultados para o Random Forest	28
4.1.2	Resultados para o Naive Baesys	29
4.1.3	Resultados para o Logistic Regression	30
4.2	Análise dos resultados e melhoramentos	31
4.2.1	Undersampling	31
4.2.2	Hyper Tuning.....	32
4.2.3	Oversampling	35
4.2.4	Desenvolvimento da API	37
4.2.5	Rotas da API	39
4.3	Testes	42
4.4	Avaliação da solução.....	43
5	Conclusões	45
5.1	Objetivos concretizados.....	45
5.2	Limitações e trabalho futuro	48
5.3	Apreciação final	49
6	Referências	50

Índice de Figuras

Figura 1 – NIST cybersecurity framework pillars [4]	9
Figura 2- CRISP-DM diagrama.....	22
Figura 3-Dataset Plot	27
Figura 4-Random forest unbalanced data	28
Figura 5 - Naïve Baeys unbalanced data	29
Figura 6-Logistic Regression unbalanced data.....	30
Figura 7 - Random Forest balanced data (undersampling).....	31
Figura 8 - Random Forest Tuning depois de undersampling	33
Figura 9 - Resultado depois do treino.....	34
Figura 10 - SMOTE data	35
Figura 11 - Visualização do dataset depois do SMOTE	36
Figura 12 - Visualização do dataset depois do SMOTE	36
Figura 13 - Código responsável pela análise do email	38
Figura 14 - Rotas da API.....	39
Figura 15 - Conteúdo Json para a rota /email	40
Figura 16 - Docker File para construção da imagem	41

Índice de Tabelas

Tabela 1 - Python e Java as diferenças	13
Tabela 2- Objetivos concretizados	47

Notação e Glossário

AI	Artificial Intelligence
ANN	Artificial Neural Networks
API	Application Programming Interface
CISA	Cybersecurity & Infrastructure security agency
CPS	Cyber-Physical System
Dos	Denial of Service
DP	Deep Learning
GECAD	Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development
IoT	Internet of Things
KNN	K-Nearest Neighbors
KNN	K-Nearest Neighbors
LSTM	Long-Short Term Memory
MITM	Man in the Middle
MLP	Multi-Layer Perceptron
R&D	Research and development
RF	Random Forest
SMOTE	Synthetic Minority Oversampling Technique
SQL	Structured Query Language
SVM	Support Vector Machines

1 Introdução

Este capítulo fornece uma visão geral do contexto e objetivos do projeto, bem como as motivações pessoais do autor. A abordagem empregada é brevemente descrita e as principais contribuições derivadas deste trabalho são cuidadosamente indicadas. O plano de trabalho e a estrutura do relatório são também abordado.

1.1 Enquadramento/Contexto

Este trabalho deriva de uma bolsa de investigação no Grupo de Pesquisa em Engenharia Inteligente e Computação para Inovação e Desenvolvimento Avançado (GECAD) [1], que por sua vez, é uma referência nacional e internacional, unidade de investigação e desenvolvimento de soluções inteligentes (R&D) de ponta para um vasto leque de setores, nomeadamente, Mercados de Eletricidade, Smart Grids, Internet de coisas (IoT) e Sistemas Ciber-Físicos (CPS).

O objetivo deste projeto consistirá em criar um Sistema de Detecção de Phishing, de tal modo que qualquer pessoa possa o usar de maneira muito facilmente. Este sistema tem que suportar vários pedidos ao mesmo tempo, bem como ser altamente preciso. Este dará uma percentagem de quanto acha que o email será ou não suspeito e a partir daí fica ao critério do utilizador se continua ou não com o email.

1.2 Descrição do Problema

Com o avançar da tecnologia, as técnicas de ataques começaram também estas a evoluir e para atender a este problema, técnicas para defesa destes também tem que evoluir. Para tal, o uso de machine learning tem sido cada vez mais usado nos mais diversos problemas, desde deteção de phishing de emails [2], deteção de malware apesar que esta não irá funcionar em malware polimórfico e metamórfico que mudam o seu código de máquina para máquina fazendo que não haja padrões em comum, também é usado na deteção de intrusões entre outros.

Em 2019, o algoritmo da Google para detecção de spam e de phishing era baseado no algoritmo Naive Bayes que mais à frente este será abordado e usado.

Este projeto visa melhorar a cibersegurança de qualquer pessoa/entidade de forma a ter uma camada a mais de segurança para dificultar possíveis ataques via email.

1.2.1 Objetivos

Este projeto terá como principal objetivo principal desenvolver uma solução combinada com segurança, bem como machine learning e estatística. A ideia seria ter acesso a diversos email recebidos ao longo do tempo em uma determinada organização, mas como tal não foi possível terá que ser usado dataset públicos para o mesmo efeito. Importante lembrar que quanto maior o dataset mais preciso será o modelo para futuras previsões.

Depois de ter o dataset pronto bem como o seu conteúdo completamente normalizado e limpo, falta somente usar os diversos modelos para avaliar qual destes modelos estatísticos apresentará uma maior taxa de sucesso. Este trabalho foi desenvolvido neste contexto e teve os seguintes objetivos principais:

- Estudo de diferentes modelos e abordagens de AI;
- Extração dos dados em tokens visto que estes modelos funcionam melhor com números em vez de texto;
- Distribuição do modelo final para o maior número de pessoas;

Foi realizada uma pesquisa para determinar o modelo de AI mais adequado para a compreensão e utilização prática, bem como desenvolvido uma solução de software para fornecer as funcionalidades necessárias.

1.2.2 Abordagem

A abordagem adotada para este projeto é composta por quatro fases principais:

- Entender os requisitos do projeto;
- Explorar o estado da arte sobre os desafios proporcionados por este projeto;
- Reformular a implementação já feita;
- Implementar e testar os novos modelos, ou até novos datasets;

No primeiro ponto serão abordados outros papers, papers estes que serão analisados e estudados de forma a decidir quais os melhores modelos/datasets para poder obter o melhor resultado possível. Relativamente ao segundo ponto, perante os modelos abordados nos diversos papers, serão estes também criados e estudados. Também serão estudadas as melhores tecnologias para a resolução dos mesmos. No ponto 3, serão abordadas as metodologias usadas bem como ferramentas de versão de controlo. No ponto 4, serão explicados todos os processos desde limpeza de dados, até aos vários modelos usados, tuning de hiperparâmetros entre outros.

Por fim, é discutido trabalho futuro, bem como entraves da solução, no caso da solução não retornar os resultados pretendidos, caso aconteça isso, segue-se para o próximo modelo, ou até mesmo outro dataset. É pertinente lembrar que os dois últimos pontos seguem a metodologia CRISP-DM.

1.2.3 Contributos

Este projeto apresenta uma abordagem para a deteção de phishing baseado em certas regras. Ele propõe um sistema que estará recebendo emails em tempo real de vários utilizadores. A principal contribuição aqui é para R&D na descoberta de uma maneira simples e eficaz na deteção de emails maliciosos, usando machine learning de forma simples, eficaz e muito barata.

Empresas podiam beneficiar deste sistema, uma vez que se destina a ser implementado em tais contextos físicos, por exemplo no servidor de email.

Qualquer pessoa também poderia tirar como benefício, pois muitos emails de phishing continuam a passar, e no caso de até o utilizador usar um email que não apresenta filtros para tal, pois pode-se tratar de um email mais antigo, este algoritmo podia dar a essas pessoas uma mais-valia, isto é, uma camada extra de segurança.

Pois o algoritmo irá fazer a respetiva análise e deteção caso a haja, de tal forma que podia ajudar a proteger os dados pessoais dos indivíduos de serem expostos para uso malicioso.

Além disso, do ponto de vista da engenharia de software (ES), não é tarefa fácil projetar um sistema como este, pois é introduzido Aprendizagem Automática. Isso pode servir como um bom exemplo para outros desenvolvedores de como integrar Aprendizagem Automática com lógica do negócio.

1.2.4Planeamento do trabalho

Foi desenvolvido um plano de trabalho para assegurar o cumprimento dos objetivos declarados. O plano foi ajustado em encontros semanais distribuídos ao longo da duração do estágio. O fluxo de trabalho final pode ser descrito da seguinte forma:

1. Primeiro contato com os requisitos do projeto. Estudo autónomo de Machine Learning (ML) (desde 01/03/2020 até 01/04/2020).
 - a. Compreensão do projeto;
 - b. Pesquisa de datasets públicos, para respetiva análise;
 - c. Pesquisa de abordagens de Machine Learning (ML) de última geração;
 - d. Criação de um modelo usando o Logistic Regression;
 - e. Criação de um modelo usando o Naive Baeyes;
 - f. Criação de um modelo usando o Random Forest;
2. Implementação de técnicas de oversampling e de undersampling, e tuning aos parâmetros relativamente ao undersampling (desde 01/04/2020 até 01/05/2020).
 - a. Oversampling usando SMOTE;
 - b. Undersampling usando Pandas;
 - c. Tuning dos hiperparâmetros dos modelos;
 - d. Documentação sobre o dataset;
 - e. Documentação dos métodos e resultados obtidos;
3. Produção de uma API usando o modelo mais preciso tendo em conta o f1-score, depois de este estar tunned (desde 01/05/2020 até 01/06/2020).
 - a. Aprendizagem de como usar python para backend.
 - b. Aprendizagem de docker com python.
 - c. Aprender a usar FastAPI, de modo a o modelo poder ser usado por várias pessoas em simultâneo

4. Escrita do relatório de estágio (desde 01/06/2020 até 01/09/2020).
 - a. Documentação de análise e projeto;
 - b. Relatar todos o trabalho realizado e problema até aqui ocorridos;
 - c. Conclusão do mesmo;

1.3 Estrutura do relatório

Este relatório está dividido em seis capítulos principais, que são:

1. **Estado da arte:** Apresenta uma revisão de literatura sobre alguns dos desafios que esta projeto fornece, bem como algumas tecnologias que podem ser usadas para implementar este projeto.
2. **Metodologias de trabalho:** Apresenta uma breve explicação das metodologias de trabalho utilizadas e apresenta algumas ferramentas que foram utilizadas a fim de demonstrar como e porquê que foram utilizadas.
3. **Implementação da solução:** apresenta a solução final produzida a partir de um ponto de implementação. Neste capítulo, são fornecidas explicações sobre as decisões tomadas quando implementando o sistema, bem como melhorias e até alguns entraves.
4. **Conclusão:** apresenta os objetivos alcançados, bem como possível continuação em um futuro.

2 Estado da arte

Neste capítulo, irá ser demonstrado a informação relativamente à cibersegurança bem como à inteligência artificial de maneira a aumentar o conhecimento dos leitores acerca deste problema de domínio. A escrita do mesmo foi feita de maneira cuidada e adequada às principais contribuições dos trabalhos de outros autores.

2.1 Cibersegurança

Desde o início da era dos computadores, o tema de segurança tem sempre mantido a sua presença, porém só recentemente é que este tem vindo a ser um tema mais importante para o dia a dia.

Um ataque malicioso, é aquele que busca aceder a dados ilegalmente, interromper um serviço ou até danificar informação. Estes ataques, podem ser originados por múltiplos atores, dos quais podem variar desde hackers, a terroristas, espiões ou até mesmo de funcionários. De acordo com o seguinte website [3], existem sete tipos de ameaças à segurança cibernética:

- **Malware:** Software instalado no computador da vítima, este pode incluir spyware, ransomware, vírus, vermes, entre outros;
- **DOS:** Tipo de ataque cibernético que sobrecarrega um ativo ou rede, deixando-o incapaz de responder a pedidos.
- **Emotet:** Como pode ser descrito pela CISA, este é um trojan avançado que ataca na sua maioria bancos.
- **Phishing:** Esse tipo de ataque finge uma comunicação credível, como e-mail para enganar o recetor a executar instruções dentro dele.
- **SQL injection:** Conforme o nome indica, consiste na injeção de código malicioso, que irá provocar dumps da base de dados.
- **Ataques de Password:** Descoberta da senha certa através de força bruta para aceder a recursos restritos.

- **MITM:** Esse tipo de ataque ocorre quando um hacker obtém acesso não autorizado a uma transação entre duas entidades.

Com a evolução da tecnologia das quais a internet das coisas (IoT), muitos destes dispositivos começaram a ser alvo de ataques, abrindo assim portas para os intrusos ganharem acesso.

Para mitigar tais problemas, qualquer organização deve ter alguns modelos de cibersegurança bem como protocolos. De uma forma geral, existe uma framework que nos pode ajudar com este tipo de trabalho, a “Cybersecurity Framework”, é composta por três pilares: pessoas, processos e tecnologia. Estes pilares devem funcionar juntos, de maneira a criar uma defesa eficaz. Cada pilar é definido como:

- **Pessoas:** Trabalhadores devem perceber e cumprir simples princípios de segurança
- **Processos:** As organizações por norma deviam de ter modelos específicos ou até mesmo uma framework para lidar com ataques, sendo a NIST um bom exemplo [4].
- **Tecnologia:** Esta é essencial para fornecer às organizações e aos indivíduos ferramentas de segurança para ajudar a pôr em prática os processos implementados. As três principais entidades que devem ser protegidas são dispositivos de endpoint, redes e a nuvem.



Figura 1 – NIST cybersecurity framework pillars [4]

2.1.1 Inteligência artificial aplicada à segurança

Segundo uma recente literatura, alguns estudos nesta área, certos métodos de inteligência artificial, dos quais, Random Forest (RF), Multi-Layer Perceptron (MLP), Long-Short Term Memory (LSTM), Decision Trees (DT), Artificial Neural Networks (ANN), K-Nearest Neighbors (KNN) e Support Vector Machines (SVM), têm se mostrado bastante promissores e com uma taxa elevada de detecção dos mesmos. Visto que na área de segurança existe uma enorme dinâmica relativamente aos ataques, faz sentido usar métodos não lineares, mas sim probabilísticos, de maneira a ser o mais geral possível. Um dos exemplos práticos, é o que a universidade Federal de Pernambuco, que usou vários algoritmos para a detecção de vírus, sendo que as redes neurais foram as que prevaleceram com a maior percentagem de acerto [6].

Para este caso, foram precisas várias amostras de ficheiros infetados e não infetados, depois foi usado a API do Vírus Total para fazer a sua classificação real. Depois de ter os resultados baseados nos melhores antivírus do mercado, foram extraídos desses ficheiros infetados ou não mais de 630 features [6]. Depois de ter as classificações reais, e as features falta só por um modelo de AI a aprender. Segundo o paper acima falado, foram usados 11 tipos de funções de aprendizagem baseadas em MLP com backpropagation. Depois, as outras redes irão se diferenciar perante o número de neurónios sendo que o máximo foram 1261 em cada uma das “hidden layers”. No fim, foi obtido uma percentagem de 99% o que é bastante bom tendo em conta o orçamento obtido em comparação a qualquer antivírus do mercado pois muito dinheiro foi investido nestes. Porém, é importante lembrar que os 99% é refletido nos ficheiros de input, pois esta percentagem iria descer de ano para ano, visto que a milhões de vírus são criados de maneiras cada vez diferentes, e por vezes o modelo pode não conseguir prever tais mudanças repentinas.

Conforme se pode constatar acima descrito, a inteligência artificial, está muito presente nos temas cibernéticos. Uma das utilidades da mesma, que será abordado com mais detalhe ao longo deste relatório, é a detecção de phishing em emails.

Este tópico já vem a ser um problema desde o início da internet, pois da parte dos atacantes houve sempre a tentativa de explorar sistemas e vítimas, apesar que este tipo de ataque se baseia muito em ataques que se aproveitam diretamente das pessoas.

De acordo com o nome, estes ataques baseiam-se em ataques de engenharia social, juntamente com um grande número de emails para vários remetentes na esperança de pelo menos um funcionar. Segundo relatórios de tendências de ataque de phishing do APWG [7], o número de ataques de phishing observados vem a aumentar desde 2020, ano de pandemia, e a cada ano que passa tem aumentado quase para o dobro.

Estes ataques são espalhados por e-mail, sms, mensagens instantâneas, redes sociais, etc., mas o e-mail é sem dúvida a maneira mais popular de realizar este ataque. O e-mail de phishing pode levar desde a roubo de identidade até perdas monetárias. O atacante sempre que envia um e-mail tende a fazer a vítima acreditar que esta encontra-se a comunicar com uma entidade confiável e tenta enganar para lhe ser fornecido credenciais de email ou contas bancárias para depois o atacante poder atacar o serviço ou os interesses da vítima. Muitas informações que os atacantes pedem são os números de cartão de crédito, dados de login da conta ou informações de identidade.

Para termos uma ideia, do quanto ridículo de número de email de phishing foram mandados, no ano 2019 estimasse que foram mandados cerca de 293.6 biliões de emails diários, aos quais 47.3% eram maliciosos, que por sua vez causaram problemas sociais e perdas económicas [8]. Muitos destes continham cavalos de troia, spyware e ransomware.

Posto isto, as companhias tinham que começar a preocupar-se com isto, pois bastava somente um funcionário para por uma empresa em risco, assim foram inventadas técnicas de maneira a eliminar potências fraudes, de tal modo que os funcionários já não corriam o risco de abrir pois o algoritmo eliminava o email com antecedência.

Estes chamados algoritmos, são nada mais nada menos que algoritmos de inteligência artificial. Estes podem ter como base um dataset dos diversos emails em lista negra e também com base em aprendizado de máquina (ML). Porém recentemente Deep Learning (DL) tem surgido com técnicas mais eficientes [9].

2.2 Tecnologias existentes

Nesta seção, será feita uma revisão das tecnologias que serão ou poderão ser usadas para a implementação deste projeto.

2.2.1 Python

Python é relativamente similar ao Java, uma linguagem de programação de alto nível e orientada a objetos. Esta língua não precisa de compilação do programa antes da execução, pois é uma linguagem interpretada, e faz isso no runtime, o que significa que ele executa o código linha por linha. A biblioteca padrão do Python é vasta, contendo muitas funções reutilizáveis, e é bem complementada com um gerenciador de pacotes (pip), que permite o acesso a bibliotecas externas, ampliando a quantidade e variedade de conteúdo disponível, bem como a sua ideologia de open-source [10]. O Python pareceu-me ser o mais adequado para a componente de AI, pois facilita o uso e criação de modelos de inteligência artificial, Machine Learning (ML) e análise de dados. As bibliotecas mais usadas para estas tarefas anteriormente descritas e que serão mais explicadas ao longo do relatório, são: Keras, Tensorflow, Scikit-learn, NumPy e Pandas.



Dimension	 Python	 Java
Compilation	Python is an Interpreted language	Java is a Compiled language
Typing-discipline	Dynamic typed (duck-typing)	Statically typed
Learning Curve	Easier than Java	Easy to learn
Performance	Slower	Fast
Verbosity	Concise	Verbose
Database Support	Comparatively weak	Stable
Ideally suitable for	Data science, ML, AI	Embedded & Cross-platform apps
Other Tools built using the technology	Django, Flask, Pycharm, CircleCI	Docker, Android SDK, Spring Boot, Sentry
Companies using	Uber, Dropbox, Google	Airbnb, Netflix, Spotify, Instagram

Tabela 1-Python e Java as diferenças

2.2.2 Keras vs Tensorflow vs Scikit-learn

Keras, Tensorflow e Scikit-learn são APIs/bibliotecas de código aberto, usadas principalmente para criar modelos de ML. Embora compartilhem alguns dos mesmos recursos, ainda existem algumas diferenças que pode ser significativo para o projeto em si. Neste projeto, o objetivo de utilizar uma dessas bibliotecas servirá para criar modelos de ML, sendo estes por exemplo o Naive Baeyes, Logistic Regression, Random forest, bem como redes neurais.

O Tensorflow é um framework especializada para aprendizagem máquina, com uma abordagem abrangente e com um ecossistema de ferramentas altamente flexível que fornece fluxos de trabalho com APIs de alto nível. Tensorflow promove também a construção e teste de modelos acessíveis com várias camadas de abstração. Isso é agnóstico de linguagem e plataforma, permitindo uma produção robusta de ML em qualquer lugar e um ótimo e ferramenta confiável quando se trata de pesquisa devido à sua fácil integração com outras bibliotecas como Keras para criar modelos mais complexos [11].

Scikit-learn é um framework amigável que contém uma grande variedade de ferramentas como classificação, regressão, modelos de agrupamento, pré-processamento, redução de dimensionalidade e avaliação do modelo. Ele contém a maioria dos algoritmos de aprendizado supervisionado conhecidos, como SVM, modelos lineares, métodos Bayesianos, mas para o escopo do projeto, também possui Árvores de Decisão. Ele também contém vários métodos para realizar a validação cruzada para verificar a precisão no ML modelos, o que é importante para avaliar os resultados de um modelo [12].

Keras é uma biblioteca de redes neurais de alto nível que funciona a cima do Tensorflow, CNTK e Theano. É usado principalmente para DP, permitindo prototipagem fácil e rápida, bem como funcionando perfeitamente na CPU e GPU. É muito fácil de usar, permitindo informações claras e acionáveis feedback para erros do usuário. Também é muito modular e escalável. Os modelos Keras podem ser facilmente interconectados com poucas restrições usando blocos de construção configuráveis, que também podem ser feitos sob medida, tornando-os muito mais adequados para pesquisa [11].

Para concluir, podemos dizer com avanço que o sklearn será sem dúvida o mais usado neste projeto pois os principais algoritmos que serão usados esta biblioteca faze-os muito facilmente sem muito “overhead”, e o “keras” será somente usado para as redes neuronais.

2.2.3 Numpy vs Pandas

O NumPy e o Pandas são bibliotecas de python com código aberto usadas principalmente para análise de dados. Ambos contêm ferramentas poderosas para exploração de dados, limpeza de dados e análise.

O Pandas, em particular, permite ao utilizador manipular os dados em todos os tipos de estruturas. Além disso, a maioria das bibliotecas de Machine Learning giram em torno dos DataFrames do pandas, como entradas para os modelos [13].

NumPy é usado principalmente por seu suporte a arrays N-dimensionais, que são muito mais robustos em comparação com as listas do Python. Esta biblioteca também se destaca pela sua rapidez visto que esta se encontra escrita em C, daí a sua enorme vantagem em comparação as listas de base do python. Esta ferramenta se integra muito bem com outras bibliotecas, como Tensorflow, com a sua computação interna em tensores. Um dos grandes recursos do NumPy é que ele usa cálculos orientados a matriz para que, ao trabalhar com várias classes, torna-se mais fácil [13].

Concluindo de maneira geral, tanto o Pandas quanto o NumPy, possuem as ferramentas necessárias para o tratamento de dados, e ao longo deste projeto ambas serão usadas, devido a sua enorme eficiência e simplicidade.

2.2.4 FastAPI

O FastAPI é uma framework moderna muito similar ao FLASK, porém com mais algumas vantagens, isto é, conforme o nome indica esta tem como objetivo a rapidez com que lida com os requests. Esta serve para criar web frameworks, ou seja, API de maneira fácil e rápida, pois também apresenta uma curva de aprendizagem bastante rápida.

Esta foi desenhada com os seguintes princípios em mente:

- **Rapidez:** Desempenho muito alto, a par com NodeJS e Go (graças a Starlette e Pydantic). Um dos frameworks Python mais rápidos disponíveis.
- **Rápido para programar:** Aumente a velocidade de desenvolvimento de recursos em cerca de 200% a 300%.
- **Menor número de bugs:** Reduza cerca de 40% dos erros induzidos por humanos.
- **Intuitivo:** Menos tempo a dar debug.
- **Fácil:** Projetado para ser fácil de usar e aprender. Menos tempo para ler documentos.
- **Curto:** Minimiza a duplicação de código. Vários recursos de cada declaração de parâmetro.
- **Robusto:** Desenvolvimento de código pronto para produção. Com documentação interativa automática.
- **Baseado em padrões:** Baseado em (e totalmente compatível com) os padrões abertos para APIs: OpenAPI (anteriormente conhecido como Swagger) e JSON Schema.

2.2.5 Docker

O docker é um conjunto de produtos de plataformas como serviço (PaaS) que com pouco acoplamento permite criar uma máquina com qualquer sistema operativo ou outro tipo de software de maneira muito fácil. A estas instâncias dá-se o nome de containers. Depois destas instâncias estarem criadas procede-se ao upload da aplicação em questão API ou outro tipo de serviço. Isto pode ser feito automaticamente no ficheiro DockerFile, fazendo com que este método possa ser distribuído da maneira mais simples possível.

Uma das grandes vantagens do docker, é completa remoção da famosa expressão “funciona no meu”, assim se funciona em um container, irá funcionar em todos os computadores que tenham docker, pois o sistema operativo dentro daquela instância será igual para todos, sem nenhuma diferença.

Outra vantagem, é que como obtemos uma instância, ou seja, um container, podemos simplesmente dizer que queremos mais 4 instâncias iguais, e assim de maneira muito fácil a nossa aplicação passa a ser 4 vezes mais rápida a atender os serviços, é claro que se todas as instâncias dependerem todas da mesma Base de dados se calhar não haverá muita performance [14].

Para além das várias vantagens do docker é importante lembrar que estes containers, são altamente seguros pois mesmo que estes se encontrem vulneráveis, não irão afetar a máquina principal.

3 Ambiente de Trabalho

Este capítulo irá ser focado nas metodologias de trabalho utilizadas, dando uma breve explicação das mesmas, bem como as tecnologias de controle de versão utilizadas.

3.1 Metodologias de trabalho

Nos dias de hoje, os projetos são organizados por metodologias ágeis de maneira a providenciar uma organização dos projetos. Isso é útil para padronizar e organizar os métodos de trabalho. Utilizando metodologias de trabalho adequadas nos mais diversos projetos, pode-se realmente beneficiar, alcançando assim grande sucesso, mas também podem ocorrer certos erros caso as metodologias não sejam as mais adequadas para a determinada topologia. Uma das vantagens das metodologias de trabalho, é que caso o sucesso seja alcançado esta pode ser replicado em outros projetos, enquanto os erros são identificados e corrigidos aprendendo com eles. Isso resulta em um processo de melhoria contínua.

Outros benefícios do uso de metodologias de trabalho são [15]:

- Envolvimento e satisfação das partes interessadas;
- Transparência da evolução do projeto;
- Constante entrega de valor ao cliente;
- Alta previsibilidade de entregas;
- Controle do risco;
- Melhoria da qualidade, as revisões regulares permitem encontrar erros e corrigi-los precocemente, melhorando a qualidade geral.

3.1.1 CRISP-DM

O “Cross Industry Standard Process for Data Mining (CRISP-DM)” é um modelo de processo que serve como base para um processo de ciência de dados. Possui seis fases sequenciais:

1. Entendimento do negócio – O que o negócio precisa?
2. Compreensão dos dados – Que dados temos/precisamos? Está limpo?
3. Preparação de dados – Como organizamos os dados para modelagem?
4. Modelagem – Quais técnicas de modelagem devemos aplicar?
5. Avaliação – Qual modelo melhor atende aos objetivos do negócio?
6. Implantação – Como as partes interessadas podem ver os resultados?

Esta técnica foi publicada em 1999 para padronizar os processos de mineração de dados em todos os setores, desde então tornou-se a metodologia mais comum para projetos de mineração de dados, análise e ciência de dados.

É importante lembrar que esta metodologia funciona por vezes em par com outras metodologias ágeis.

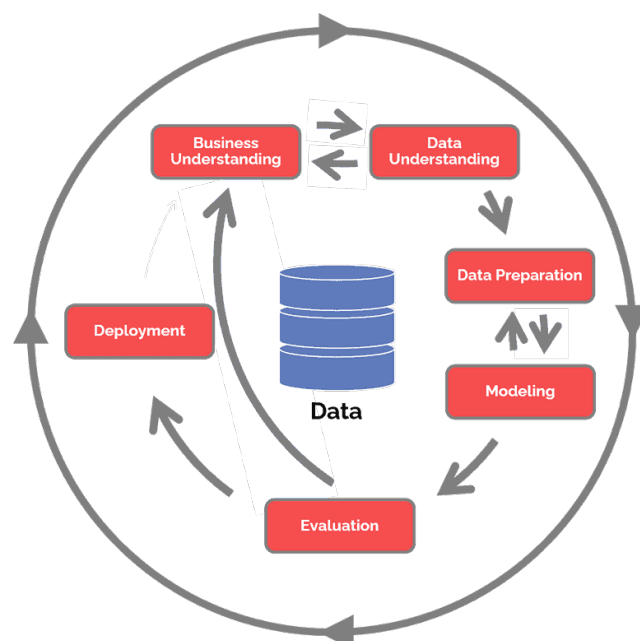


Figura 2- CRISP-DM diagrama

Primeiramente houve uma compreensão do modelo de negócios e dos requisitos do projeto. Foram analisados vários documentos relativamente ao estado da arte. Graças a estes documentos foi possível ter uma ideia de que metodologias foram usadas, bibliotecas, frameworks, e até mesmo alguns datasets analisados.

Depois de ver bastantes datasets, e em conjunto com o cliente/orientador foi escolhido um determinado dataset para a resolução do problema. Porém agora depois de já se ter escolhidos surge o problema que o dataset não se encontra limpo, bem como balanceado, por isso procederemos às respetivas medidas para corrigir tais problemas. Estas medidas não serão discutidas nesta secção, mas sim na próxima secção com mais detalhe.

Relativamente ao ponto 3, seguiremos então à preparação dos dados para usar no modelo, isto é, já procedemos à limpeza dos dados, ou seja, os valores NA bem como tabelas sem valores foram limpas, e os dados já foram divididos em 70% para treino e 30% para testes.

Para o ponto 4, serão usados vários modelos de classificação, e algoritmos dos quais, Logistic Regression, Naive Baeys, Random Forest e até mesmo redes neuronais.

Para o ponto 5, procedemos à respetiva avaliação dos diversos modelos abordados e testados. Segundo os papers relativos ao tema de phishing para emails todos eles diziam que independentemente das técnicas usadas, o Random Forest seria aquele que iria sair com a maior percentagem de acerto, o que mais à frente iremos presenciar que é verdade.

Por último, é guardado o modelo que possui uma maior taxa de acerto bem como o maior f1-score, pois desta maneira a cada previsão que se faça não seja preciso compilar e treinar cada vez. Desta forma, torna-se muito mais eficiente para no final por este modelo em uma API e servir vários clientes em simultâneo de maneira assíncrona.

3.2 Versões de Controlo

O controlo de versões permite que as equipas de software rastreie as alterações no código, enquanto aprimoram a comunicação e a colaboração entre os membros da equipe. O controle de versão facilita uma maneira contínua e simples de desenvolver software em equipa.

Esta ferramenta é bastante útil especialmente quando se trabalha com várias pessoas pois deteta anomalias em versões, isto é, se os desenvolvedores programarem simultaneamente e criam alterações incompatíveis, o controle de versão identificará as áreas problemáticas para que os membros da equipe possam reverter rapidamente as alterações para uma versão anterior. Ajuda depois a comparar alterações ou até mesmo identificar quem cometeu o erro que resultou no problema por meio do histórico de revisões. Com sistemas de controle de versões, uma equipe de software pode resolver um problema antes de avançar em um projeto. Por meio de revisões de código, as equipes de software podem analisar versões anteriores para entender como uma solução evoluiu [16].

Apesar que este trabalho tenha sido individual, foram usados mecanismos de controlo de versões dos quais o GitHub, mais propriamente o repositório do GECAD.

4 Implementação da Solução

4.1 Descrição da implementação

Perante o que já foi visto anteriormente, a solução mais apropriada para desenvolvimento do algoritmo de detecção de phishing, envolverá o uso da linguagem Python, bem como a respetiva API esta também em Python que vai por sua vez consumir este algoritmo responsável por ter um modelo de classificação.

Primeiramente houve uma escolha no dataset, que de vários eles optou-se por um que já apresentava algumas features, facilitando assim o trabalho do algoritmo. Caso não se opte por um já com número, por um somente de texto, já teria que ser outra metodologia, isto é, provavelmente já teríamos que usar DP para tal.

Depois de escolhido o “dataset”, foi feito o respetivo tratamento de dados, desde a limpeza destes mesmo, bem como a limpeza dos valores nulos que o dataset continha. No final de já ter os dados tratados, foi dado plot com o matplotlib para ver como é que os dados estavam distribuídos e como se pode ver na imagem abaixo, o seguinte aconteceu.

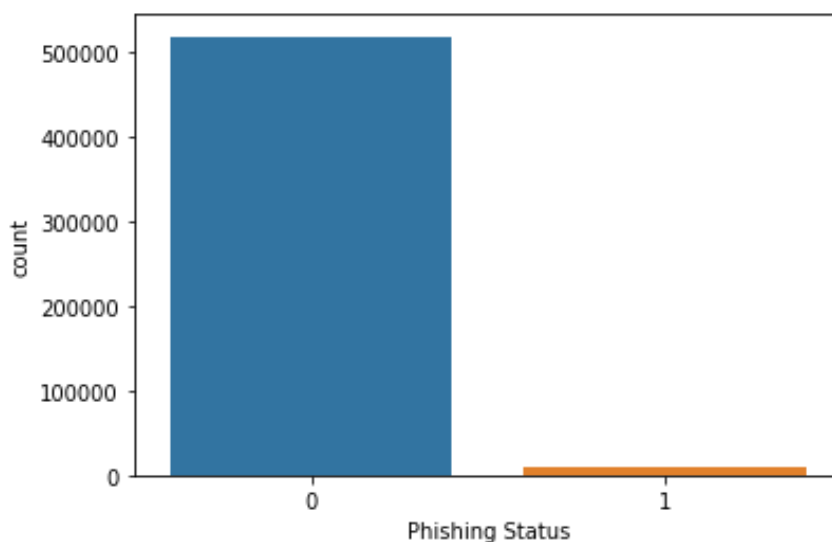


Figura 3-Dataset Plot

Como se pode observar, o “dataset” não está nada balanceado, pois a quantidade de email positivos em phishing é muito menor que emails normais, porém mesmo assim irão ser usado diversos algoritmos para ver a percentagem de acerto que nos retorna. Caso os resultados não sejam satisfatórios irá se proceder a outro tratamento, desde o undersampling dos dados relativamente aos 0, ou então ao oversampling com a técnica de SMOTE para os 1 (0 se for email verídico, 1 se for phishing).

De vários algoritmos que a ferramenta do sklearn nos fornece serão utilizados o Random Forest, Naive Baeyes e o Logistic Regression. Depois de dividir os dados em 70% de treino e 30% de teste, iremos analisar o comportamento para cada um destes algoritmos. É importante relembrar, que esta separação de 70/30 usa o parâmetro “stratify”, porque senão, os 70% aleatórios podia não apanhar nenhum phishing status de 1, então com o stratify é possível dividir em partes iguais, 70% de 0, e 70% de 1.

4.1.1 Resultados para o Random Forest

Relativamente ao Random Forest, foi obtido a seguinte informação depois de treinar durante alguns minutos:

	precision	recall	f1-score	support
0	0.99	1.00	1.00	155221
1	0.97	0.69	0.80	2506
accuracy			0.99	157727
macro avg	0.98	0.84	0.90	157727
weighted avg	0.99	0.99	0.99	157727

Figura 4-Random forest unbalanced data

Como podemos ver, o f1-score(macro) é aquele que nos dá o valor mais real do algoritmo, e este perante o dataset não balanceado apresenta 90% de taxa de acerto o que já não é mau, porém ainda pode melhorar, com técnicas que serão faladas mais à frente.

4.1.2 Resultados para o Naive Baeyes

Para o Naive Bayes, que é um modelo estatístico que usa como recurso probabilidades foi obtido:

	precision	recall	f1-score	support
0	0.99	0.97	0.98	155221
1	0.25	0.55	0.35	2506
accuracy			0.97	157727
macro avg	0.62	0.76	0.67	157727
weighted avg	0.98	0.97	0.97	157727

Figura 5 - Naïve Baeyes unbalanced data

Foi obtido de f1-score(macro) os 67% o que em si é muito pior que o random forest, o que em si faz sentido pois este método estatístico tende a ter uma maior taxa de sucesso quando este é treinado usando texto e não números.

4.1.3 Resultados para o Logistic Regression

	precision	recall	f1-score	support
0	0.99	1.00	0.99	155221
1	0.77	0.31	0.44	2506
accuracy			0.99	157727
macro avg	0.88	0.65	0.72	157727
weighted avg	0.99	0.99	0.98	157727

Figura 6-Logistic Regression unbalanced data

Foi obtido 72 % de taxa de acerto, o que também é muito inferior aos resultados do Random forest.

4.2 Análise dos resultados e melhoramentos

4.2.1 Undersampling

Antes desta conclusão prática, já tinha ideia de que os resultados seriam assim, pois vários papers, papers estes já analisados e descritos acima no estado da arte, diziam que o random forest era o algoritmo que para esta tarefa apresentaria resultados superiores para este tipo de tarefa, conclusão a que também foi chegada, ver tabelas acima.

Porém perante estes valores, irão ser usadas técnicas para subir um pouco mais as percentagens. Uma dessas técnicas é o undersampling, bem como o oversampling. Começemos primeiro com o undersampling, que é a mais simples pois consiste simplesmente em dividir os dados do dataset não balanceado de maneira equitativa, mantendo todos os dados na classe minoritária e diminuindo o tamanho da classe majoritária. Isto é possível usando o método do pandas sample (), que permitirá selecionar de um dataset um número de amostras, passando por parâmetro o tamanho dos valores de minoria, para os dados da maioria. Assim ficamos com ambas as partes equilibradas. Com isto se usarmos novamente o random forest veremos que este sobe para os 95% como se pode ver na figura abaixo.

	precision	recall	f1-score	support
0	0.95	0.95	0.95	2506
1	0.95	0.95	0.95	2506
accuracy			0.95	5012
macro avg	0.95	0.95	0.95	5012
weighted avg	0.95	0.95	0.95	5012

Figura 7 - Random Forest balanced data (undersampling)

4.2.2 Hyper Tuning

Relativamente ainda ao undersampling podemos fazer algumas experiências, sendo uma das técnicas mais usadas o tuning dos hiperparâmetros. Esta, consiste na seleção de alguns parâmetros que à priori consideramos que serão bons, e depois irá tentar com diversas combinações desses hiperparâmetros até depois retornar no fim aquele que será mais ótimo. Visto isto, procedemos à parte prática do mesmo, em que iremos pegar nos valores de undersampling e passar por uma lista de hiperparâmetros e no fim veremos a diferença. É de lembrar que este método é um método exaustivo e que pode demorar bastante tempo devido à enorme capacidade computacional que precisa. Para ainda otimizar mais, será usado o RandomSearchCV isto porque este irá substituir a enumeração exaustiva de todas as possíveis combinações selecionando-as aleatoriamente. É importante ter em conta que este método usa instâncias em paralelo para ainda agilizar ainda mais o processo.

Para este problema, o RandomSearchCV, irá treinar 1296 instâncias do Random Forest e no final retornar aquela que for melhor. Graças a este treino extensivo, que demorou mais de uma hora, a percentagem de acerto usando o f1-score(macro) como medida subiu de 95% para 99%, porém para isto ser possível, teve que ser treinado usando uma pequena parte dos dados, pois se fossem usados os dados na totalidade provavelmente estaríamos a falar de umas boas horas de treino. Porém veja-se a imagem abaixo que comprava o resultado.

```

from pprint import pprint
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor

# Parameters currently in use
# pprint(randomForest.get_params())

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 500, num = 3)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [10,20,30]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}

print(random_grid)

rf_Model = RandomForestClassifier()
rf_Grid = GridSearchCV(estimator=rf_Model, param_grid=random_grid, cv=3, verbose=10, n_jobs=4)
rf_Grid.fit(X_test,y_test)

print(rf_Grid.best_params_)

#true_result = test_Y
#pred_result = ran_pred

#print(classification_report(true_result, pred_result))

{'n_estimators': [200, 350, 500], 'max_features': ['auto', 'sqrt'], 'max_depth': [10, 20, 30, None], 'min_samples_split': [2, 5, 10], 'min_
samples_leaf': [1, 2, 4], 'bootstrap': [True, False]}
Fitting 3 folds for each of 432 candidates, totalling 1296 fits
{'bootstrap': True, 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 350}

```

Figura 8 - Random Forest Tuning depois de undersampling

Nesta imagem podemos ver desde o início a importação da biblioteca do sklearn, que será encarregue de fazer todo o trabalho pesado que será preciso. Depois é definido o número de estimativas, que neste caso decidiu-se 200,350,500. O número máximo de features a cada corte, bem como o número de níveis da árvore. No máximo definiu-se 30 pois estamos a trabalhar com uma pequena parte dos dados, senão teríamos que usar um número bem maior caso fosse para usar na totalidade do dataset. Novamente foram definidos números pequenos para o número mínimo de amostras

necessárias para dividir um nó bem como cada folha da árvore. Depois de todos os parâmetros serem carregados para um dicionário, este é passado por parâmetro, com `verbose=10`, para ver cada iteração e 4 threads para correr em simultâneo.

```
ran_pred=rf_Grid.predict(X_test)
acc = accuracy_score(ran_pred,y_test)

true_result = y_test
pred_result = ran_pred

print(classification_report(true_result, pred_result))
```

```
0.9931158335827597
              precision    recall  f1-score   support

     0       0.99         1.00         0.99         1671
     1       1.00         0.99         0.99         1670

 accuracy                   0.99         3341
 macro avg       0.99         0.99         0.99         3341
weighted avg       0.99         0.99         0.99         3341
```

Figura 9 - Resultado depois do treino

Por fim, depois do treino, foram obtidos os melhores parâmetros, agora prossegue-se ao seu uso no Random Forest para ver o quanto este irá melhorar, conforme se pode confirmar na imagem a seguir.

Conforme podemos ver, foi obtido um resultado de 99%, porém temos somente 3341 amostras o que é muito pouco, assim em vez de usar este modelo para incorporar na API iremos tentar outras técnicas primeiro que nos garantam aproximadamente os 99%, mas que usem de base mais amostras.

4.2.3 Oversampling

Perante estes valores já se conseguiu subir de 90% para 99%, o que já é bastante bom, porém ainda existem mais técnicas que irão possibilitar melhores resultados, que será o caso do oversampling como será demonstrado agora.

Procedemos então à próxima técnica, denominada de oversampling. Dentro do oversampling existe muitas técnicas algumas delas consistindo somente na repetição da classe minorante, porém uma estratégia mais eficaz é na criação de novas amostras. A esta técnica dá-se o nome de SMOTE (Synthetic Minority Oversampling Technique) consiste na criação de amostras na classe minorante de maneira sintética a partir de exemplos existentes [17]. Para isto usa de maneira auxiliar o algoritmo KNN para tal.

Passando então à parte prática, usaremos o SMOTE para aumentar a classe minorante (classe do 1 representante dos emails de phishing) em pelo menos 35 % pois a partir daí começamos a ter mais valores sintéticos do que valores reais do dataset, como se pode ver pela imagem seguir:

```
smote = SMOTE(random_state=42, sampling_strategy=0.35)
X_sm, y_sm = smote.fit_resample(train_X, train_Y)
```

Figura 10 - SMOTE data

Depois de usar SMOTE o nosso dataset já se encontra mais balanceado, logo o random forest irá ter melhores resultados. Na imagem abaixo podemos ver uma visualização do dataset mais balanceado, ainda não se encontra 50% porque também não convém dar ao modelo o mesmo número de amostras de phishing visto que no mundo real a maioria dos emails não são de phishing.

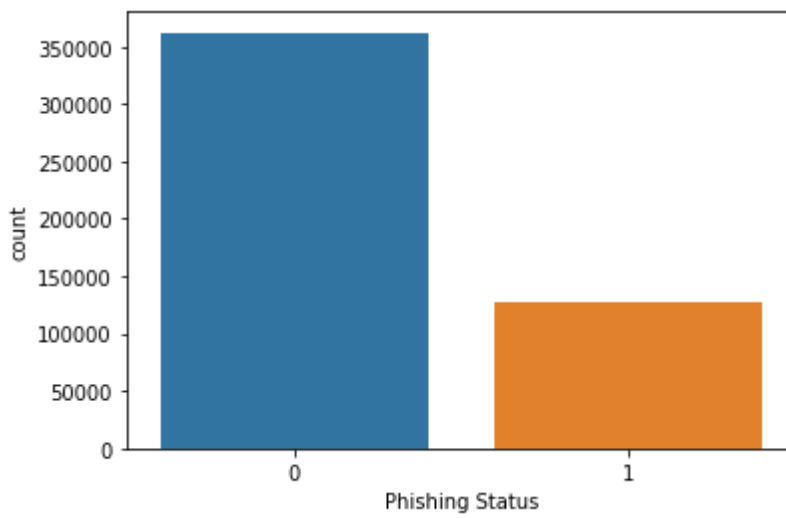


Figura 11 - Visualização do dataset depois do SMOTE

Agora com os valores mais balanceados, procedemos então novamente a utilização do random forest na expectativa de aumentar as percentagens de acerto. Na imagem abaixo podemos concluir que de facto o modelo será mais preciso e com maior taxa de acerto.

	precision	recall	f1-score	support
0	0.99	0.99	0.99	72436
1	0.99	0.98	0.98	25353
accuracy			0.99	97789
macro avg	0.99	0.99	0.99	97789
weighted avg	0.99	0.99	0.99	97789

Figura 12 - Visualização do dataset depois do SMOTE

4.2.4 Desenvolvimento da API

Como o objetivo deste trabalho era poder disponibilizar um algoritmo que dissesse se um determinado email era ou não relacionado com phishing, decidi fazer uma API bastante simples para atender a essa necessidade.

Para esta API decidi usar docker, fast-api uma biblioteca que permite criar API's com bastante performance bem como de forma rápida entre outras tecnologias.

Esta API devido à sua simplicidade irá ter uma só rota, uma rota de POST para o /email, onde o cliente manda em JSON o respetivo email que pretende analisar. Esta rota irá guardar o email numa variável e esta irá passar por vários métodos que procederão à extração das features exatamente iguais as features que o dataset originalmente disponibilizava. Estas features podem-se dividir em 2 grupos. Um grupo em que verifica a presença de certas palavras que à priori são suspeitas, sendo estas as seguintes: account, acess, bank, credit, click, identity, inconvenience, information, limited, log, minutes, password, recently, risk, social, security, service, suspende. E o outro grupo, que analisa a quantidade de palavras, de caracteres, palavras únicas, a riqueza do vocabulário, bem como os conetores das frases, para verificar se o texto está bem ou mal escrito.

Tome-se a imagem abaixo como prova que o conteúdo do email está a ser analisado e partido nos diversos tokens que foram falados aqui acima.

```
def email_analysis(text):  
    array_to_convert = []  
  
    lower_text = to_lower_case(text)  
  
    n_char = count_character(text)  
    array_to_convert.append(n_char)  
  
    voc_richness = vocabulary_richness(lower_text)  
    array_to_convert.append(voc_richness)  
  
    words = to_lower_case("Account,Access,Bank,Credit,Click,Identity,Inconvenience,Inf  
  
    for word in words.split(","):  
        counter = count_words(lower_text, word)  
        array_to_convert.append(counter)  
  
    fun_words = function_words(lower_text)  
    array_to_convert.append(fun_words)  
  
    unique_words = count_words_unique(lower_text)  
    array_to_convert.append(unique_words)  
  
    model = joblib.load("../fastapi-master/app/ML model/trainedForest.joblib")  
  
    is_spam = model.predict_proba(np.array([array_to_convert]))  
    print(is_spam)  
  
    print(array_to_convert)
```

Figura 13 - Código responsável pela análise do email

Este método irá pegar nos diversos tokens e colocá-los num array pela mesma ordem que o dataset os disponibilizou. Primeiro valor a colocar é o número de caracteres, depois a riqueza do vocabulário (importante lembrar que só funciona para emails da língua inglesa, pois a biblioteca usa um dicionário inglês para analisar a semântica das palavras), e depois no ciclo for, a quantidade de vezes que as palavras acima descritas irão aparecer. Depois irão também as functions words, que é em português podemos chamar de conetores, bem como as palavras únicas.

Depois de obter todos estes tokens o modelo é lido da memória, pois este já foi treinado e guardado em um ficheiro binário para aumentar a rapidez da API, e assim poder dar-nos uma previsão instantânea se o email será ou não malicioso e a sua respetiva percentagem.

4.2.5 Rotas da API

Conforme já foi falado acima esta API será algo bastante simples tendo somente uma rota que será usada /email, e uma rota /health para verificar se o sistema está em baixo ou se este está a demorar mais do que é suposto.

Tome-se a seguinte imagem como prova. Esta foi construída através da API embutida no FastAPI denominada Swagger.



default

GET	/health/	Get Health Status
POST	/email/	Post Email

Figura 14 - Rotas da API

O método a azul será um simples GET que já foi falado acima. Por outro lado, o método a verde será um POST onde qualquer pessoa poderá fazer o “request” com o email que acha ou não suspeito de phishing. É muito fácil mandar um request para esta rota, visto que esta só leva um parâmetro que é o email. Veja-se a imagem seguinte.

POST /email/ Post Email	
Parameters	
No parameters	
Request body <small>required</small>	
Example Value Schema	
<pre>{ "email": "string" }</pre>	
Responses	
Code	Description
200	Successful Response

Figura 15 - Conteúdo Json para a rota /email

Em suma, isto irá resumir a funcionalidade desta API, assim fica-nos a sobrar como disponibilizar esta de maneira eficiente e de maneira segura para qualquer colaborador.

Para atender a este problema temos duas opções ou instalamos no computador de cada um uma estância desta mesma, ou então a solução mais simples, um servidor geral que possa atender aos requestes de qualquer pessoa. Esta segunda opção é sem dúvida mais prática, porém em casos de produção podia resultar em alguns custos.

Por último, para instalar esta estância de maneira simplificada em um servidor que pode ter um sistema operativo diferente deste computador utilizaremos o docker, pois este consegue encapsular o código em um ambiente seguro e de funcionamento confiável. Isto porque em ambientes docker se funciona em um irá funcionar para todos. Tome-se a imagem abaixo que ilustra a configuração do docker para que o código possa rodar em qualquer máquina independente da arquitetura e dos pacotes previamente instalados.

```
FROM python:3.9 as builder
ENV DEBIAN_FRONTEND=noninteractive
ENV VIRTUAL_ENV=/venv
WORKDIR /source

RUN apt-get update && \
    apt-get --no-install-recommends install -y gettext locales-all tzdata build-essent

RUN python -m venv $VIRTUAL_ENV
ENV PATH=$VIRTUAL_ENV/bin:$PATH
COPY ./requirements.txt /source/
RUN pip install --no-cache-dir --upgrade -r /source/requirements.txt

FROM python:3.9-slim-bullseye as runner
ENV PYTHONUNBUFFERED 1
ENV USERNAME fa
ENV VIRTUAL_ENV=/venv
WORKDIR /source

COPY --from=builder $VIRTUAL_ENV $VIRTUAL_ENV
ENV PATH=$VIRTUAL_ENV/bin:$PATH

RUN adduser --disabled-password --gecos "" $USERNAME && id $USERNAME
RUN echo export PYTHON_PATH=/usr/local/bin/python >> /home/$USERNAME/.profile

COPY ./app /source/app

COPY ./entrypoint.sh /entrypoint.sh
CMD /entrypoint.sh
```

Figura 16 - Docker File para construção da imagem

Este ambiente virtualizado será construído no diretório em que se encontra e irá ter a porta 8000 como aberta, estando as restantes fechadas relativas a fatores de segurança. Quanto ao resto irá conter uma versão de python 3.9 bastante estável e irá instalar os diversos pacotes pois a imagem de base do sistema operativo é nada mais que uma folha em branca, por isso temos que instalar todos os pacotes para o bom funcionamento. Os requerimentos são instalados para a API, as diversas variáveis de ambiente serão exportadas e por fim irá executar o entrypoint.sh que é nada mais do que um “server start” com o auxílio do unicórnio para disponibilizar a rapidez visto que este é um server com paralelismo, opção que o python nativo não suporta totalmente.

4.3 Testes

Como estamos perante um domínio de inteligência artificial não existe necessidade de fazer testes, especialmente teste unitário, integração, entre outros visto que o domínio não o permite nem faz sentido.

Porém, neste ramo usa-se a palavra testes para dividir o dataset em uma parte para testar depois de treinar para averiguar se o modelo é bom ou não. Ao longo deste trabalho a testagem do dataset ou das variáveis contendo arrays de informação foram divididos em 30%, máximo 35% quando se fez o SMOTE.

Com esta testagem foi possível prever estatisticamente as percentagens de acerto para dados que o algoritmo ainda não tinha visto, dando ao algoritmo pela primeira vez nova informação fazendo com que este avalie a situação e dê uma previsão. Esta previsão para os restantes valores de testes irá gerar uma percentagem do acerto viável para o exterior, apesar que temos que ter sempre em conta que num ambiente externo a performance do algoritmo pode sempre ou não baixar.

4.4 Avaliação da solução

Como o principal objetivo desde o início era o desenvolvimento de um algoritmo que soubesse distinguir emails normais de emails de phishing, e é com segurança que afirmo dizer que este foi cumprido. É claro que como em qualquer programa, este podia ter ficado ainda melhor, isto é, foi analisado num dataset de 500000 entradas, se calhar se fosse num dataset 10 vezes maior, teríamos resultados ainda mais precisos. Também as técnicas que foram usadas foram todas à base de machine learning, e para este caso o uso de técnicas de Deep Learning talvez fosse melhor, analisado a semântica das palavras entre outras metodologias, porém para isso seria preciso mais tempo, melhores máquinas visto que estes modelos funcionam principalmente com GPUS, bem como outro tipo de conhecimento mais fundo neste ramo, que até a data ainda não o tenho.

A nível de precisão o algoritmo obteve os 99% que era o que se pretendia no início deste projeto. Esta precisão só foi possível ser atingida usando o Random Forest bem como técnicas de balanceamento de dados, adição de dados sintéticos via SMOTE, bem como o ajuste dos hiperparâmetros.

Relativamente à usabilidade, esta já foi descrita, mas para qualquer pessoa pudesse verificar se o seu email será ou não suspeito foi feita uma API para responder a tais necessidades.

Por último, quanto ao desempenho, teve-se bastante cuidado, pois se é feito para ser usado por diversas pessoas, este tem que ser rápido e concorrente. Para tal foi usado o servidor HTTP bastante rápido e threaded(concorrente). Graças a este conseguimos fazer 50 request em um segundo todos estes em simultâneo ao servidor, e este analisa o texto passando para o modelo de machine learning pré-treinado e retornando a percentagem que o modelo vai dar de output. Quando às outras rotas que não usam as previsões do modelo este servidor encontra-se por volta dos 10 mil requests por segundo, valor pequeno tendo em conta outro tipo de API/linguagens, mas também um dos principais problemas sendo a própria linguagem Python.

5 Conclusões

Este último capítulo apresentará as conclusões sobre o trabalho realizado. Na primeira seção será realizada uma revisão dos objetivos, bem como um entendimento de quais deles foram alcançados. A segunda seção realiza uma abordagem ao tema das limitações e trabalhar. Por fim, haverá algumas considerações finais sobre o trabalho desenvolvido.

5.1 Objetivos concretizados

Objetivo	Foi cumprido	Como/Porque não
Estado da arte	Sim	Diversos papers relativos a phishing em email foram analisados, e as conclusões desses mesmos coincidem com os apresentados ao longo deste relatório.
Estudo de diversos datasets	Sim	Apesar de analisados vários datasets foi escolhido um que apresenta features em vez de puro texto.
Aprendizagem de modelos de machine learning	Sim	Maioria dos métodos de machine learning foram aprendidos e postos em prática.
Aprendizagem de controlo de big data	Sim	Com ajuda da internet, bem como de algumas cadeiras de análise de dados, foi obtido o conhecimento para

		fazer o devido tratamento de dados.
Uso de redes neurais	Sim, mas não na totalidade	Foi criado um modelo, porém este dava sempre percentagem de 73% e independente do que fosse feito este não mudava, provavelmente devido a algum erro a mim desconhecido.
Uso de vários algoritmos de machine learning para resolução do problema	Sim	Usado desde o Random Forest, Naive Baesys, Logistic Regression ...
Gerenciamento de dados não balanceados	Sim	Através de técnicas de undersampling e oversampling
Algoritmo com precisão elevada	Sim	No final, foi obtido um modelo com 99% precisão
Aprendizagem de docker	Sim	Como o projeto era relativamente simples o ficheiro docker também não foi muito difícil de o fazer
Aprendizagem de API's em Python	Sim	Como já sabia fazer usando Flask, a transição para FastAPI foi relativamente simples.
Desenvolvimento de API para fornecer algoritmo	Sim	Esta foi desenvolvida, e passou por um processo de testes de desempenho ao que suportava concorrentemente pelo

		menos 40 pessoas com poucos recursos.
Utilização de um servidor multi-thread para rápida distribuição.	Sim	Inicialmente usava-se o unicorn, porém este era muito lento para as necessidades e foi optado por um servidor multi-thread (gunicorn)

Tabela 2-Objetivos concretizados

5.2 Limitações e trabalho futuro

Como em todos os programas de software existirá sempre limitações, e apesar de que quase todos os objetivos terem sido cumpridos alguns destes ainda se deviam melhorar. Inicialmente gostaria de ter conseguido por as redes neuronais a funcionar direito como eu queria, porém devido a ainda não perceber muito bem na integra como é que estas funcionam não foi possível levar o modelo até ao fim, apesar que mesmo assim foi atingido os 99%. Outras das limitações a este algoritmo é o facto de só funcionar para língua inglesa, ou seja, se lhe for entregue um input de outra língua, este não irá saber tratar, pois o modelo nunca viu. É importante relembrar que o algoritmo usa um dicionário inglês para verificar a ortografia das palavras, daí neste momento só funcionar em uma só língua. Uma possível limitação, seria se este algoritmo começasse a ser disponibilizado para várias entidades e pessoas, pois neste momento este só consegue cerca de 50 pedidos por segundo o que é bastante pouco, porém conforme está feito e em container, se fosse preciso mais velocidades bastaria somente criar mais réplicas, pois da maneira que foi feita, temos um escalamento horizontal sem termos que alterar nada. Assim devido a topologia pensada, como estamos perante um escalonamento horizontal podia-se escalar teoricamente para infinitos requests por segundo.

Por outro lado, quando a trabalhos futuros, sem dúvida seria interessante em vez de usar um dataset somente com números, ou com poucos emails, fazer um dataset de emails, por exemplo um servidor de emails de uma empresa recebe ao longo de vários anos bastantes emails, sem dúvida existiria muitos de phishing apesar que a maioria destes seriam normais. Depois de este dataset pronto, em vez de tomarmos a rota do machine learning optaríamos pelo Deep Learning e desta forma podíamos usar novas técnicas com melhor precisão e sem dúvida mais indicadas para este tipo de problema, porém com isto implicaria conhecimentos de NLP entre outro tipos, sem falar nos custos pois seria preciso uma placa gráfica bastante rápida para fazer o respetivo tratamento e treino.

5.3 Apreciação final

Este projeto foi algo que sem dúvida não estava habituado, pois desenvolvimento de software na área de investigação é bastante diferente das metodologias até agora abordadas. Deste as linguagens de programação usadas, os padrões usados, bem como as metodologias aqui abordadas. No entanto sinto que foi uma experiência enriquecedora e muito diferente até agora habituado, porém acho que este tema em específico fosse demasiado simples, e talvez podia ter dado mais de mim, ou até querer fazer mais alguma coisa, mas com este tema sinto que não havia muita expansão.

De um modo geral sinto que o principal objetivo deste projeto foi alcançado, desde obter um algoritmo com uma elevada taxa de acerto e que pudesse prever futuros emails de phishing até à respetiva distribuição deste mesmo para os mais diversos utilizadores via API. Porém ainda estaria longe de algo para ser libertado para o mercado, pois conforme disse em cima, seria preciso muitos mais dados e outros tipos de técnicas para chegar ao nível de perfeito.

6 Referências

- [1] “GECAD: Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development.” <http://www.gecad.isep.ipp.pt/GECAD/Pages/Presentation/Home.aspx>.
- [2] “Machine Learning Techniques used in Cybersecurity” <https://www.securityscientist.net/current-machine-learning-techniques-used-in-cybersecurity/>
- [3] “7 Types of Cyber Security Threats.” <https://onlinedegrees.und.edu/blog/types-ofcyber-security-threats/>
- [4] “Cybersecurity Framework | NIST.” <https://www.nist.gov/cyberframework>
- [5] “5 Functions of the NIST Cybersecurity Framework.” <https://www.forescout.com/blog/how-to-comply-with-the-5-functions-of-the-nist-cybersecurity-framework/>
- [6] “Artificial intelligence-based antivirus in order to detect malware preventively” <https://link.springer.com/article/10.1007/s13748-020-00220-4>
- [7] “Hong, J.: The state of phishing attacks. Commun. ACM 55(1),74–81 (2012) “
- [8] “<https://www.statista.com/statistics/1253213/employees-phishing-emails-back-to-work-uk-us/>”
- [9] “Vinayakumar, R., Soman, K., Poornachandran, P., Akarsh, S., Elhoseny, M.: Deep learning framework for cyber threat situational awareness based on email and URL data analysis. In: Hassanien, A.E., Elhoseny, M. (eds.) Cybersecurity and Secure Information Systems, pp. 87–124. Springer, New York (2019)”

[10] “Python Advantages and Disadvantages - Step in the right direction - TechVidvan.”
<https://techvidvan.com/tutorials/python-advantages-and-disadvantages/>

[11] “TensorFlow vs Keras: Which One Should You Choose.”
<https://analyticsindiamag.com/tensorflow-vs-keras-which-one-should-you-choose/>

[12] “SKLearn | Scikit-Learn In Python | SciKit Learn Tutorial.”
<https://www.analyticsvidhya.com/blog/2015/01/scikit-learn-python-machinelearning-tool/>

[13] “Pandas vs NumPy - javatpoint.” <https://www.javatpoint.com/pandas-vs-numpy>

[14] “What is Docker” – <https://www.redhat.com/pt-br/topics/containers/what-is-docker>

[15] “Benefícios da Metodologia Agile” -
<https://infoportugal.pt/2021/09/02/metodologia-agile/>

[16] “What is version control?” - <https://about.gitlab.com/topics/version-control>

[17] “SMOTE - Towards Data Science” - <https://towardsdatascience.com/smote-fdce2f605729>