

«Título do relatório»

«Nome da organização»

20XX / 20XX

«Número(s)» «nome(s) do(s) aluno(s)»



«Título do relatório»

«Nome da organização»

200X / 200X

«Número(s)» «nome(s) do(s) aluno(s)»



Licenciatura em Engenharia Informática

«Mês e Ano de apresentação»

Orientador ISEP: **«nome(s)»**

Supervisor Externo: **«nome(s)»**

«*Dedicatória*»

Agradecimentos

Nesta secção (opcional) colocam-se notas de agradecimento às pessoas que contribuíram para a realização da tarefa. No caso de terem usufruído de financiamento via bolsa ou qualquer outro mecanismo formal, deverá ser incluído um agradecimento.

Resumo

O resumo do relatório (que só deve ser escrito após o texto principal do relatório estar completo) é uma apresentação abreviada e precisa do trabalho, sem acrescento de interpretação ou crítica, escrita de forma impessoal, podendo ter, por exemplo, as seguintes três partes:

1. Um parágrafo inicial de introdução do contexto e do problema/objetivo do trabalho.
2. Resumo dos aspetos mais importantes do trabalho descrito no presente relatório, que por sua vez documenta abordagem adotada e sistematiza os aspetos relevantes do trabalho realizado durante o estágio. Deve mencionar tudo o que foi feito, por isso deve concentrar-se no que é realmente importante e ajudar o leitor a decidir se quer ou não consultar o restante do relatório.
3. Um parágrafo final com as conclusões do trabalho realizado.

**Palavras-chave (Tema):** Incluir 3 a 6 palavras/expressões chave que caraterizem o projeto do ponto de vista de tema/área de intervenção.

**Palavras-chave (Tecnologias):** Incluir 3 a 6 palavras/expressões chave que caraterizem o projeto do ponto de vista de tecnologias utilizadas.

**(O Resumo só deve ocupar 1 página, cerca de 20 linhas**)

Índice

[1 Introdução 1](#_Toc5805924)

[1.1 Enquadramento/Contexto 1](#_Toc5805925)

[1.2 Descrição do Problema 1](#_Toc5805926)

[1.3 Estrutura do relatório 2](#_Toc5805927)

[2 Estado da arte 3](#_Toc5805928)

[2.1 Trabalhos relacionados 3](#_Toc5805929)

[2.2 Tecnologias existentes 3](#_Toc5805930)

[3 Análise e desenho da solução 5](#_Toc5805931)

[3.1 Domínio do problema 5](#_Toc5805932)

[3.2 Requisitos funcionais e não funcionais 5](#_Toc5805933)

[3.3 Desenho 5](#_Toc5805934)

[4 Implementação da Solução 7](#_Toc5805935)

[4.1 Descrição da implementação 7](#_Toc5805936)

[4.2 Testes 7](#_Toc5805937)

[4.3 Avaliação da solução 7](#_Toc5805938)

[5 Conclusões 9](#_Toc5805939)

[5.1 Objetivos concretizados 9](#_Toc5805940)

[5.2 Limitações e trabalho futuro 9](#_Toc5805941)

[5.3 Apreciação final 9](#_Toc5805942)

[Referências 11](#_Toc5805943)

[Anexo A Conteúdo em anexos 13](#_Toc5805944)

[Anexo B Regras de Conteúdo e Estrutura 14](#_Toc5805945)

[B.1 Linguagem 14](#_Toc5805946)

[B.2 Formatação 15](#_Toc5805947)

[B.3 Imagens e tabelas 15](#_Toc5805948)

[B.4 Referências 17](#_Toc5805949)

[Anexo C Plágio 19](#_Toc5805950)

Índice de Figuras

Figura 1 - Exemplo de imagens a) difícil leitura; b) fácil leitura (Sousa 2002) 16

Figura 2 - Exemplo de lista de Referências (Sousa, 2002) 18

Índice de Tabelas

Tabela 1 - Exemplo de tabela 16

Notação e Glossário

**Dos** Denial of Service

**SQL** Denial of Service

**MITM** Man in the Middle

**IoT** Internet of Things

**RF** Random Forest

**MLP** Multi-Layer Perceptron

**LSTM** Long-Short Term Memory

**ANN** Artificial Neural Networks

**KNN** K-Nearest Neighbors

**SVM** Support Vector Machines

**API** Application Programming Interface

**DP** Deep Learning

**KNN** K-Nearest Neighbors

**SMOTE** Synthetic Minority Oversampling Technique

1. Introdução

**Nota: Este guia de elaboração de relatório é apenas um guia não substituindo o diálogo necessário entre estudante e orientador para a melhor definição da estrutura e conteúdo do relatório em cada caso concreto.**

A introdução deve dar ao leitor a informação básica necessária por forma a facilitar o enquadramento dos objetivos, da abordagem seguida e dos resultados. A introdução começa com uma perspetiva geral do problema em estudo (na secção de enquadramento) em que, à medida que vai progredindo, deve ir fornecendo informação mais específica até se abordar a área em concreto tratada no relatório. Deve descrever, de forma sucinta, o problema em estudo e os objetivos. Deve também enunciar os principais métodos utilizados no trabalho, bem como a identificação clara dos contributos e aspetos inovadores da solução. A introdução deve terminar com a apresentação sucinta das secções que fazem parte do resto do relatório.

Nota: deve usar frases curtas; adotar o impessoal em vez do pessoal (e.g. adotou-se vs. adotei vs. adotamos); usar verbos conjugados no presente; evitar encadear verbos seguidos (e.g. “esta secção vai ser descrito” vs. “esta secção descreve”); usar voz passiva vs. ativa (ver Anexos).

* 1. Enquadramento/Contexto

Enquadrar o problema no âmbito do projeto da organização proponente e descrever a motivação do estudante em aceitar este desafio. No caso de se tratar de um projeto proposto pelo estudante (sem estágio), descrever a motivação pessoal.

* 1. Descrição do Problema

O problema deve ser descrito de forma clara, de forma a que o leitor compreenda facilmente.

* + 1. Objetivos

Os objetivos devem ser descritos de forma detalhada devendo refletir a compreensão do estudante sobre o trabalho que foi desenvolvido.

* + 1. Abordagem

A aproximação preconizada para a solução do problema ou do tratamento do tema focado, onde estejam claras as contribuições previstas. O objetivo é identificar a abordagem adotada (usando referência bibliográfica) e como foi aplicada ao projeto.

* + 1. Contributos

Devem ser apresentados os aspetos inovadores e de realce do trabalho, bem como a identificação dos benefícios trazidos para a organização (para a sociedade, para o ambiente,…). Devem ser identificadas as contribuições previstas (podendo usar estilo de apresentação por itens).

* + 1. Planeamento do trabalho

Nesta subsecção deve ser apresentado o planeamento (cronograma, etc.) definido para a execução do projeto com indicação das tarefas, datas e principais fases (*milestones*).

**(poderá ir para anexo)**

* 1. Estrutura do relatório

Apresentação sucinta dos capítulos que fazem parte do relatório, descrevendo em poucos parágrafos o que cada um deles trata.

Para além da introdução, esta dissertação contém mais x capítulos. No capítulo 2, é descrito o estado da arte e são apresentados trabalhos relacionados. No capítulo 3, …

**(Este capítulo deverá ter no máximo 6 páginas**)

1. Estado da arte

Neste capítulo, irá ser demostrado 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.

* 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 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. Como foi possível ver acima, estes podem ser originados por múltiplos atores, dos quais podem variar desde hackers, a terroristas, espiões ou até mesmo de funcionários. De acordo com [1], 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 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 [2].
* 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.



Figure 1 – NIST cybersecurity framework pillars [3]

* + 1. Inteligência artificial aplicada à segurança

Segundo uma recente literatura, alguns estudos nesta área, certos métodos de inteligência artificial, dos quais, Random Forests (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 deteção dos mesmos. Visto que na área de segurança existe uma enorme dinâmica relativamente aos ataques, faz sentido usar métodos não linear, 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 deteção de vírus, sendo que as redes neuronais foram as que prevaleceram com a maior percentagem de acerto [4].

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 [4]. 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 relembrar que os 99% é refletido nos ficheiros de input, pois esta percentagem iria descer de ano para ano, visto que a milhões de vírus a ser 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 deteção 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 [5], 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 se está comunicando com uma entidade confiável e tenta enganar para lhe ser fornecido credenciais de email ou 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 [6]. 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 tais potências fraudes, de tal modo que os funcionários já não corriam o risco de abrir pois o algoritmo chegava e 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 como uma das técnicas mais eficientes da Aprendendo Automática [7].

* 1. Tecnologias existentes

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

* + 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 complementado 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 [8]. 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.

ss
 Tabela 1- Python vs Java

* + 1. Keras vs Tenserflow 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 é criar dois modelos de ML, sendo o primeiro um modelo de classificação usando o Naive Baeys, Logistic Regression, Random forest, bem como redes neuronais.

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 [9].

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 [10].

Keras é uma biblioteca de redes neurais de alto nível que roda em 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 [9].

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 “overthead”, e o “keras” será somente usado para as redes neuronais.

* + 1. 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 [11].

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 [11].

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.

* + 1. 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:

* Rápido: 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.
  + 1. 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 dependeram todas da mesma Base de dados se calhar não haverá muita performance [12].

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

1. 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.

* 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 [13]:

* Envolvimento e satisfação das partes interessadas;
* Transparência da evolução do projeto;
* Constante entrega de valor ao cliente;
* Alta previsibilidade de entregas;
* Controlo do risco;
* Melhoria da qualidade, as revisões regulares permitem encontrar erros e corrigi-los precocemente, melhorando a qualidade geral.
  + 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 relembrar que esta metodologia funciona por vezes em par com outras metodologias ágeis.

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.

* 1. Versões de Controlo

O controlo de versões permite que as equipes 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 programam simultaneamente e criam alterações incompatíveis, o controle de versão identifica 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 [14].

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

1. Implementação da Solução

* 1. Descrição da implementação

Perante o que já foi visto anteriormente, a solução mais apropriada para desenvolvimento do algoritmo de deteção 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.

Uma imagem com quadrado

Descrição gerada automaticamente

Figure 2 – Dataset Plot

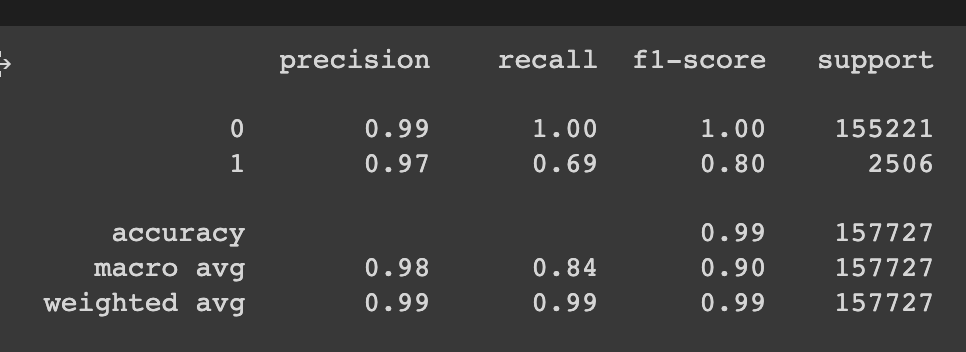
Como se pode observar, o “dataset” não está nada balanceado, pois a quantidade de email positivos em phishing é muito menor que emails normais, porém mesmo assim irão ser usado diversos algoritmos para ver a percentagem de acerto que nos retorna. Caso os resultados não sejam satisfatórios irá se proceder 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 Baeys 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.

* + 1. Resultados para o Random Forest

Figure 3 – Random forest unbalanced data

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



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

* + 1. Resultados para o Naive Baeys

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

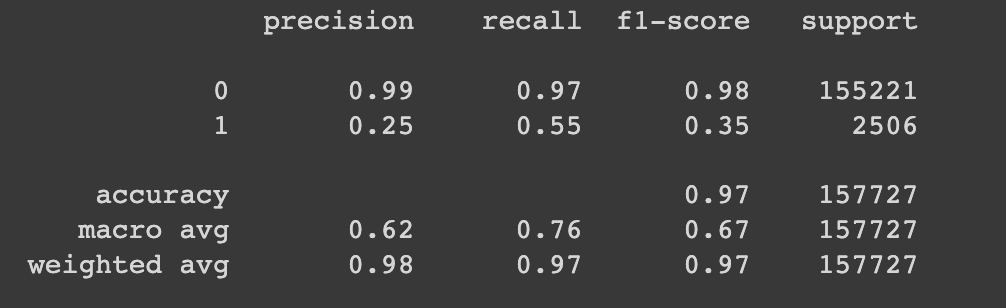


Figure 4 – Naïve Baeys unbalanced data

Foi obtido de f1-score(macro) os 67% o que em si é muito pior que o random forest.

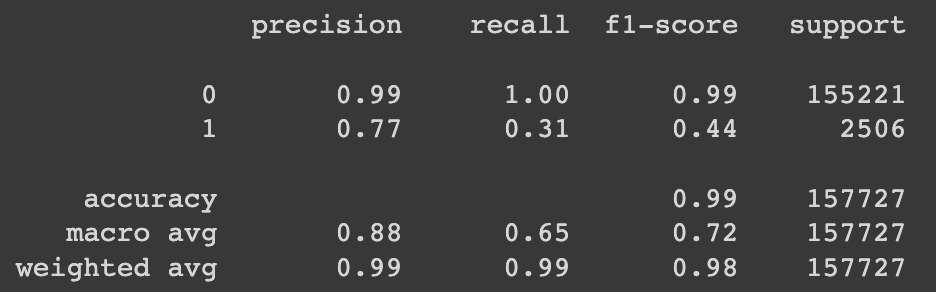
* + 1. Resultados para o Logistic Regression

Figure 5 – Logistic Regression unbalanced data

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

* 1. Análise dos resultados e melhoramentos
     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. Comecemos 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 maioritária. 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.

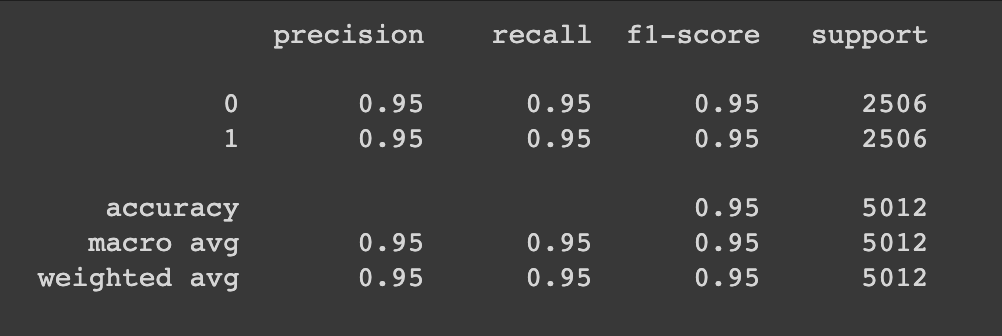
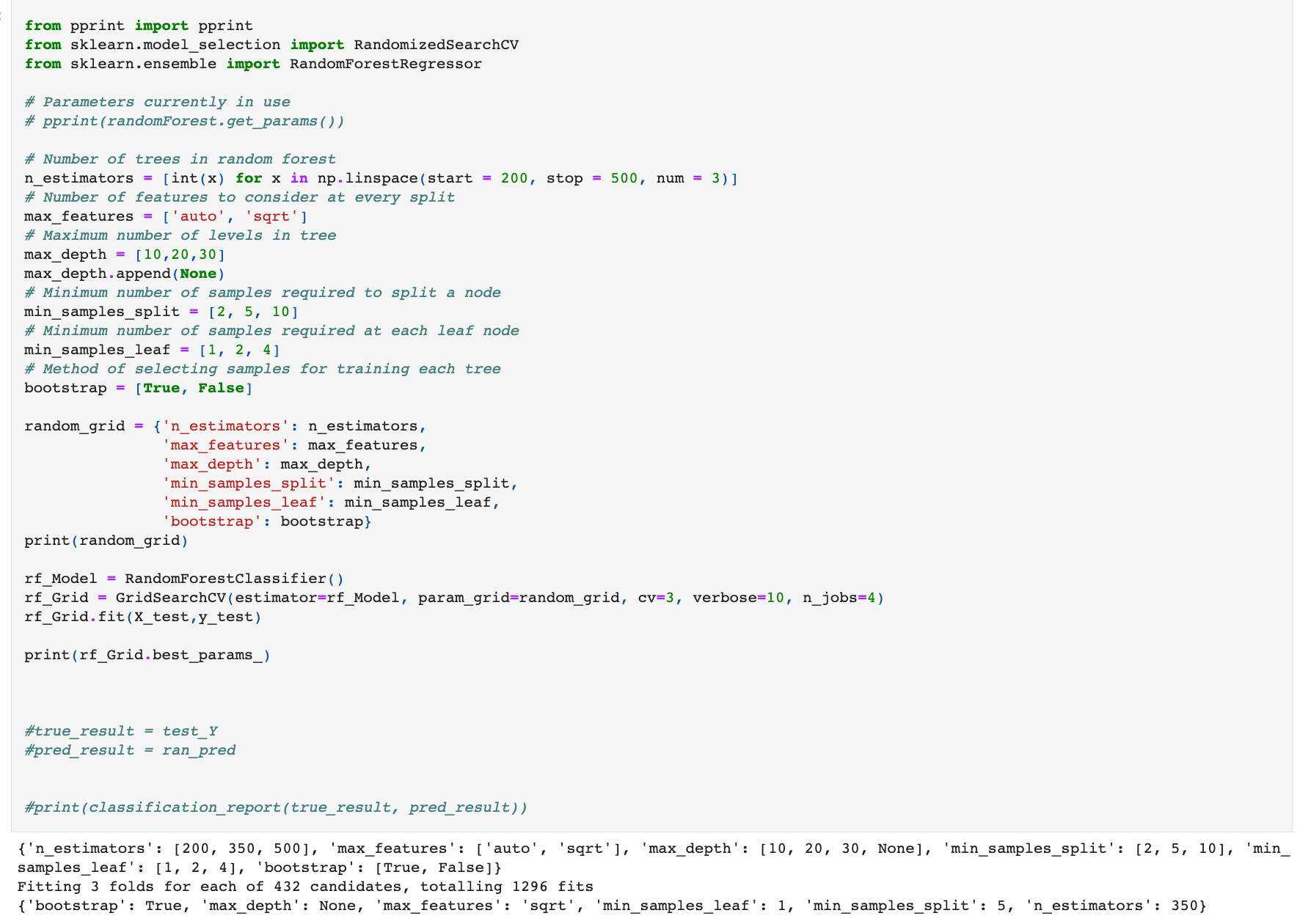


Figure 6 – Random Forest balanced data (undersampling)

* + 1. Hyper Tuning

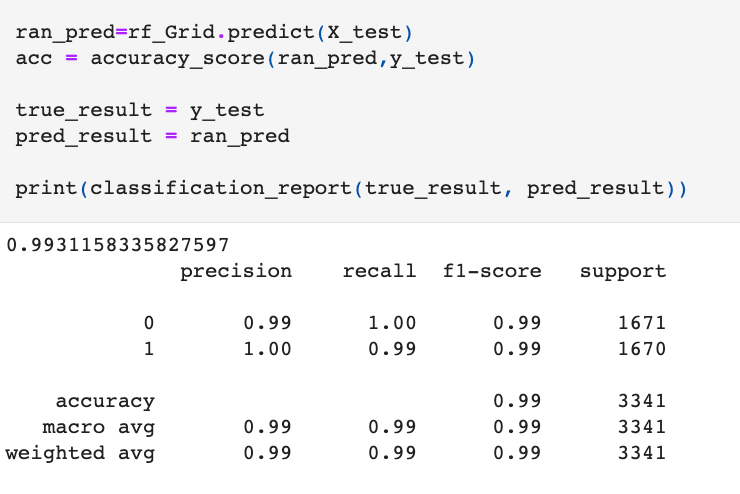
Relativamente ainda ao undersampling podemos fazer algumas experiências, sendo uma das técnicas mais usadas o tunning dos hiperparâmetros. Esta, consiste na seleção de alguns parâmetros que à priori consideramos que serão bons, e depois irá tentar com diversas combinações desses hiperparâmetros até depois 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 relembrar 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.



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.

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.

* + 1. 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 demostrado 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. 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:

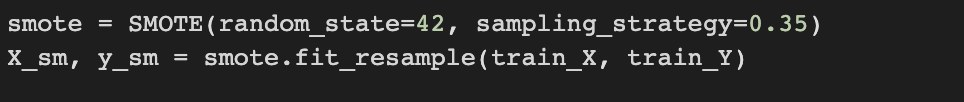


Figure 7 – SMOTE data

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.

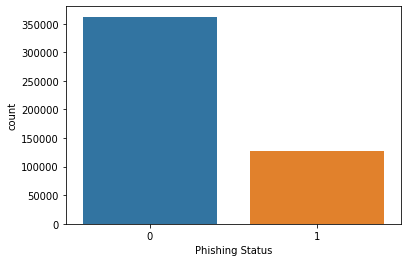


Figure 8 – Visualização do dataset depois do SMOTE

Uma imagem com texto, contador, dispositivo, calculadora

Descrição gerada automaticamenteAgora 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.

Figure 8 – Visualização do dataset depois do SMOTE

* + 1. 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.

Uma imagem com texto

Descrição gerada automaticamente

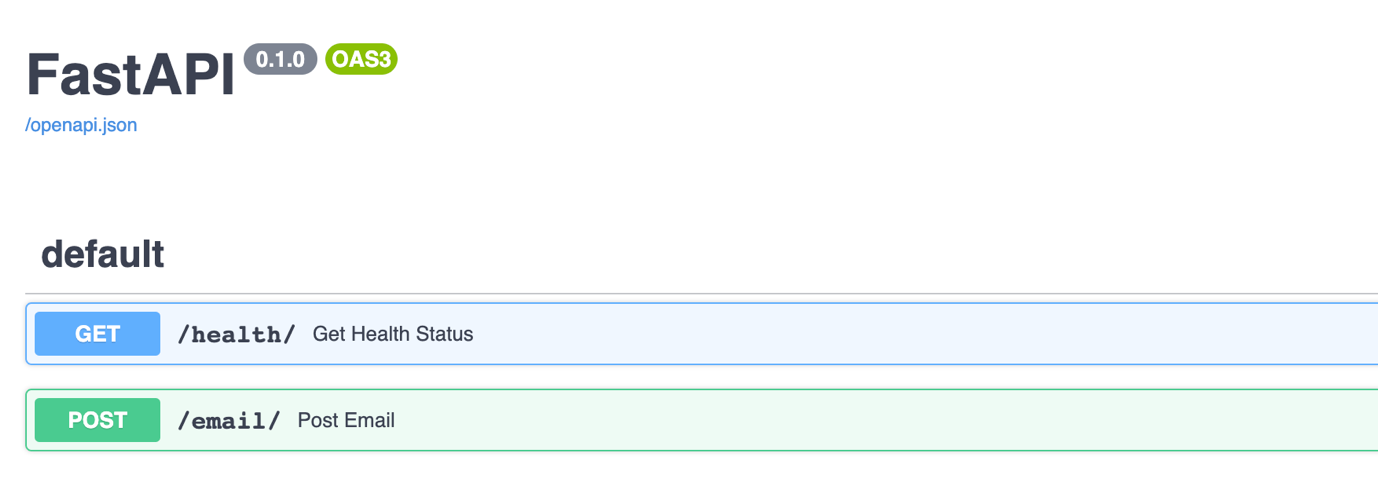
Figura 1-Código responsável pela análise do email

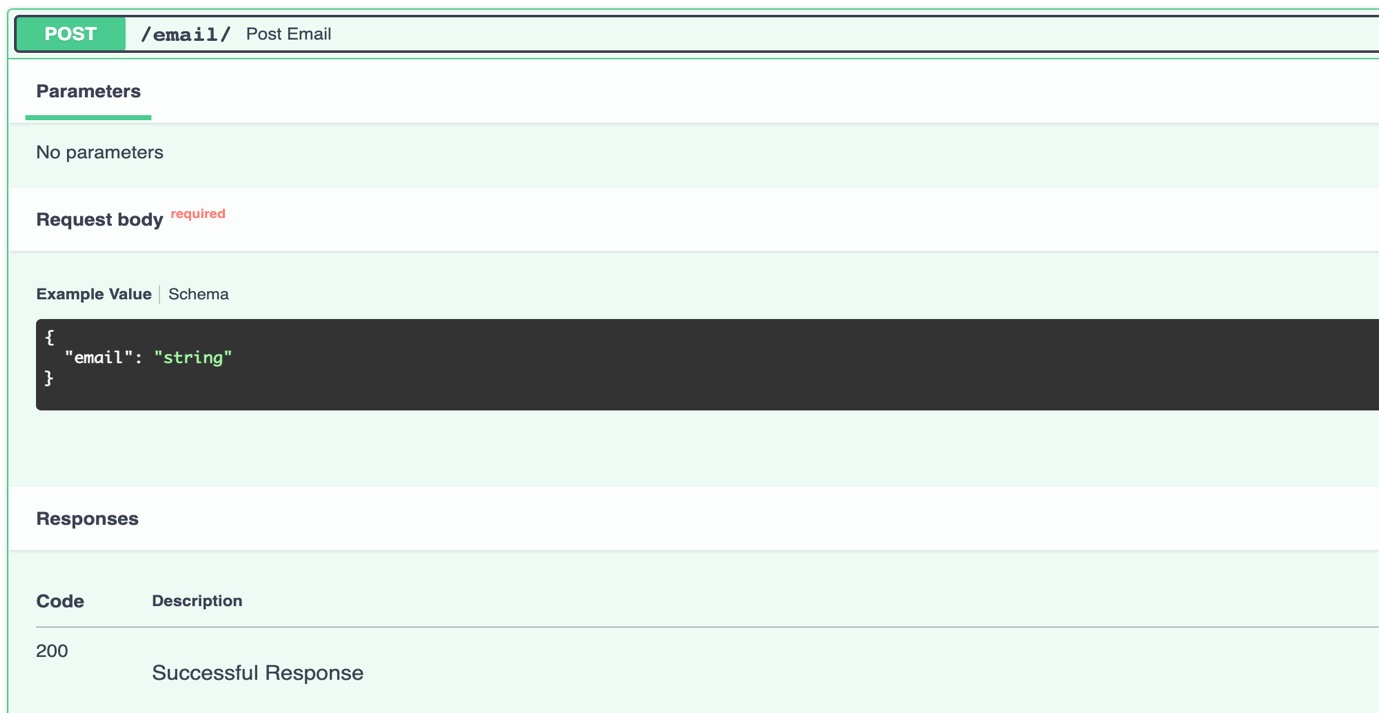
Este método irá pegar nos diversos tokens e colocá-los num array pela mesma ordem que o dataset os disponibilizou. Primeiro valor a colocar é o número de caracteres, depois a riqueza do vocabulário (importante relembrar que só funciona para emails da língua inglesa, pois a biblioteca usa um dicionário 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.

* + 1. 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.

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.

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.

Uma imagem com texto

Descrição gerada automaticamente

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.

* 1. 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.

* 1. 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 comprido. 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.

1. Conclusões

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

* 1. Objetivos concretizados

|  |  |  |
| --- | --- | --- |
| **Objetivo** | **Foi comprido** | **Como/Porque não** |
| **Estado da arte** | Sim | Diversos paper 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 neuronais** | 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 Baeys, 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) |

* 1. Limitações e trabalho futuro

Como em todos os programas de software existirá sempre limitações, e apesar de que quase todos os objetivos terem sido compridos 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.

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.

* 1. 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.

* 1. Referências

[1] “7 Types of Cyber Security Threats.” https://onlinedegrees.und.edu/blog/types-ofcyber-security-threats/ (accessed May. 25, 2022).

[2] “Cybersecurity Framework | NIST.” https://www.nist.gov/cyberframework (accessed May. 30, 2021).

[3] “5 Functions of the NIST Cybersecurity Framework.” https://www.forescout.com/blog/how-to-comply-with-the-5-functions-of-the-nist-cybersecurity-framework/ (accessed May. 30, 2021).

[4] “Artificial intelligence-based antivirus in order to detect malware preventively” https://link.springer.com/article/10.1007/s13748-020-00220-4 (accessed May. 30, 2021).

[5] “Hong, J.: The state of phishing attacks. Commun. ACM 55(1),74–81 (2012) “

[6] “https://www.statista.com/statistics/1253213/employees-phishing-emails-back-to-work-uk-us/”

[7] “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)”

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

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

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

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

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

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

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