Faculdade de Tecnologia de Americana "Ministro Ralph Biasi"

Curso Superior de Tecnologia em Segurança da Informação

Guilherme Henrique Crivelari Beloti

Lucas Amazonas Oliveira Ayres

Matheus da Costa Rosa

CLASSIFICAÇÃO DE EXECUTÁVEIS MALICIOSOS COM APRENDIZADO DE MÁQUINA:

uma abordagem usando o Dataset EMBER em amostras de ransomware

Americana, SP

2025

Guilherme Henrique Crivelari Beloti
Lucas Amazonas Oliveira Ayres
Matheus da Costa Rosa

CLASSIFICAÇÃO DE EXECUTÁVEIS MALICIOSOS COM APRENDIZADO DE MÁQUINA:
uma abordagem usando o Dataset EMBER em amostras de ransomware

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Segurança da Informação na área de concentração em segurança da informação.

Orientador(a): Prof. Me. Rafael Rodrigo Martinati

Este trabalho corresponde à versão final do Trabalho de Conclusão de Curso apresentado por Guilherme Henrique Crivelari Beloti, Lucas Amazonas Oliveira Ayres e Matheus da Costa Rosa e orientado pelo Prof. Me. Rafael Rodrigo Martinati.

Americana, SP
2025

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana Ministro Ralph Biasi- CEETEPS Dados Internacionais de Catalogação-na-fonte

BELOTI, Guilherme Henrique Crivelari

Classificação de executáveis maliciosos com aprendizado de máquina: uma abordagem usando o Dataset EMBER em amostras de ransomware. / Guilherme Henrique Crivelari BELOTI, Lucas Amazonas Oliveira AYRES, Matheus da Costa ROSA – Americana, 2025.

75f.

Monografia (Curso Superior de Tecnologia em Segurança da Informação) - - Faculdade de Tecnologia de Americana Ministro Ralph Biasi – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Rafael Rodrigo Martinati

1. Analise de Dados 2. Inteligência artificial 3. Python – linguagem de programação. I. BELOTI, Guilherme Henrique Crivelari, II. AYRES, Lucas Amazonas Oliveira, III. ROSA, Matheus da Costa IV. MARTINATI, Rafael Rodrigo V. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana Ministro Ralph Biasi

CDU: 681519
007.52
681.3.061Python

Elaborada pelo autor por meio de sistema automático gerador de ficha catalográfica da Fatec de Americana Ministro Ralph Biasi.

Guilherme Henrique Criverlari Beloti
Lucas Amazonas Oliveira Ayres
Matheus da Costa Rosa

CLASSIFICAÇÕES DE EXECUTAVEIS MALICIOSOS COM APRENDIZADO DE MÁQUINA: uma abordagem usando o Dataset EMBER em amostras de ransomware

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Curso Superior de Tecnologia em Segurança da Informação pelo Centro Paula Souza – FATEC Faculdade de Tecnologia de Americana Ministro Ralph Biasi.
Área de concentração: Segurança da informação.

Americana, 08 de novembro de 2025.

Banca Examinadora:

Rafael Rodrigo Martinati
Mestre
Fatec Americana "Ministro Ralph Biasi"

Rodrigo Brito Battilana
Mestre
Fatec Americana "Ministro Ralph Biasi"

Rogério Nunes de Freitas
Mestre
Fatec Americana "Ministro Ralph Biasi"

RESUMO

O presente trabalho tem como objetivo usar técnicas de aprendizado de máquina para classificar executáveis maliciosos usando o Dataset EMBER. Com o aumento e evolução constante das ameaças digitais, destacando-se o *ransomware*, a necessidade da aplicação de sistemas e técnicas de segurança adaptáveis torna-se imprescindível na proteção de sistemas e informações relevantes. Esse estudo, fazendo uso do Dataset Elastic Malware Benchmark for Empowering Researchers (EMBER), busca a aplicar o *machine learning* para testar um modelo capaz de distinguir entre arquivos benignos e maliciosos. A metodologia utilizada consistiu na preparação e configuração do ambiente de teste e do Dataset EMBER, que permitiu-se organizar e preparar os dados extraídos dos executáveis, etapa importante para a realização dos testes posteriores. Para a classificação dos executáveis, foi utilizado o algoritmo Light Gradient Boosting Machine (LightGBM), conhecido por ser eficiente e adequado no treinamento com um volume grande de dados, que permitiu o treinamento de um modelo seguindo as recomendações e dados fornecidos pelo próprio *benchmark* EMBER, incluindo amostras de *ransomware* para avaliar seu comportamento frente a tipos diversificados de *malwares*. Feito os testes com executáveis de *ransomware* e arquivos inofensivos, o modelo mostrou-se adequado para a função, confirmando sua capacidade de identificar realmente maliciosos e o validando como uma ferramenta promissora para a detecção proativa de ameaças, porém ele exibiu certas limitações na classificação de *ransomwares* mais recentes e modernos, algo que deve ser levado em consideração. A pesquisa contribui para o campo da segurança da informação ao validar uma metodologia moderna para a detecção de *malware*, oferecendo *insights* sobre a utilização de *datasets* como o EMBER para o desenvolvimento de sistemas de segurança mais resilientes. Logo, conclui-se que a aplicação do aprendizado de máquina na análise de executáveis pode alterar a forma como as ameaças são detectadas, trazendo implicações significativas para a proteção de sistemas e o combate a danos causados por códigos maliciosos.

Palavras-Chave: Dataset EMBER, *ransomware*, aprendizado de máquina, classificação de *malware*, LightGBM.

ABSTRACT

This work aims to use machine learning techniques to classify malicious executables using the EMBER dataset. With the increasing and constant evolution of digital threats, particularly ransomware, the need for the application of adaptive security systems and techniques becomes essential in protecting systems and relevant information. This study, using the Elastic Malware Benchmark for Empowering Researchers (EMBER) dataset, seeks to apply machine learning to test a model capable of distinguishing between benign and malicious files. The methodology used consisted of preparing and configuring the test environment and the EMBER dataset, which allowed for the organization and preparation of the data extracted from the executables, an important step for conducting subsequent tests. For the classification of executables, the Light Gradient Boosting Machine (LightGBM) algorithm was used, known for being efficient and suitable for training with a large volume of data, which allowed the training of a model following the recommendations and data provided by the EMBER benchmark itself, including ransomware samples to evaluate its behavior against diverse types of malware. After testing with ransomware executables and harmless files, the model proved suitable for the function, confirming its ability to identify truly malicious files and validating it as a promising tool for proactive threat detection; however, it exhibited certain limitations in classifying more recent and modern ransomware, something that should be taken into consideration. The research contributes to the field of information security by validating a modern methodology for malware detection, offering insights into the use of datasets such as EMBER for the development of more resilient security systems. Therefore, it is concluded that the application of machine learning in the analysis of executables can change the way threats are detected, bringing significant implications for the protection of systems and the fight against damage caused by malicious code.

Keywords: *EMBER Dataset, ransomware, machine learning, malware classification, LightGBM.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ABC	Atanasoff-Berry Computer
AOL	America Online
AUC	Area Under the Curve
AutoCAD	Autodesk Computer-Aided Design
BSD	Berkeley Software Distribution
CAGR	Taxa de Crescimento Anual Composta
COFF	Common Object File Format
CPU	Central Processing Unit
CSV	Comma-Separated Values
DDoS	Distributed Denial of Service
DLL	Dynamic-link Library
DOS	Disk Operating System
ELF	Executable Link File
ENIAC	Electronic Numerical Integrator and Computer
EMBER	Elastic Malware Benchmark for Empowering Researchers
ESET	Essential Security against Evolving Threats
FDISK	Fixed Disk Setup Program
FPR	False Positive Rate
GPU	Graphics Processing Unit
IA	Inteligência Artificial
IBM	International Business Machines Corporation
IBM PC	IBM Personal Computer
LightGBM	Light Gradient Boosting Machine
MBR	Master Boot Record
MMS	Multimedia Messaging Service
NACA	National Advisory Committee for Aeronautics
NASA	National Aeronautics and Space Administration
NHS	National Health Service

NT	New Technology
PC	Personal Computer
PE	Portable Executable
RAM	Random Access Memory
RNA	Redes Neurais Artificiais
ROC	Receiver Operating Characteristic
SHA	Secure Hash Algorithm
SKlearn	Scikit-Learn
SNARC	Stochastic Neural Analog Reinforcement Calculator
SSD	Solid-State Drive
SSH	Secure Shell
SunOS	Sun Operating System
VMS	Video Management System
XFCE	XForms Common Environment

SUMÁRIO

1 INTRODUÇÃO

A segurança da informação é uma área de importância e complexidade crescentes no cenário tecnológico atual. Com a vida e os processos sendo digitalizados cada vez mais, o desafio de proteger os dados provenientes dessa digitalização aumenta de forma constante. Com isso em mente, o *malware* (*software* malicioso) é visto como a maior e mais persistente ameaça, devido as suas novas variantes que surgem todos os dias e as técnicas avançadas, como polimorfismo e ofuscação, empregadas na confecção deles. Logo, têm-se uma corrida que exige a criação de soluções mais adaptativas e robustas para a identificação desses novos arquivos executáveis maliciosos que surgem a cada dia.

Com o aumento de *malwares* polimórficos e a criação de diversas variações daqueles já existentes, a detecção baseada somente em assinaturas torna-se ineficiente. O campo de aprendizado de máquina ou *machine learning* emergiu com uma alternativa promissora, permitindo que se empregue modelos pré-treinados que podem aprender padrões complexos e sutis a partir de grandes volumes de dados e amostras de *malwares*, na proteção de sistemas e informações. Ou seja, ao invés de se depender apenas de regras e assinaturas pré-definidas, os modelos, com sua capacidade de aprendizagem e previsão, conseguem aprender novos padrões e classificar essas ameaças de maneira automatizada, incluindo aquelas ainda desconhecidas para eles. Essa abordagem vai mais a fundo na análise dos arquivos maliciosos, não dependendo apenas de assinaturas e verificações de *hashes*.

Esse trabalho se foca nesse cenário, na aplicação de técnicas de aprendizado de máquina para automatizar e aprimorar a capacidade de classificação de ameaças. Especificamente, utiliza-se o Dataset EMBER do ano de 2018, um conjunto de dados de referência maduro e amplamente aceito pela comunidade de pesquisa em segurança, que fornece uma base rica e diversificada de amostras de executáveis, tanto maliciosos quanto benignos, para treinamento e avaliação de modelos, sendo inclusive de código aberto e gratuito para uso.

1.1 PROBLEMA DE PESQUISA

Diante da necessidade de métodos de detecção mais eficazes e adaptáveis, o problema de pesquisa que norteia esse trabalho é:

Como aplicar técnicas de aprendizado de máquina para identificar e classificar executáveis maliciosos de forma eficaz, utilizando o Dataset EMBER como base de treinamento e teste?

1.2 JUSTIFICATIVA

A relevância do estudo realizado está na necessidade de se adaptar a evolução constante das ameaças digitais. A sofisticação dos ataques e a velocidades com que novas variantes são produzidas, além das técnicas de polimorfismo e ofuscação empregadas, tornam a classificação de *malwares* algo cada vez mais desafiador. O uso do *machine learning*, junto com um *benchmark* de qualidade como o Dataset EMBER, representa uma abordagem promissora no desenvolvimento de sistemas de classificação automatizados, capazes de se adaptar e aprender constantemente. A contribuição desse trabalho é demonstrar a eficácia de um modelo de classificação baseado em aprendizado de máquina na classificação de executáveis maliciosos, além de fornecer *insights* sobre a utilização de *datasets* na segurança de sistemas.

1.3 OBJETIVOS

1.3.1 OBJETIVO GERAL

O objetivo geral desse trabalho é:

Aplicar técnicas de aprendizado de máquina na classificação de executáveis maliciosos utilizando o Dataset EMBER.

1.3.2 OBJETIVOS ESPECÍFICOS

Para alcançar o objetivo geral, os seguintes objetivos específicos foram definidos:

1. Preparar o ambiente de desenvolvimento e configurar o Dataset EMBER para o treinamento do modelo de aprendizado de máquina voltado à classificação de executáveis.

2. Treinar um modelo de classificação de executáveis maliciosos com base nas informações e parâmetros recomendados para o Dataset EMBER.

3. Testar o modelo treinado utilizando um arquivo malicioso e um arquivo benigno, avaliando sua capacidade de classificação e desempenho.

1.4 ESTRUTURA DO TRABALHO

Esse Trabalho de Conclusão de Curso está organizado em cinco capítulos, organizados de forma a conduzir o leitor desde o embasamento teórico até a apresentação dos resultados práticos.

O capítulo 2, referencial teórico, apresenta os fundamentos teóricos necessários para a compreensão do tema, abordando conceitos essenciais de segurança da informação, o panorama das ameaças de *malware*, a arquitetura de arquivos executáveis e os princípios do aprendizado de máquina aplicados na detecção de ameaças, além de uma descrição detalhada do Dataset EMBER.

O capítulo 3, metodologia de desenvolvimento, descreverá o caminho percorrido para a realização da pesquisa, detalhando o processo de preparação do ambiente, a obtenção e pré-processamento dos dados, a escolha e configuração do algoritmo de *machine learning* e os procedimentos de treinamento e teste do modelo realizados.

O capítulo 4, resultados e análise, apresenta e discute os achados da pesquisa exibindo as métricas de desempenho do modelo treinado, como acurácia, precisão, *recall* e AUC, além de uma análise crítica dos resultados obtidos na classificação das amostras de teste.

Finalmente, o capítulo 5, resultados e considerações finais, irá retomar o problema e os objetivos propostos, apresentando as conclusões do estudo e discutindo as limitações encontradas no decorrer do trabalho e sugeridas direções para futuras pesquisas na área de classificação de executáveis maliciosos utilizando aprendizado de máquina.

2 REFERENCIAL TEÓRICO

Antes de se debater sobre a IA (Inteligência artificial), é necessário entender bem suas origens, antecessores e criadores, além dos usos que essas tecnologias tiveram no decorrer da história, com o intuito de se contextualizar melhor e compreender como tudo se desenvolveu até os dias atuais. Tais assuntos serão tratados nos subcapítulos abaixo, começando pelos computadores.

2.1 A HISTÓRIA DOS COMPUTADORES

2.1.1 A ORIGEM DO TERMO “COMPUTADOR” E “COMPUTAR”

Inicialmente, quando se refere ao termo “computador” a primeira imagem que se vem à mente é a de uma incrível máquina amplamente usada nos últimos tempos com finalidades profissionais, pessoais e de entretenimento. No entanto, pouco se pensa sobre como essa tecnologia chegou a esse ponto e menos ainda sobre a origem do seu nome.

A palavra “computador” é um nome errôneo dado a máquinas digitais que agora residem nas mesas da maioria das pessoas, já que, até meados da Guerra Fria, o termo se referia a uma ocupação muito necessária, é o que diz Kelly *et al.* (2013, p. 19). As chamadas “computadores humanos” eram pessoas responsáveis pela realização de cálculos matemáticos complexos e pelo armazenamento e manipulação de informações, exatamente o que é feito por máquinas atualmente (Kelly *et al.*, 2013, p. 19).

Um exemplo histórico desse trabalho se dá no Reino Unido que, no início da Segunda Guerra Mundial, com um contrato com o Ministério da Guerra, Leslie John Comrie, pioneiro em computação mecânica da época, usou uma equipe de dezesseis “computadores humanos” para calcular e produzir tabelas de artilharia que eram usadas pelos soldados para mirar os canhões (Kelly *et al*., 2013, p. 74). Outro momento, na década de sessenta, em meio a corrida espacial, esses profissionais, a maioria mulheres afrodescendentes, eram muito empregados nos cálculos de trajetória orbital de voos espaciais na NACA (National Advisory Committee for Aeronautics) que, mais tarde, se tornou a NASA (National Aeronautics and Space Administration), o que é demonstrado por Shetterly, (2016, p. 23).

Com o fim da guerra e, consequentemente, com o desenvolvimento de diversas máquinas eletrônicas capazes de fazer o trabalho de computar diversas vezes mais rápido e com melhor eficiência do que humanos, esses trabalhadores foram deixados de lado (Kelly *et al*., 2013, p. 74). Assim sendo, o termo “computador” deixou de se referir a trabalhadores cuja função era desempenhar cálculos complexos e foi atribuído a máquinas eletrônicas que realizavam o mesmo trabalho, porém de forma mais eficiente, rápida e barata (Crevier, 1993, p.28). Agora, no próximo parágrafo, será abordado o termo “computar”.

Independente de um computador ser digital, baseia-se em circuitos e estruturas eletrônicas para funcionar, ou analógico, que faz uso de partes mecânicas para operar, suas funções básicas permanecem as mesmas: obter informações, interpretá-las e gerar um resultado (Woiler, 1970). É evidente que, na informática, essas são exatamente as etapas que um computador percorre para cumprir seu papel na agilização e precisão das mais diversas atividades. Portanto, “computar” significa obter uma entrada, processá-la e produzir uma saída (Woiler, 1970). A origem do termo vem do latim *computo* que significa “fazer o cômputo de”, “contar”, “calcular”, “orçar” ou, no contexto da informática, “processar” (Dicionário Priberam, 2025). Agora, com essas informações em mente, será abordado os principais computadores e dispositivos de cálculo criados durante a história, no intuito de fornecer uma clara linha do tempo de fatos que levaram essas máquinas a se tornarem o que se vê hoje.

2.1.2 OS PRIMEIROS COMPUTADORES, PROGRAMAS E DISPOSITIVOS DE CÁLCULO

É de conhecimento geral que, no decorrer da história, foram desenvolvidos muitos dispositivos tecnológicos cujo objetivo era automatizar cálculos. Kelly *et al.* (2013, p. 12) cita que máquinas de cálculo de mesa já eram estudadas e desenvolvidas por Blaise Pascal e Gottfried Leibniz, indicando que no século XVII essa tecnologia já era cobiçada por grandes mentes. Kelly *et al*. (2013, p. 45) também menciona alguns outros dispositivos históricos como o tear de Joseph-Marie Jacquard, desenvolvido no início do século XIX, que revolucionou a indústria têxtil por usar cartões perfurados para o armazenamento de instruções para padrões de tecelagem e o Aritmômetro de Thomas de Colmar de Alsácia, feita em 1820, sendo a primeira máquina comercialmente produzida que permitia realização das quatro

operações matemáticas básicas: adição, subtração, multiplicação e divisão. A seguir, uma imagem da calculadora mecânica de Pascal, criada por volta de 1642, e o tear de Jacquard, feito em 1804.

Além disso, Russell e Norvig (2009, p. 33) abordam alguns dos exemplos mais conhecidos como a Máquina Diferencial e a Máquina Analítica de Charles Babbage, ambas criadas na década de 1830, cujo objetivo era o cálculo de tabelas matemáticas. A segunda foi programada por Ada Lovelace, colega de Babbage e considerada a primeira programadora do mundo, que especulou que um dia a máquina poderia compor música ou jogar xadrez, o que mostra que ela já compreendia a dimensão em que a inteligência das máquinas poderia chegar (Russell; Norvig, 2009, p. 33).

Agora, já na segunda metade do século XIX, embora nem todas sejam exatamente máquinas de cálculo matemático, Dyson (2012, p. 88), juntamente com Suleyman e Bhaskar (2023, p. 43), Kelly (2013, p. 13) e Crevier (1993, p. 27), citam outros dispositivos que agilizaram e automatizaram processos, como a máquina de escrever comercialmente bem-sucedida da Remington, em 1874, o telefone, introduzido por Alexander Graham Bell, em 1876, algo que, para a época, foi um marco na agilização das comunicações, as primeiras estações elétricas em Londres e Nova York, por volta de 1882 e a máquina de tabulação de Herman Hollerith, em 1890, que processava dados para o censo dos Estados Unidos usando cartões perfurados, uma grande inovação para e época.

Já trazendo para o século passado, no ano de início da Primeira Guerra Mundial, em 1914, Schmidhuber (2022, p. 23) menciona o El Ajedrecista, uma máquina funcional, construída pelo espanhol Leonardo Torres y Quevedo, capaz de jogar xadrez, a qual é considerada até mesmo o marco inicial da IA. A Segunda Guerra Mundial, apesar dos horrores desse momento sombrio da história, foi o período em que mais houve avanços na computação, algo que é indicado por vários autores como Kelly (2013, p. 75) que menciona a construção do Harvard Mark I da IBM (International Business Machines), iniciada em 1937 e finalizada em 1943, revelando a convergência, mesmo a partir daquela época, de máquinas de cálculo e escritório.

Agora, focando nos computadores criados no período da Segunda Guerra Mundial, Crevier (1993, p. 312) fala do Zuse-2, o primeiro computador eletromecânico construído pelo alemão Konrad Zuse, em 1939. Russell e Norvig (2009, p. 33) mencionam a série Heath Robinson, em 1940, construídos pela equipe de Alan Turing na Grã-Bretanha cujo uso era decifrar mensagens alemãs e a Bombe, também

conhecida como Máquina de Turing, finalizada em 1940, cujo papel foi vital no desenrolar da guerra devido sua capacidade de quebrar as cifras geradas pela Enigma alemã. Russell e Norvig (2009, p. 34-35) também citam o ABC (Atanasoff-Berry Computer), iniciado em 1940 e finalizado em 1942, sendo o primeiro computador eletrônico, construído por John Atanasoff e Clifford Berry, o Zuse-3, em 1941, a evolução do Zuse-2 de Konrad Zuse, onde se introduziu os números de ponto flutuante e a primeira linguagem de programação de alto nível chamada Plankakül e o Colossus, em 1943, máquina baseada em válvulas de vácuo, também feita pela equipe de Turing cujo uso era a quebra das cifras geradas pela Lorenz alemã. Por fim, Schmidhuber (2022, p. 24) aborda o desenvolvimento do ENIAC (Electronic Numerical Integrator and Computer), no final da guerra, em 1945, desenvolvido na Universidade da Pensilvânia, considerado o primeiro computador digital programável multiuso cujo objetivo inicial era o cálculo de tabelas de artilharia para o exército dos Estados Unidos.

Dito isso, próximo subtópico, será discutido o termo inteligência artificial, juntamente com os seus conceitos e história.

2.2 INTELIGÊNCIA ARTIFICIAL

Agora que já se tem em mente alguns dos principais eventos e antecessores da inteligência artificial, pode-se partir para os primeiros projetos, estudos e ideias que posteriormente levaram a criação do campo de estudo dessa incrível tecnologia.

No ano de 1943, em meio ao conflito entre as potências mundiais, o neurofisiologista Warren McCulloch e o matemático Walter Pitts, escreveram um artigo revolucionário intitulado “A Logical Calculus of the Ideas Immanent in Nervous Activity” (Um Cálculo Lógico de Ideias Imanente na Atividade Nervosa) (Santos, 2023), onde propuseram um modelo matemático de redes neurais, assim como as bases para o desenvolvimento da inteligência artificial.

Já em 1950, Alan Turing, o famoso matemático britânico e criador da “The Bombe”, abordada anteriormente, publica o artigo intitulado "Computing Machinery and Intelligence" (Máquinas de Computação e Inteligência), que estabelece questionamentos acerca da capacidade das máquinas de pensarem e agirem como humanos e seu método, atualmente conhecido como “Teste de Turing” ou “Jogo da Imitação”, que permite calcular a capacidade delas de imitarem o comportamento

humano (Santos, 2023). Isso se tornou um pilar central no estudo e desenvolvimento da inteligência artificial até os dias atuais (Mucci, 2024).

Pouco tempo depois, em 1951, Marvin Minsky e Dean Edmunds, apoiados pelo matemático e físico John von Neumann, criaram a primeira rede neural artificial, chamada SNARC (Calculadora de Reforço Analógico Neural Estocástico) (Mucci, 2024), que por meio de três mil válvulas eletrônicas, simulava quarenta unidades semelhantes a neurônios. Ela foi uma tentativa inicial de modelar os processos de aprendizado no cérebro humano. Com essas informações em mente, a partir do próximo parágrafo, será tratado a criação oficial do termo “inteligência artificial”, juntamente com seu significado.

Mesmo com toda a contribuição e invenções citadas até o momento, o termo “inteligência artificial” ainda não existia formalmente e, sim, apenas como uma expressão para definir uma ideia. Foi somente no ano de 1956, na Conferência de Dartmouth, que, John McCarthy, considerado o pai da inteligência artificial, juntamente com Marvin Minsky, Claude Shannon, Nathaniel Rochester e outras figuras importantes, fundaram o termo “inteligência artificial” (Abeliuk; Gutiérrez, 2021), a formalizando de vez como um novo campo de estudo científico. Somente a partir daí, essa expressão começou de fato a ganhar popularidade, sendo um marco inicial para a tecnologia em questão.

Agora que o termo para essa invenção já existe, deve-se também atribuir os conceitos que definem uma inteligência artificial. Segundo o dicionário Oxford Languages, o termo “inteligência” significa “faculdade de conhecer, compreender e aprender” e “artificial” define-se por “produzido pela mão do homem, não pela natureza; postiço”. Portanto, “inteligência artificial”, juntando a definição de cada palavra separadamente, seria algo como uma simulação da capacidade de aprendizado e conhecimento do ser humano em algo criado por ele mesmo. Agora, com isso tudo definido, serão abordados os subcampos da IA: aprendizado de máquina, redes neurais e aprendizado profundo. Com isso, no subcapítulo a seguir, será abordado a respeito do aprendizado de máquina.

2.2.1 APRENDIZADO DE MÁQUINA

Agora, que já se sabe um pouco sobre a IA, deve-se ter conhecimento de alguns dos seus subcampos, como o *machine learning* ou aprendizado de máquina.

Esse termo é uma área da ciência de programação dos computadores para que eles possam aprender com dados, utilizando algoritmos que aprimoram seu desempenho e precisão com base em experiência, sem que precisem ser explicitamente programados, é o que diz Géron (2019, p. 30).

Logo, observa-se a importância desse campo, afinal, um programador, na construção de um sistema de *antispam* de e-mails, por exemplo, precisaria observar as palavras mais comuns contidas neles e então criar uma série de regras a um algoritmo, para que então ele possa filtrá-los, cenário citado por Russell e Norvig (2009, p. 884). O problema existente nesse sistema é que, caso ele não seja continuamente atualizado pelo programador com novos dados, ele se tornará ineficiente em seu trabalho, pois os autores dos *spams*, rapidamente, adaptariam seus e-mails, seja usando palavras-chave diferentes ou alterando diversas palavras por seus sinônimos, assim contornando o algoritmo (Géron, 2019, p. 32).

Com isso em mente, é aqui que entra o *machine learning*: ao se coletar um grande volume de dados, nesse caso, e-mails já verificados como *spam* por humanos, rotulá-los, assim treinando um modelo, e disponibilizá-los para o software, ele, com base nisso, saberá quais e-mails são *spam* ou não. Géron (2019, p. 33) explica que, no aprendizado de máquina, o algoritmo se foca em encontrar padrões nos dados e não em seguir regras, assim permitindo que, mesmo que chegue um dado ainda desconhecido, ele, com base nas informações já adquiridas, possa fazer uma previsão sobre esse dado. No exemplo citado, ele poderia prever se um e-mail, mesmo não sendo conhecido, é *spam* ou não com base no volume de dados já possuído. Esse é o objetivo central e a particularidade do *machine learning*.

Bishop (2006, p. 22), juntamente com Goodfellow, Bengio e Courville (2016, p. 130), revelam que o principal desafio do aprendizado de máquina é criar algoritmos que funcionem bem tanto com os dados com os quais foram treinados, mas também com novas informações, nunca vistas pela máquina, capacidade chamada de generalização, que é desenvolvida ao se encontrar padrões nos dados. Existem três principais tipos de aprendizado de máquina, que serão tratados nos parágrafos seguintes.

Aprendizado supervisionado: Géron (2019, p. 74), Goodfellow, Bengio e Courville (2016, p. 125) dizem que esse é o tipo mais comum. Resume-se a alimentar o algoritmo com dados de treinamento com exemplos de pares de entrada e saída, onde cada saída é rotulada por um humano, cujo objetivo é mapear as entradas às

saídas corretas. As duas tarefas mais comuns supervisionadas são classificar e prever valores numéricos.

Aprendizado não supervisionado: Géron (2019, p. 409) e Bishop (2006, p. 23) explicam que nesse método os dados de treinamento são compostos por um conjunto de entradas sem quaisquer valores de saída correspondentes, com o objetivo de descobrir padrões e estruturas nos próprios dados e, com isso, formar agrupamentos, distribuir e projetar os dados para posterior visualização.

Aprendizado por reforço: Géron (2019, p. 37), Bishop (2006, p. 23) e Goodfellow, Bengio e Courville (2016, p. 126) citam que essa é uma abordagem muito utilizada para treinamento de *bots* para as mais diversas atividades. Resume-se a ensinar um agente de *software* a se comportar em um cenário realizando ações e observando resultados. Esse programa recebe punições ou recompensas, uma espécie de *feedback*, com base no resultado que ele gera, assim o ensinando uma política de atividades que maximizem sua recompensa total ao longo do tempo.

Portanto, os maiores benefícios do uso do aprendizado de máquina é a resolução de problemas que seriam intratáveis pela programação tradicional e a adaptação a novos ambientes e ameaças, coisas que são, de certa forma, a fundação da IA, afinal a característica central da inteligência é a capacidade de aprender, algo enfatizado por Russell e Norvig (2009, p. 21). Agora, no próximo subcapítulo, serão abordadas as redes neurais, outro subtópico da inteligência artificial.

2.2.2 REDES NEURAIS

Outro subcampo da IA são as Redes Neurais Artificiais (RNA), que são modelos de aprendizado de máquina inspirados nos neurônios biológicos em nossos cérebros, daí o nome “redes neurais” (Géron, 2019, p. 463). Eles são feitos para realizar atividades que exigiram inteligência humana, como reconhecimento de padrões, previsão e classificação, cuja ideia principal é fazer muitas unidades computacionais, chamadas, nesse caso, de neurônios, trabalharem juntas para resolver tarefas complexas e exibir um comportamento inteligente quando interconectadas em uma rede (Goodfellow; Bengio; Courville, 2016, p. 39).

Assim como em um computador, o ciclo de vida de informações em uma rede neural segue o mesmo padrão: entrada, processamento e saída. Na inserção de dados (que são recebidos pela chamada camada de entrada, uma das camadas de

neurônios que constituem a rede neural, que por sua vez pode ser composta por uma ou mais camadas) deve-se transformá-los, antes de tudo, em números, normalmente na forma de vetores ou matrizes, é o que dizem Géron (2019, p.12) e Mitchell (2019, p. 33). Dependendo do formato desses dados, podendo ser imagem, texto ou dados tabulares, pode ser usado uma transformação usando cálculos diferentes, mas o objetivo é sempre o mesmo nessa etapa: obter uma representação numérica dessas informações (Bishop, 2006, p. 158). A camada de entrada não realiza nenhuma computação, ela apenas passa os dados brutos para a camada oculta da rede.

Agora, após a entrada ser feita com sucesso, essas informações passam por uma ou mais camadas ocultas (*Hidden layers*) que é onde serão, de fato, processadas. Mitchell (2019, p. 44) aborda que o objetivo principal nessa parte é transformá-las em representações mais abstratas e, consequentemente, úteis para a tarefa final, como classificação ou regressão. O processamento nas camadas ocultas pode ser dividido em duas etapas principais, com a primeira sendo a combinação linear, ou soma ponderada, em que, segundo Géron (2019, p. 472) e Bishop (2006, p. 247), cada neurônio (ou unidade de processamento) recebe as saídas de todos os neurônios da camada anterior e, considerando o peso (*weight*) associado a cada uma dessas conexões, que indica a força ou importância daquela conexão, o neurônio calcula uma soma ponderada de suas entradas, adicionando também um valor de viés (*bias*) a esse cálculo.

Utilizando uma função de ativação, que é um cálculo matemático que auxilia o neurônio a aprender um padrão complexo, a segunda etapa de processamento faz uso dessa função do tipo não linear, que, após a etapa anterior, que ao ser aplicada pelo neurônio, o resultado obtido é usado para produzir sua saída final (Bishop, 2006, p. 248). A não linearidade é crucial nessa etapa, senão a rede neural constituída de múltiplas camadas seria equivalente a uma rede neural de única camada, limitando severamente sua capacidade de aprender padrões complexos (Géron, 2019, p. 483).

Após o processamento realizado, a camada de saída, que é a última da rede, recebe as saídas da última camada oculta e as transforma no formato final desejado (Goodfellow; Bengio; Courville, 2016, p. 384). Dependendo do tipo de atividade sendo executada, a saída produzida será diferente, por exemplo, se a tarefa é prever um valor contínuo, a camada de saída é composta por um único neurônio com uma função de ativação do tipo linear, cuja saída é a previsão numérica final (Géron, 2019, p. 485).

Para classificações, começando pela binária, para identificar se uma entrada é ou não spam, por exemplo, a camada de saída também será constituída por somente um neurônio, que desta vez usará uma função de ativação sigmoide, comprimindo a saída em um valor entre zero e um, interpretado como a probabilidade de ser positivo ou negativo. Géron (2019, p. 488) explica que, para classificação multiclasse, o segundo tipo, se consiste em classificar a entrada em um certo número de classes exclusivas, usando um número igual de classes e neurônios, cada um representando uma classe, que constituirão a camada de saída. Géron (2019, p. 489) também salienta que a função de ativação *softmax* é aplicada a toda a camada para garantir que as saídas de todos os neurônios sejam valores entre zero e um e que juntos somem um. Ao analisar cada uma delas, pode-se descobrir a probabilidade de a entrada pertencer àquela classe ou não. Agora, no subtópico abaixo, será abordado o *deep learning.*

2.2.3 APRENDIZADO PROFUNDO

O *deep learning* ou aprendizado profundo, é um subcampo do machine learning que usa modelos computacionais das redes neurais, mais precisamente as *Deep Neural Networks* ou Redes Neurais Profundas, que são redes neurais constituídas por muitas camadas de neurônios, permitindo um aprendizado muito mais profundo, daí o seu nome (Friedman; Hastie; Tibshirani, 2009, p. 7). Ele é muito usado em domínios como reconhecimento de imagem e fala, tradução de idiomas, análise de dados de aceleradores de partículas, descoberta de medicamentos e análise de dados genômicos, justamente por serem áreas amplas que exigem um bom aprofundamento para que sejam dominadas, é o que dizem Suleyman e Bhaskar (2023, p. 158) e Lecun, Bengio e Hinton (2023, p. 2).

A ideia fundamental do *deep learning* é permitir que computadores aprendam a partir da experiência e entendam o mundo em termos de uma hierarquia de conceitos, onde cada um deles é definido em relação a outros mais simples (Goodfellow; Bengio; Courville, 2016, p. 24). Em vez de serem programadas com regras explícitas por humanos, como “gatos têm orelhas pontudas e bigodes”, as redes de aprendizado profundo aprendem automaticamente a partir de dados (Lecun; Bengio; Hinton, 2023, p. 2). Essa abordagem é um tipo de aprendizado de representação, onde, em cada camada, o modelo transforma a representação do nível

anterior em outra de nível superior sendo um pouco mais abstrata (Lecun; Bengio; Hinton, 2023, p. 3). As primeiras camadas aprendem a detectar características mais simples e de baixo nível enquanto as camadas mais profundas combinam esse aprendizado para assimilar conceitos mais complexos e abstratos, também explicado por Lecun, Bengio e Hinton (2023, p. 2). A principal característica do *deep learning* está no fato de que essas camadas não são projetadas por engenheiro humanos, mas sim aprendidas a partir dos dados com um procedimento de aprendizado de propósito geral, a diferenciando das técnicas de aprendizado de máquina convencionais (Suleyman; Bhaskar, 2023, p. 162).

Lecun, Bengio e Hinton (2023, p. 4) dizem que o funcionamento do aprendizado profundo é um processo iterativo, normalmente usando um algoritmo de otimização chamado *stochastic gradient descent* ou descida de gradiente estocástico. O processo pode ser divido nas seguintes etapas: antes de tudo, como no *machine learning*, é obtido uma entrada de grande quantidade de dados rotulados, sejam imagens, textos, áudios etc. Feito isso, inicia-se o treinamento de minimizar a função de custo (ou perda), que mede o erro entre a saída produzida pela rede e a saída desejada (o rótulo correto) (Lecun; Bengio; Hinton, 2023, p. 2). Agora, com a função de custo reduzida, começa o processo de *forward propagation* ou passagem direta, em que um lote de dados de treinamento é passado através da rede, em cada camada nela, desde a entrada até a saída. Usando todo o processo descrito anteriormente no tópico de redes neurais (cálculo de soma ponderada, aplicação da função de ativação etc.) a esses dados, é produzido, no final dessa etapa, a previsão da rede, o que é explicado por Goodfellow, Bengio e Courville (2016, p. 191) e Lecun, Bengio e Hinton (2023, p. 3).

Terminando essa etapa, após a passagem direta, o erro é calculado. Em seguida, o algoritmo de retropropagação calcula o gradiente da função de erro em relação a cada peso e viés da rede (Géron, 2019, p. 480). Esse algoritmo, que é uma aplicação eficiente da regra da cadeia do cálculo, propaga o gradiente de erro da camada de saída até a camada de entrada, ou seja, “para trás”, o que determina a contribuição de cada parâmetro para o erro total (Lecun; Bengio; Hinton, 2023, p. 5). Por fim, com os resultados obtidos até agora, é realizado um ajuste de pesos, em que o gradiente calculado é usado por um algoritmo de otimização para ajustar os pesos e vieses da rede na direção que reduz o erro. Lecun, Bengio e Hinton (2023, p. 5) citam que esse processo é repetido milhões de vezes, usando lotes de dados

diferentes, até que o desempenho da rede em um conjunto de validação pare de melhorar, o que mostra que chegou em seu ponto de precisão máximo. Com isso, encerra-se o tópico de aprendizado profundo, assim como o de inteligência artificial. No subtópico seguinte, será discutido o cenário atual da IA no mundo.

2.2.4 CENÁRIO ATUAL DA INTELIGÊNCIA ARTIFICIAL

A inteligência artificial se tornou um pilar fundamental no nosso século e com o passar do tempo ela deixou de ser uma tecnologia em desenvolvimento e se tornou uma ferramenta utilizada no cotidiano (Thunderbit, 2025).

Em 2025, 78% das organizações relataram que usam a IA, com um aumento de 55% em relação ao ano anterior, seu uso acabou impactando diversos setores da sociedade (Ramos, 2024). Na saúde, por exemplo, algoritmos de aprendizado de máquina são usados para diagnósticos mais precisos e aceleram as descobertas de medicamentos. Um estudo da IBM Watson Health demonstrou que sistemas de IA podem identificar anomalias em imagens médicas com até 95% de precisão, superando, em alguns casos, a acurácia de médicos humanos (IBM, 2025). Em outros setores como na educação, plataformas personalizam e facilitam o aprendizado, no mercado veículos autônomos geram rotas e se controlam sem auxílio, plataformas como o YouTube e Netflix aprendem os gostos do cliente para deduzir e recomendar anúncios e vídeos personalizados, e na indústria financeira, detecta fraudes e otimiza investimentos.

Em 2025, o mercado global de IA está em expansão exponencial. É estimado que o valor do mercado está na faixa de US$ 391 bilhões, com projeções de crescimento de cerca de US$ 1,81 trilhão até 2030, com uma taxa de crescimento anual composta (CAGR) de 37,3% (Founders Forum Group, 2025). A inteligência artificial generativa sozinha atingiu 33,9 bilhões em 2024 e deve atingir cerca de US$ 356,10 dólares até 2030, um aumento de sessenta vezes em relação a 2020 (Demetrio, [s.d.]). No subcapítulo seguinte será iniciado o assunto sobre executáveis maliciosos, onde será explicado o que são, seus tipos, como funcionam e suas origens.

2.3 EXECUTÁVEIS PE

Daniel Donda ([s.d.]), especialista em cibersegurança conhecido, define Portable Executable (PE), como o formato padrão do Windows x86 e x64 para executáveis portáteis, equivalente ao formato Executable Link File (ELF) no sistema operacional Linux. Ele é o sucessor do antigo formato Common Object File Format (COFF) usado em sistemas Windows NT.

Um executável PE é uma estrutura de dados que oferece ao *loader* do sistema operacional todas as informações necessárias para que o código do executável seja encapsulado, carregado na memória e executado. As estruturas dos arquivos PE possuem os seguintes componentes principais, descritos nas Tabelas 1, 2 e 3:

Tabela 1 – Estrutura do arquivo PE

Estrutura	Descrição
DOS Header	Primeiros 64 bytes, identificam o arquivo como executável.
DOS Stub	Exibe uma mensagem de erro se executado em modo DOS.
PE File Header	Inclui SIGNATURE, IMAGE_FILE_HEADER e IMAGE_OPTIONAL_HEADER, definindo a aparência do restante do arquivo.
Image Optional Header	Apesar do nome, este não é apenas um cabeçalho opcional, ele contém informações críticas que estão além das informações básicas contidas na estrutura.

Fonte: Próprios autores

Na Tabela 1, pode-se observar a estrutura do arquivo PE, algo importante para entender melhor como ele funciona e quais campos ele possui, afinal isso é parte das informações contidas no Dataset EMBER que permite a classificação de um executável de maneira correta. A seguir, encontra-se a Tabela 2, contendo o *section table* e seus campos:

Tabela 2 – Section table e seus campos

Estrutura	Descrição
Section Table	São seções do arquivo
Name	Nome da seção
VirtualSize	Tamanho em memória
SizeOfRawData	Tamanho no disco
PointerToRawData	Deslocamento dos dados
Characteristics	Atributos da seção

Fonte: Próprios autores

A Tabela 2 exibe as informações contidas em uma tabela de seções de um arquivo PE, ela descreve as seções do programa para que o Windows possa compreendê-lo corretamente para assim poder fazer bom uso dele. Abaixo, na Tabela 3, será abordado as descrições dessas seções:

Tabela 3 - Sections e suas descrições

Seção	Descrição
.text	Código executável, com o ponto de entrada do programa.
.data	Dados inicializados, como *strings.*
.rdata ou .idata	Tabela de importação com APIs do Windows e DLLs.
.reloc	Informações de realocação.
.rsrc	Recursos como imagens de interface.
.debug	Informações de depuração.

Fonte: Próprios autores

A Tabela 3 é autoexplicativa, ela exibe as principais seções contidas no arquivo PE juntamente com a descrição deles. A seguir, no próximo subtópico, será tratado o tema dos executáveis maliciosos.

2.4 EXECUTÁVEIS MALICIOSOS

De acordo com a Microsoft Corporation (2025), executáveis maliciosos ou *malwares*, são softwares projetados especificamente com a intenção de causar danos, roubar informações ou comprometer a integridade de dispositivos e redes. Eles frequentemente se disfarçam como arquivos legítimos, como executáveis ou documentos, induzindo o usuário a ativá-los inadvertidamente. O impacto pode variar de roubo de dados pessoais a interrupções graves em infraestruturas críticas. Os *malwares* são classificados por seu comportamento e impacto, sendo os principais conforme descrito na Tabela 4:

Tabela 4 - Malwares

Tipo de Malware	Descrição
Vírus	Programas que se replicam infectando arquivos ou programas legítimos, ativando-se quando o arquivo é executado, como o Melissa e o ILOVEYOU.
Worms	Autorreplicantes que se espalham por redes sem precisar de um arquivo host, explorando vulnerabilidades, são conhecidos por consumir recursos e instalar *backdoors* como o Conficker (2008), que infectou milhões de computadores explorando falhas no Windows.
Trojans	Disfarçam-se de software útil para ganhar acesso não autorizado, permitindo controle remoto ou instalação de outros malwares, trojans como Zeus podem roubar credenciais bancárias via *keylogging*.
Ransomware	Criptografa arquivos do usuário e exige pagamento para liberação. É uma das ameaças mais lucrativas e eficientes para cibercriminosos como o WannaCry (2017), que afetou infraestruturas globais, incluindo hospitais.
Spyware	Monitora atividades do usuário capturando dados como senhas ou histórico de navegação. *Spywares* também são usados como ferramentas de vigilância, citando *keyloggers* em relatórios de cibersegurança.
Rootkits	Os *rootkits* escondem a presença do malware em um dispositivo pelo máximo de tempo possível para que roube informações e recursos de modo contínuo, às vezes, até por anos.
Botnets	São redes de dispositivos infectados para ataques DDoS, eles são controlados remotamente por invasores, frequentemente usado para ataques em larga escala.

Fonte: Próprios autores

O ciclo de um executável malicioso tipicamente envolve quatro estágios, explicados na Tabela 5:

Tabela 5 - Etapas da infecção

Etapa	Descrição
Infecção	Entra via vetores como e-mails *phishing*, *downloads* infectados ou *exploits* de vulnerabilidades em softwares desatualizados.
Ativação	Dispara por ação do usuário como abrir um anexo ou gatilhos automáticos.
Propagação	Replica-se para outros arquivos ou redes.
Payload	Executa o dano principal, como roubo de dados ou criptografia.

Fonte: Próprios autores

Segundo o Rohr (2025), o primeiro vírus surgiu em 1986 infectando plataformas IBM PC utilizando mecanismos de ocultação, ele foi chamado de Cérebro Paquistanês e atacou a inicialização dos disquetes, o que permitiu que se propagasse em poucas semanas. Em seguida, nos anos 80, foi o Morris Worm, conhecido como o primeiro "verme" que se propagou em milhares de minicomputadores e estações de trabalho como VMS, BSD e SunOS.

Já na década de noventa foi o vírus Michelangelo, que infectou o setor de disquetes e o setor de MBR de discos rígidos. No ano de 1994 o primeiro *ransomware* foi denominado OneHalf embora nenhum resgate fosse exigido e não houvesse código de desativação, ascendeu a primeira série do setor de disco rígido. Se o FDISK / MBR fosse usado, o setor MBR era deletado, incapacitando o sistema de iniciar.

Em 1997, o *malware* auto propagação começou a ser substituído por *trojans*, a tendência de roubar credenciais de conta AOL assumiu diferentes formas e pressagiava o fenômeno do *phishing*. Nos anos 2000 foi um *worm* de e-mail conhecido como ILOVEYOU, que atacou dezenas de milhões de PCs Windows. Ele chegava como um anexo que se passava por uma carta de amor que quando aberto, os cibercriminosos acessavam o sistema operacional, o armazenamento de dados secundários e os dados da vítima.

Em 2005, nos encontramos CommWarrior, o primeiro *malware* para telefone móvel capaz de se espalhar por meio de mensagens MMS e Bluetooth. Ele atacou a linha de smartphones Symbian Series 60. Em 2008, surge o código malicioso Conflicker, que transforma computadores infectados em parte de uma *botnet*. Esta ameaça se propagou por muito tempo e infectou milhares de usuários. Em 2010, um

verme chamado Stuxnet marcou uma nova era de *malware* moderno, os mesmos que atacam sistemas de controle industrial e são usados contra instalações nucleares iranianas.

Em 2012 surge a Medre, uma ameaça que rouba informações extraindo documentos AutoCAD. Atualmente, nos deparamos com ameaças como Hesperbot, *trojan* bancário avançado que ataca usuários mediante campanhas de estilo *phishing*, que imitam organizações confiáveis. Assim, quando os atacantes percebem que a vítima executou o *malware*, eles roubam as credenciais da pessoa. Também encontramos com Windigo, que em 2014 assumiu o controle de vinte e cinco mil servidores Unix em todo o mundo e enviou milhões de mensagens de *spam* por dia, a fim de sequestrar servidores, infectar computadores e roubar informações.

Com a evolução da internet os *malwares* tem se tornado cada vez mais complexos e imprevisíveis ao longo do tempo, se propagando com facilidade pela mídia. Di Jorge afirma “O Dia Mundial da Internet é uma data para ser comemorada, mas também para refletirmos como os códigos maliciosos têm evoluído e se tornado mais sofisticados ao longo do tempo”, “Além de mais estruturados, os seus métodos de propagação e infecção são mais elaborados, e têm como principal objetivo o retorno econômico para o cibercriminoso”, finaliza o executivo. No subtópico seguinte, será tratado o tema *ransomware.*

2.4.1 RANSOMWARE

Ransomwares são uma forma específica de *malware*, eles são programas projetados para bloquear o acesso a dados ou sistemas de uma vítima, exigindo pagamento para restaurar o acesso. Nos primeiros ataques de *ransomware* eles simplesmente exigiam um resgate em troca de uma chave de criptografia para recuperar acesso aos dados, que seriam criptografados pelo criminoso.

Eles representam uma ameaça cibernética crescente, com impactos financeiros e operacionais significativos em indivíduos, empresas e governos. Diferente de outros *malwares*, o *ransomware* foca na extorsão direta, combinando criptografia de arquivos com ameaças de divulgação de dados roubados.

2.4.2 WANNACRY

Um grande incidente de segurança que atingiu organizações em todo o mundo foi o ataque de *ransomware* WannaCry. No dia doze de maio de 2017, o *worm* do *ransomware* WannaCry se propagou para mais de duzentas mil máquinas em mais de cento e cinquenta nações. FedEx, Honda, Nissan e o Serviço Nacional de Saúde (NHS) do Reino Unido são algumas das vítimas notáveis, sendo que este último teve que redirecionar algumas de suas ambulâncias para hospitais diferentes.

2.4.3 SEGURANÇA DA INFORMAÇÃO

Neste cenário caótico, surge a segurança da informação (SI) para detecção, prevenção e mitigação de ameaças cibernéticas, ela compreende um conjunto de ações estratégias para proteger sistemas, programas, equipamentos e redes de invasões.

Conforme Bastos ([s.d.]), o intuito central da segurança da informação é identificar, registrar e combater as ameaças, garantindo assim a proteção de dados e sistemas valiosos de possíveis violações ou ataques.

> “Segurança da informação é a proteção de informações importantes contra acesso não autorizado, divulgação, uso, alteração ou interrupção. Ajuda a garantir que os dados organizacionais confidenciais estejam disponíveis para usuários autorizados, permaneçam confidenciais e mantenham sua integridade”.

A SI (Segurança da Informação) possui estratégias e práticas fundamentais baseadas em três pilares principais também conhecida pela sigla CID: confidencialidade, integridade e disponibilidade. Entretanto, com o desenvolvimento da tecnologia outros pilares foram surgindo, resultando na autenticidade, irretratabilidade e conformidade, totalizando em seis pilares principais.

Confidencialidade significa garantir que as informações sejam acessíveis somente por pessoas, processos ou sistemas autorizados. Disponibilidade significa assegurar que informações e sistemas estejam acessíveis e operacionais quando necessários, por usuários legítimos. Integridade significa garantir que as informações não sejam alteradas, corrompidas ou modificadas de forma não autorizada. Autenticidade significa comprovar que a informação, o usuário ou o sistema é genuíno e confiável. Irretratabilidade ou Não Repúdio, significa impedir que autor ou receptor neguem uma ação ou transação já realizada. Conformidade significa garantir que

todos os processos, sistemas e dados estejam em conformidade com ad leis, normas e regulamentos.

Os malwares representam uma ameaça significativa no cenário digital, com impactos que vão além do financeiro, afetando a privacidade e a confiança nas tecnologias. A segurança da informação, com suas práticas e tecnologias, é essencial para proteger sistemas e dados, garantindo a continuidade dos negócios e a proteção dos usuários. Investir em prevenção, educação e resposta rápida a incidentes é crucial para mitigar os riscos. À medida que as ameaças evoluem, as estratégias de segurança também devem se adaptar, incorporando inovações tecnológicas e políticas eficazes.

No subtópico a seguir será introduzido o Dataset EMBER, usado na realização do experimento prático desse trabalho.

2.5 EMBER DATASET

O Dataset EMBER é um conjunto de dados de código aberto e gratuito comumente utilizado como base a diversos treinamentos de *machine learning* para reconhecimento e classificação de *malwares*. Esse *dataset* é fruto de um conjunto de diversas amostras de arquivos executáveis PE (Roth, 2022).

A versão de 2018 do Dataset EMBER, que será usada neste trabalho, conta com mais de um milhão de amostras de executáveis PE digitalizados até o ano de 2018. Dentre as amostras para treinamento que compõe o Dataset EMBER, é indicado na Tabela 6:

Tabela 6 - Amostras para treinamento

Tipo de Amostra	Quantidade Aproximada	Observação
Maliciosas	300.000	Amostras maliciosas
Benignas	300.000	Amostras benignas
Não rotuladas	300.000	Amostras sem rótulo

Fonte: Roth, 2018

A Tabela 7 se refere às amostras que compõem os testes:

Tabela 7 - Amostras para testes

Classificação	Quantidade de Amostras
Malignas	100.000
Benignas	100.000

Fonte: Roth, 2018

Além disso, a Tabela 8 exibe do que cada amostra no *dataset* é composta por:

Tabela 8 - Estruturas das amostras

Atributo	Descrição
Hash SHA256 do arquivo	Identificador único da amostra
Data da primeira aparição	Data em que o arquivo surgiu
Rótulo da classificação	Indicação se é maligna ou benigna
Features extraídas	Conjunto de características coletadas da amostra

Fonte: Phil Roth, 2018

2.5.1 FUNCIONALIDADES DA BIBLIOTECA EMBER

A biblioteca EMBER utilizada no Python compõe diversas funcionalidades para a manipulação do *dataset* e manipulação do modelo de treinamento, entre elas pode-se citar, na Tabela 9:

Tabela 9 - Funções do Dataset

Função	Descrição
create_metadata(data_dir)	Escreve os metadados em um arquivo CSV e retorna o *dataframe* dele.
create_vectorized_features(data_dir, feature_version=2)	Cria os *features vectors* de um arquivo de features e os escreve no disco.
read_metadata(data_dir)	Lê um arquivo de metadados já criado e retorna o *dataframe*.
read_vectorized_features(data_dir, subset=None, feature_version=2)	Lê as features vetorizadas e carrega como *numpy arrays* dentro da memória.
predict_sample(lgbm_model, file_data, feature_version=2)	Prevê um arquivo PE com base no modelo LightGBM.
train_model(data_dir, params={}, feature_version=2)	Treina o modelo LightGBM do Dataset EMBER a partir de *vectorized features*.

Fonte: Próprios autores

As funcionalidades acima permitem automatizar e padronizar o processo de extração de dados e treinamento de modelos, reduzindo a complexidade técnica para pesquisadores e profissionais que desejam avaliar técnicas de detecção de *malwares* baseadas em aprendizado de máquina. Agora será abordada a metodologia de desenvolvimento utilizada no trabalho juntamente com o experimento prático realizado.

3 METODOLOGIA DE DESENVOLVIMENTO

Neste capítulo, são descritas as etapas da metodologia realizadas para a realização da parte prática, que consistiu na leitura de um *dataset* e no treinamento de um modelo de classificação de amostras de *ransomware*. O processo abrange desde a preparação do ambiente até a instalação de dependências e do módulo EMBER para o Python.

3.1 FERRAMENTAS E TECNOLOGIAS UTILIZADAS

Para a elaboração deste trabalho, foram utilizados ambientes e módulos com o foco em favorecer o funcionamento e compatibilidade adequados do ambiente e dos códigos que foram empregados, já que, para isso, foram necessárias as versões corretas de determinados módulos para a execução das etapas. As principais tecnologias utilizadas são exibidas na Tabela 10:

Tabela 10 – Tecnologias utilizadas

Ferramenta / Biblioteca / Sistema	**Descrição**
Python 3.6	Para o funcionamento desta metodologia, foi estritamente necessário utilizar a versão correta do Python. Algumas de suas atualizações acabaram gerando incompatibilidades com algumas partes do código.
Miniconda3	Por conta de sua flexibilidade e fácil configuração de módulos em versões especificas, foi utilizado o Miniconda3. Sua flexibilidade permite a criação de um ambiente virtual com a versão do Python necessária para o trabalho.
EMBER	Um dos módulos principais, que incluiu funcionalidades para leitura de dados, extração de features e treinamento de modelos usando o Dataset EMBER, um *dataset* de código aberto e gratuito para uso em sua versão de 2018.
LightGBM	Biblioteca de aprendizado de máquina baseada em árvores de decisão, que foi usada para o treinamento e teste do modelo.
SKLEARN	Biblioteca de machine learning que oferece suporte à avaliação do modelo, divisão de dados em treino e teste, e outras funções auxiliares como métricas de desempenho.
MATPLOITLIB	Biblioteca utilizada para visualização de dados e geração de gráficos, incluindo a exibição da árvore de decisão treinada.
Linux Mint no VirtualBox	Para este trabalho foi utilizado o sistema operacional Linux Mint, na sua versão 22.2 XFCE, virtualizado no programa VirtualBox. O mesmo processo pode ser feito no Windows, porém com algumas diferenças que não foram abordadas nesse trabalho.

Fonte: Próprios autores

3.2 REQUISITOS DE HARDWARE

Para garantir o melhor funcionamento é preciso planejar adequadamente o *hardware* disponível. O treinamento de modelo com o EMBER pode consumir muitos recursos, especialmente na extração de características e treinamento do modelo. A Tabela 11 apresenta os requisitos de hardware aproximados:

Tabela 11 – Especificações de Hardware

Componente	Requisito Mínimo	Recomendado para Desempenho Ideal	Observações
CPU	4 núcleos (Intel i5 / Ryzen 5)	8+ núcleos (Intel i7/i9, Ryzen 7/9, Xeon)	O pré-processamento do EMBER é intensivo em CPU; mais núcleos reduzem o tempo de extração de features.
GPU	Opcional (para modelos baseados em árvore)	GPU com 8GB+ VRAM (NVIDIA RTX 3060 ou superior)	O EMBER normalmente é usado com LightGBM/XGBoost (CPU), mas redes neurais (PyTorch/TensorFlow) se beneficiam muito da GPU.
Memória RAM	8 GB	16–32 GB	O *dataset* completo (~1 milhão de amostras) pode consumir bastante RAM durante o treinamento e validação.
Armazenamento	50 GB HDD	100 GB SSD NVMe	O Dataset EMBER 2018 ocupa ~2–3 GB, mas o espaço extra é necessário para versões intermediárias e checkpoints. SSD acelera leitura/escrita.
Sistema Operacional	Windows 10 / Linux (Ubuntu 20.04+)	Linux (Ubuntu 22.04 LTS ou superior)	Linux oferece melhor compatibilidade com *frameworks* de *machine learning* e bibliotecas otimizadas.

Fonte: Próprios autores

3.3 ANACONDA E JUPYTER NOTEBOOK

Neste trabalho, foram utilizados o Anaconda e o Jupyter Notebook (ambos do Miniconda3) por conta da flexibilidade provida por eles em relação a versão de módulos e da linguagem Python.

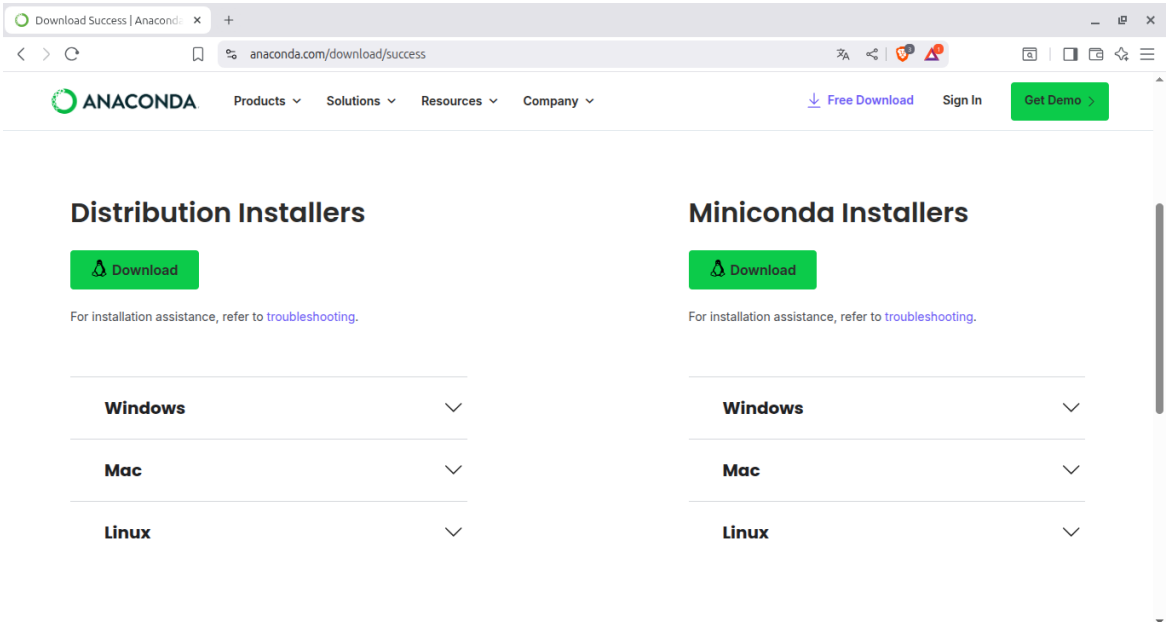
3.4 CONFIGURAÇÃO DO AMBIENTE

Nesta seção foi abordada a configuração do ambiente para a realização da leitura do *dataset* e treinamento do modelo. Todos os passos a partir daqui foram seguidos com rigor, pois qualquer erro poderia comprometer toda a execução do processo.

3.4.1 MINICONDA3

A primeira etapa consistiu no *download* e instalação do ambiente, que pôde ser feito pelo seguinte endereço: “https://www.anaconda.com/download/success”, conforme na Figura 1:

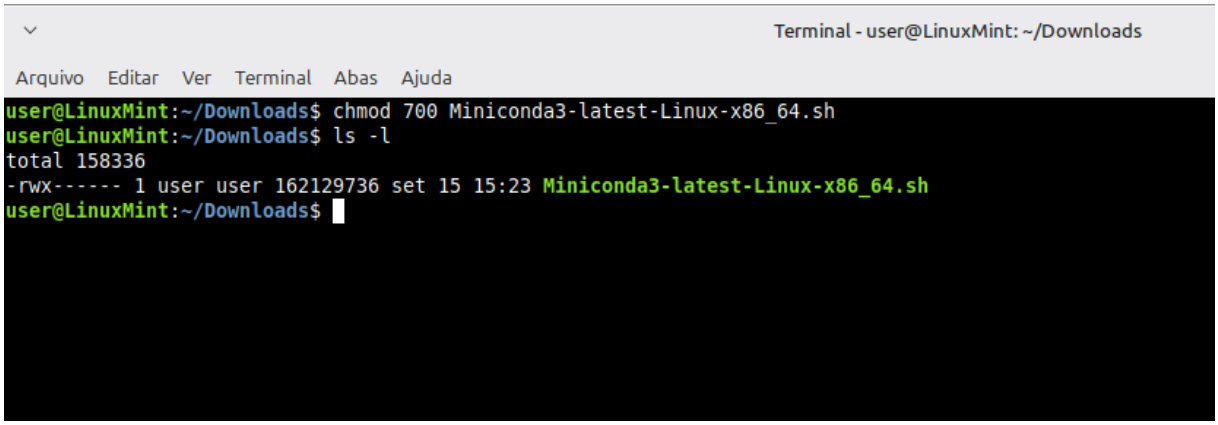
Figura 1 - Site Anaconda



Fonte: Próprios autores a partir do site oficial do Anaconda

Foi realizado o *download* do programa e, feito isso, acessou-se a pasta onde ele foi salvo e feita a alteração de permissão necessária com o comando: “chmod 700 <arquivo-baixado>” e, após isso, ele foi instalado, usando-se “./<arquivo-baixado>”. Ressalta-se que foi utilizado “<>” para sinalizar o arquivo, pois é possível haver diferenças nos nomes dos arquivos. Nesse exemplo, conforme a Figura 2, o nome do arquivo instalado foi “Miniconda3-latest-Linux-x86_64.sh”:

Figura 2 - Permissão e instalação



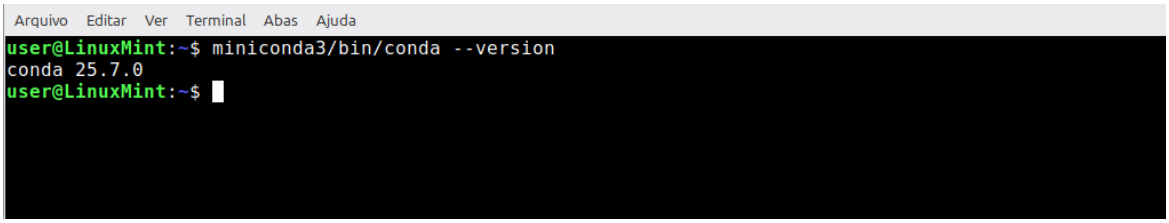
Fonte: Próprios autores a partir do VirtualBox

Após se executar o *script* de instalação, foram apenas seguidas as instruções exibidas no terminal, que consistiam, no geral, em aceitar os termos de uso e definir o diretório de instalação. Agora será tratada a criação do ambiente virtual de forma detalhada.

3.4.2 CRIANDO O AMBIENTE VIRTUAL

Logo após a instalação do Miniconda3, foi gerado um diretório na pasta usada para a instalação, a qual foi “/home/user/miniconda3”. Esse diretório continha todos os executáveis importantes para se executar o Anaconda. Para começar, foi conferido se o comando “conda” estava funcionando, com o uso de “miniconda3/bin/conda –version”, indicado na Figura 3:

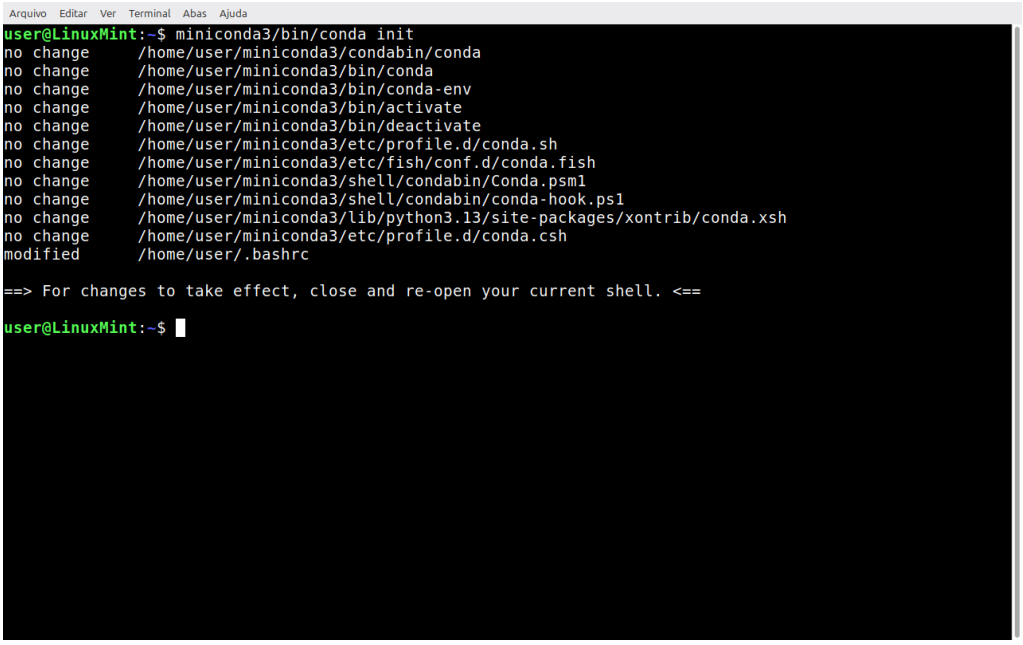
Figura 3 - Conferindo a versão



Fonte: Próprios autores a partir do VirtualBox

Ao se confirmar o funcionamento do comando “conda”, ele foi inicializado o para a interação com o *shell*, usando-se “miniconda3/bin/conda init”, processo exibido na Figura 4:

Figura 4 – Inicialização



Fonte: Próprios autores fazendo uso do VirtualBox

A seguir, foi adicionado algumas linhas de configuração no arquivo “/home/user/.bashrc” e, para que essas modificações funcionassem, ele foi recarregado com o comando “source ~/.bashrc”, mostrado na Figura 5:

Figura 5 - Atualizando o .bashrc

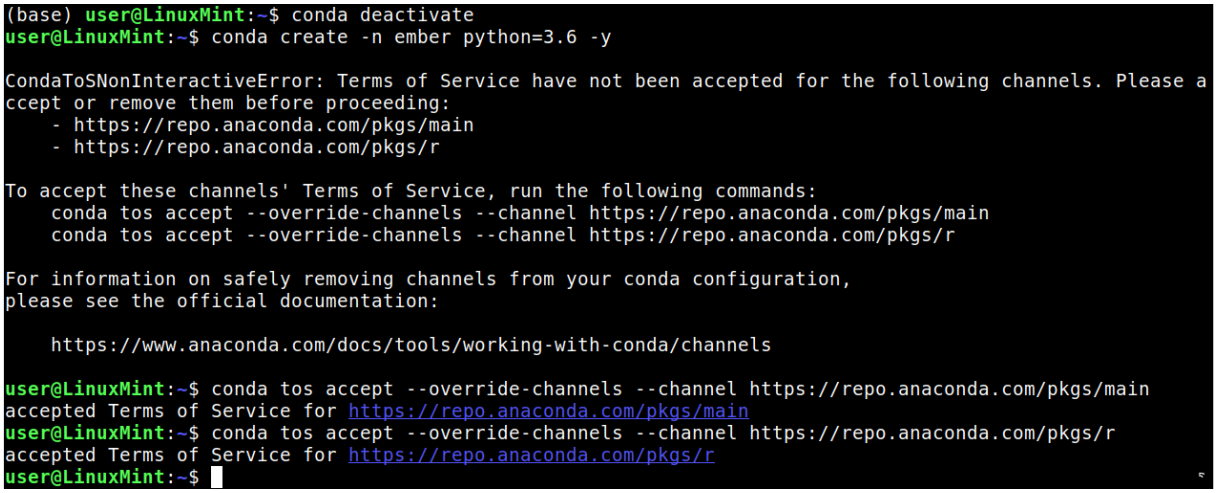
Fonte: Próprios autores usando o VirtualBox

Ao atualizar-se o “~/.bashrc”, o ambiente padrão do Conda, de nome “base” é inicializado. Como a metodologia requer módulos em determinadas versões além do próprio Python, para que não houvesse conflitos entre as dependências a serem instaladas, os ambientes e seus módulos utilizados foram criados de forma isolada, assim evitando possíveis conflitos e problemas.

Realizada essa parte, foi executado o primeiro comando para sair do ambiente padrão do Conda e criar o ambiente: “conda deactivate”.

Se os passos anteriores foram executados corretamente, os comandos que começavam com “miniconda3/bin/conda” foram descartados. No entanto, caso os termos de uso não tenham sido aceitos previamente, foi gerado um erro ao tentar criar o primeiro ambiente. Esse erro solicitava a aceitação dos termos de uso, que pôde ser feita com a execução do comando sugerido pelo próprio Anaconda. Tal situação é ilustrada na Figura 6:

Figura 6 – Termos de uso



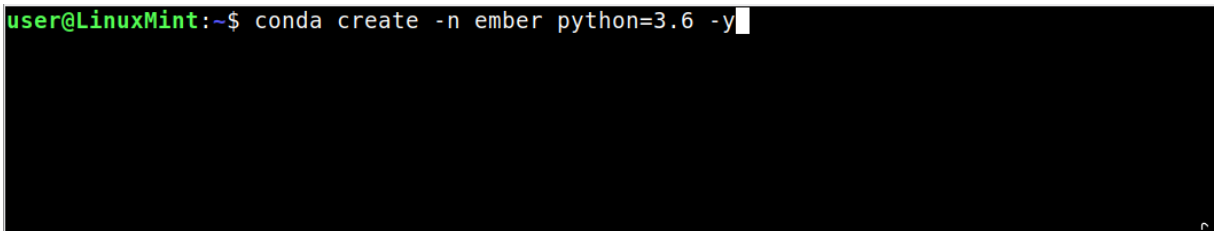
Fonte: Próprios autores a partir do VirtualBox

Ao executar o comando para sair do ambiente base, foi possível confirmar a mudança observando o terminal antes e depois da execução. Durante a criação do primeiro ambiente sem a aceitação prévia dos termos, foi retornado um erro. Esse erro, no entanto, apresentou uma solução simples, bastando executar os comandos sugeridos pelo próprio Anaconda “conda tos accept --override-channels --channel https://repo.anaconda.com/pkgs/main“ e “conda tos accept --override-channels –channel https://repo.anaconda.com/pkg/r” .

Logo após se aceitar os termos, foi usado novamente o comando para criar o ambiente, nesse momento a versão do Python que será instalada foi escolhida, um passo que é fundamental para o funcionamento do trabalho. Caso a versão do Python não seja passada no comando, ou caso se altere a versão, isso pode acabar comprometendo a execução do projeto. O ambiente foi criado usando: “conda create -n ember python=3.6 -y”.

O parâmetro “-n” especifica o nome do ambiente que será criado, enquanto o parâmetro “-y” instrui o instalador a confirmar automaticamente todas as solicitações durante o processo de instalação. A seguir iniciou-se a criação do ambiente virtual, conforme mostrado na Figura 7:

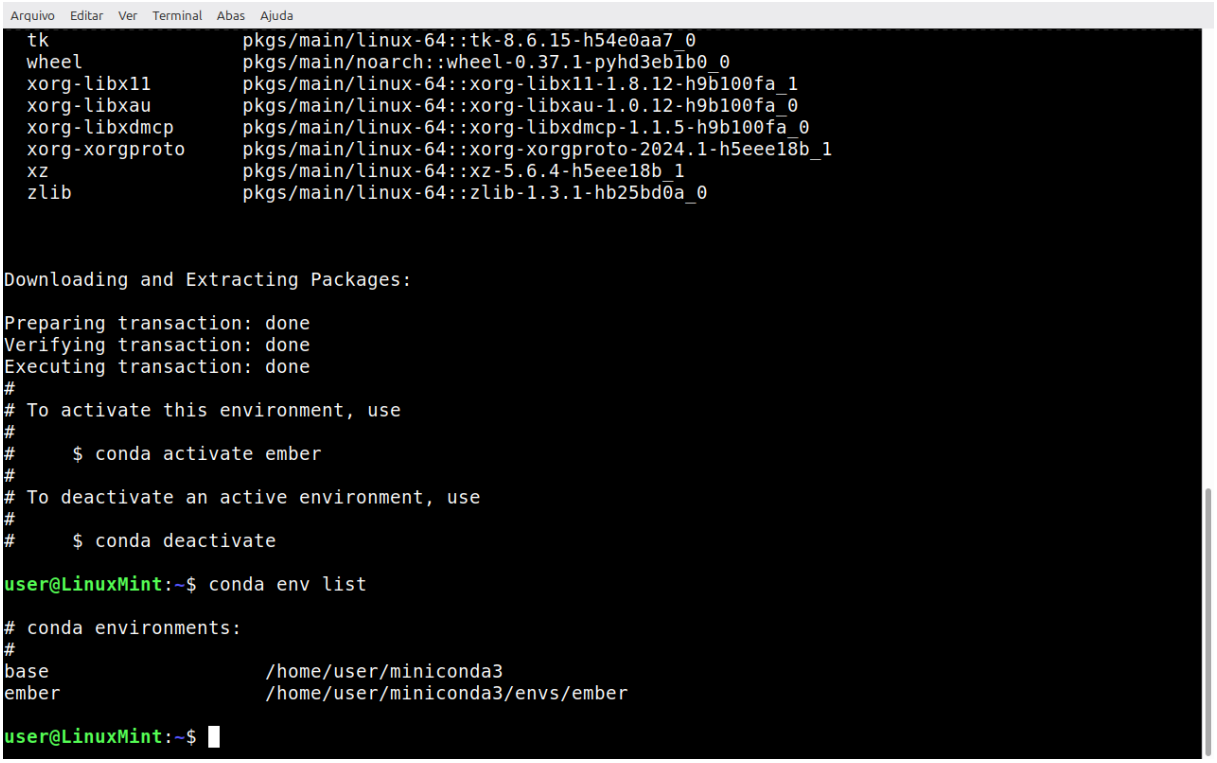
Figura 7 - Criando o ambiente virtual



Fonte: Próprios autores se usando o VirtualBox

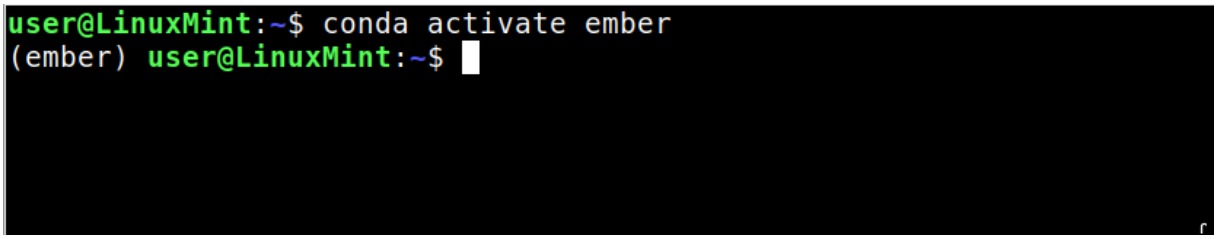
Após a criação do ambiente, foi instalado tudo o que é necessário para o funcionamento. Em seguida, foi verificado se o ambiente havia sido criado corretamente por meio do comando: “conda env list”, processo exibido na Figura 8 e 9:

Figura 8 – Criando o ambiente virtual e conferindo



Fonte: Próprios autores fazendo uso do VirtualBox

Figura 9 - Ativando o ambiente



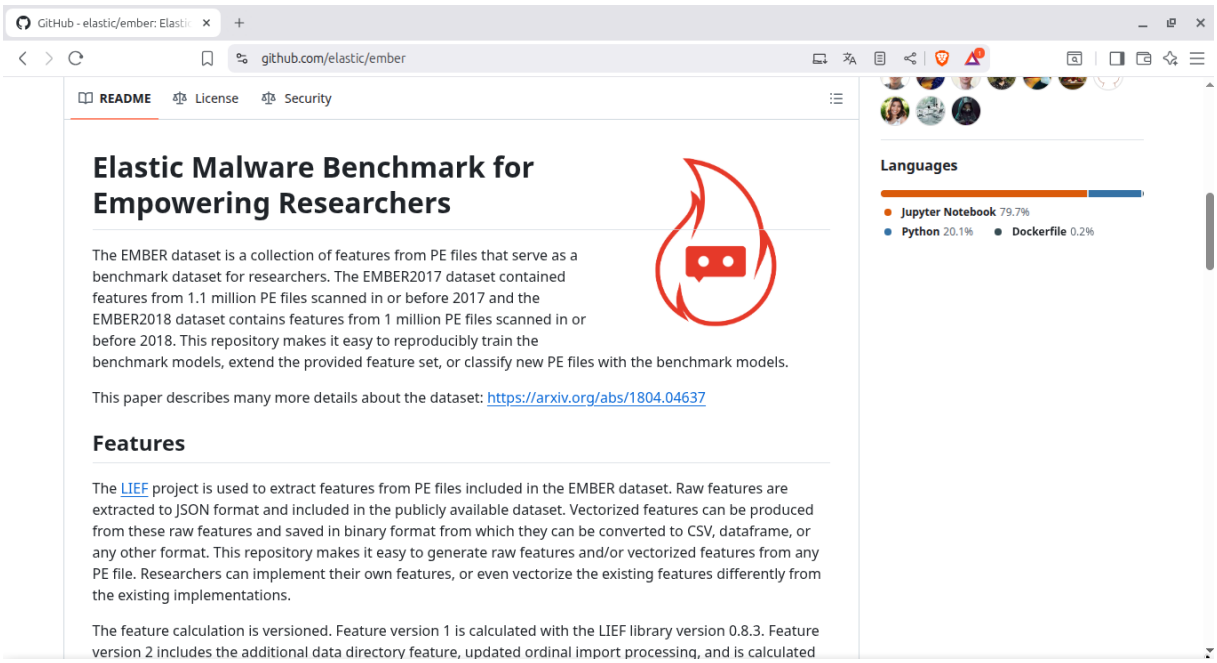
Fonte: Próprios autores com o uso do VirtualBox

Após a execução correta do comando, foi observado, por meio do terminal, que o ambiente EMBER havia sido ativado com sucesso. Agora será abordado a respeito do seu repositório oficial.

3.4.3 REPOSITÓRIO EMBER

O módulo EMBER, que faz uso do Dataset EMBER, abordado anteriormente, juntamente com suas dependências, está disponível no repositório oficial da Elastic no GitHub, acessível por meio do seguinte endereço: “https://github.com/elastic/ember”, conforme exibido na Figura 10.

Figura 10 - Github EMBER



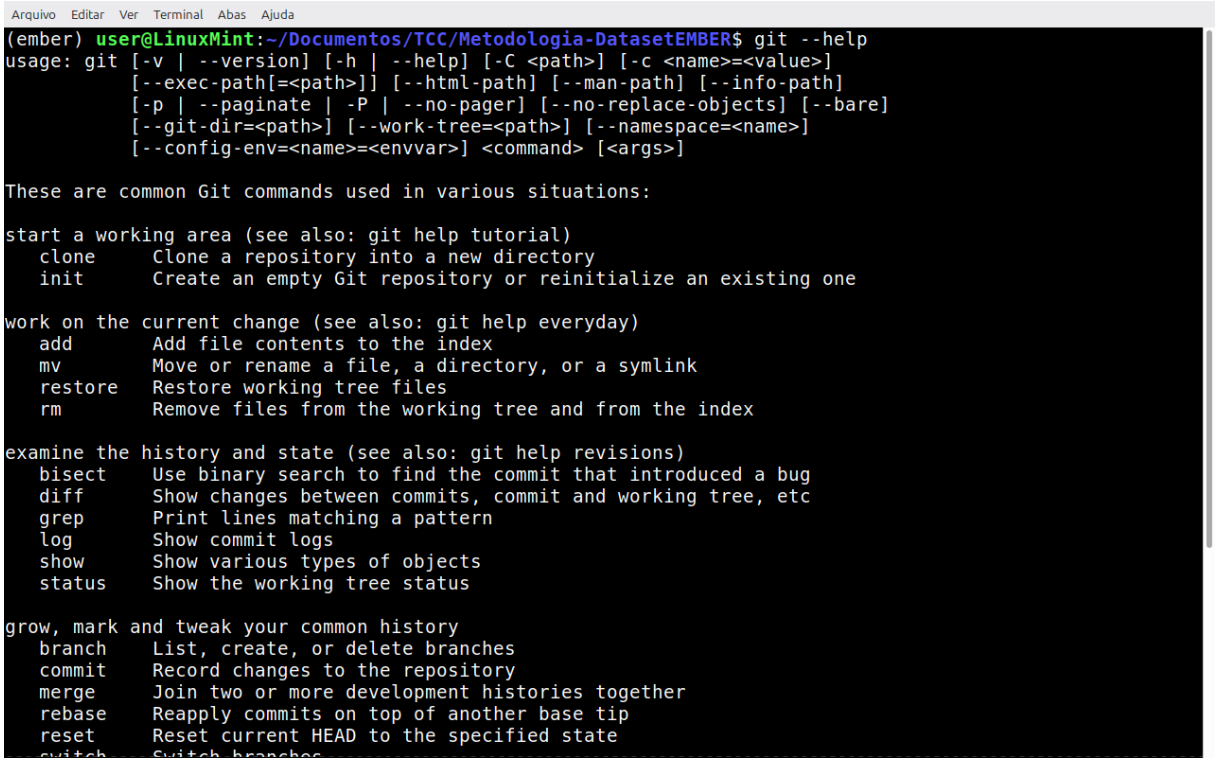
Fonte: Próprios autores a partir da página oficial do EMBER

Por meio do terminal, foi utilizado o comando “git” para clonar o repositório do EMBER. Caso o Git não estivesse previamente instalado, isso poderia ser feito com o comando: “sudo apt update && sudo apt install git”.

A partir deste ponto, como os comandos passaram a manipular arquivos e diretórios específicos, foi necessário ter atenção redobrada quanto aos caminhos utilizados. Para verificar o diretório atual, utilizou-se o comando: “pwd”.

Feito isso, foi conferido se o Git foi instalado com sucesso e em seguida clonado o repositório “git –help” e do EMBER, por meio do comando ”git clone https://github.com/elastic/ember.git“, indicado na Figura 11 e 12:

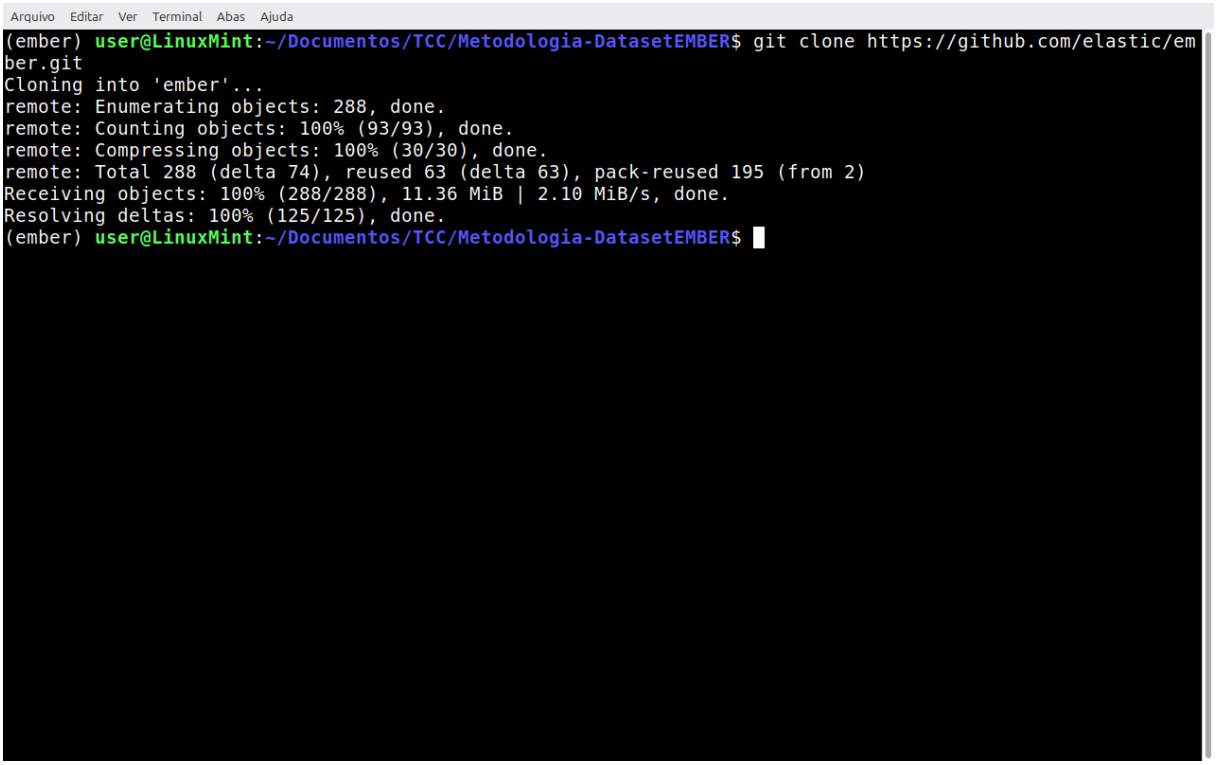
Figura 11 - Conferindo o Git



Fonte: Próprios autores a partir do VirtualBox

O procedimento se segue na Figura 12:

Figura 12 - Clonando o repositório



Fonte: Próprios autores a partir do VirtualBox

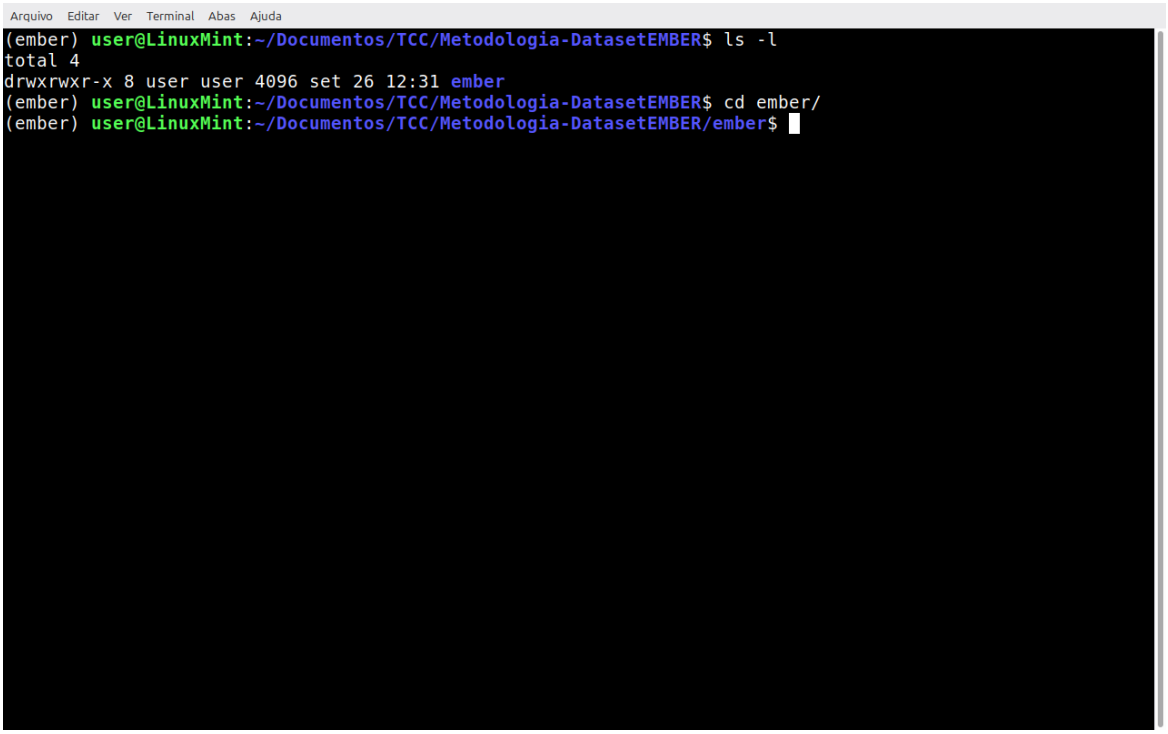
Agora, no próximo subtópico, será instalado o módulo do EMBER juntamente com suas dependências.

3.4.4 INSTALANDO MÓDULO EMBER E DEPENDÊNCIAS

Nesta etapa, foi realizada a instalação do módulo EMBER para Python, juntamente com suas dependências. A correta execução desse procedimento foi fundamental, uma vez que a ausência de alguma dependência, ou a utilização de versões incompatíveis, comprometeria a execução do projeto.

Dentro do ambiente EMBER, verificou-se se o repositório havia sido clonado corretamente, utilizando o comando “ls -l“ e em seguida acessado o diretório do repositório com ”cd ember/”, conforme demonstrado na Figura 13:

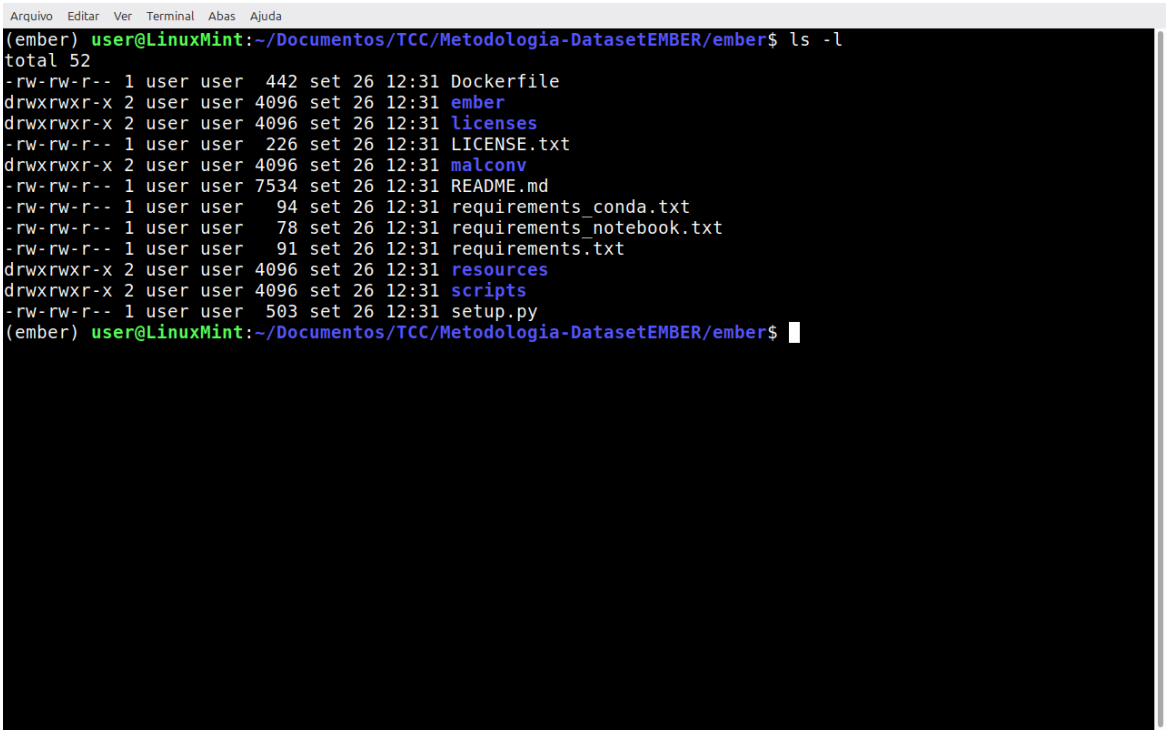
Figura 13 – Diretório EMBER



Fonte: Próprios autores usando o VirtualBox

Foi então conferido o conteúdo do diretório com “ls -l”, exibido na Figura 14:

Figura 14 – Conteúdo do diretório



Fonte: Próprios autores usando-se o VirtualBox

No diretório do EMBER, encontram-se três arquivos essenciais: “requirements_conda.txt”, “requirements_notebook.txt” e “setup.py”, sendo o último o principal responsável pela instalação do módulo EMBER, enquanto os dois arquivos restantes especificam as dependências necessárias. Estes arquivos de dependências incluem módulos essenciais para o treinamento do modelo e algoritmos de árvore de decisão, como LightGBM, Scikit-learn, Matplotlib, entre outros. A seguir, na Tabela 12, descritos alguns dos comandos utilizados para se instalar as dependências do ambiente e o módulo EMBER.

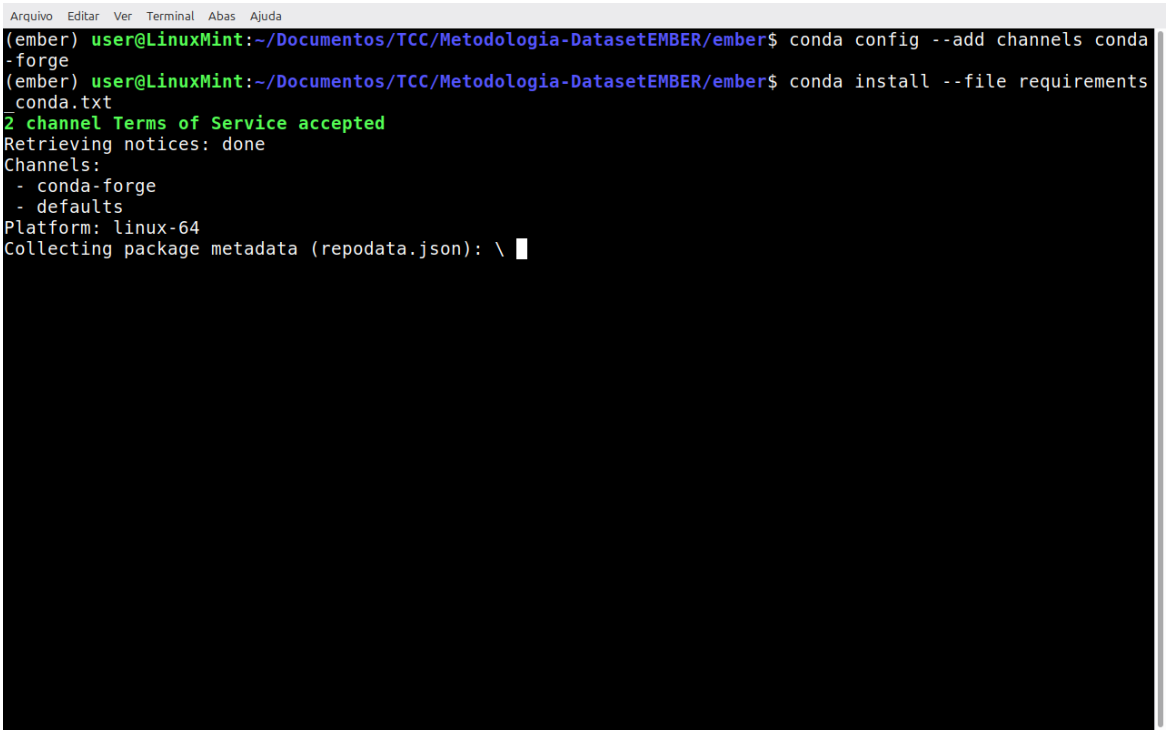
Tabela 12 - Comandos utilizados na instalação do módulo EMBER e dependências

Comando	Descrição
conda config --add channels conda-forge	Adiciona o repositório Conda-forge como fonte de pacotes. Ele é uma comunidade que mantém pacotes atualizados e mais variados do que os disponíveis no canal padrão.
conda install --file requirements_conda.txt	Instala todos os módulos contidos no arquivo “requirements_conda.txt”.
conda install --file requirements_notebook.txt	Instala todos os módulos existentes no arquivo “requirements_notebook.txt”.
python setup.py install	Instala o projeto EMBER como um pacote Python.

Fonte: Roth, 2022

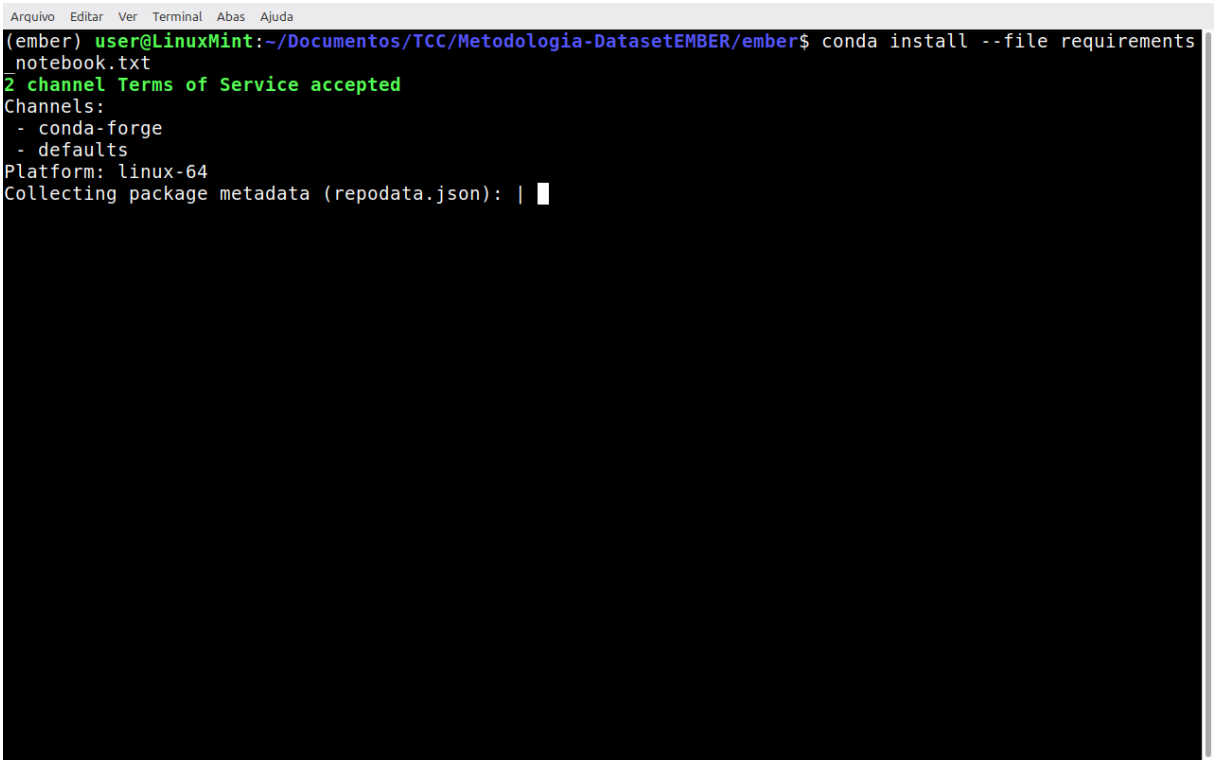
Na Figura 15 e 16, é ilustrado o processo de instalação das dependências necessárias descritas anteriormente na Tabela 12, com o intuito de garantir o funcionamento correto do ambiente:

Figura 15 - Instalando dependências



Fonte: Próprios autores fazendo uso do VirtualBox

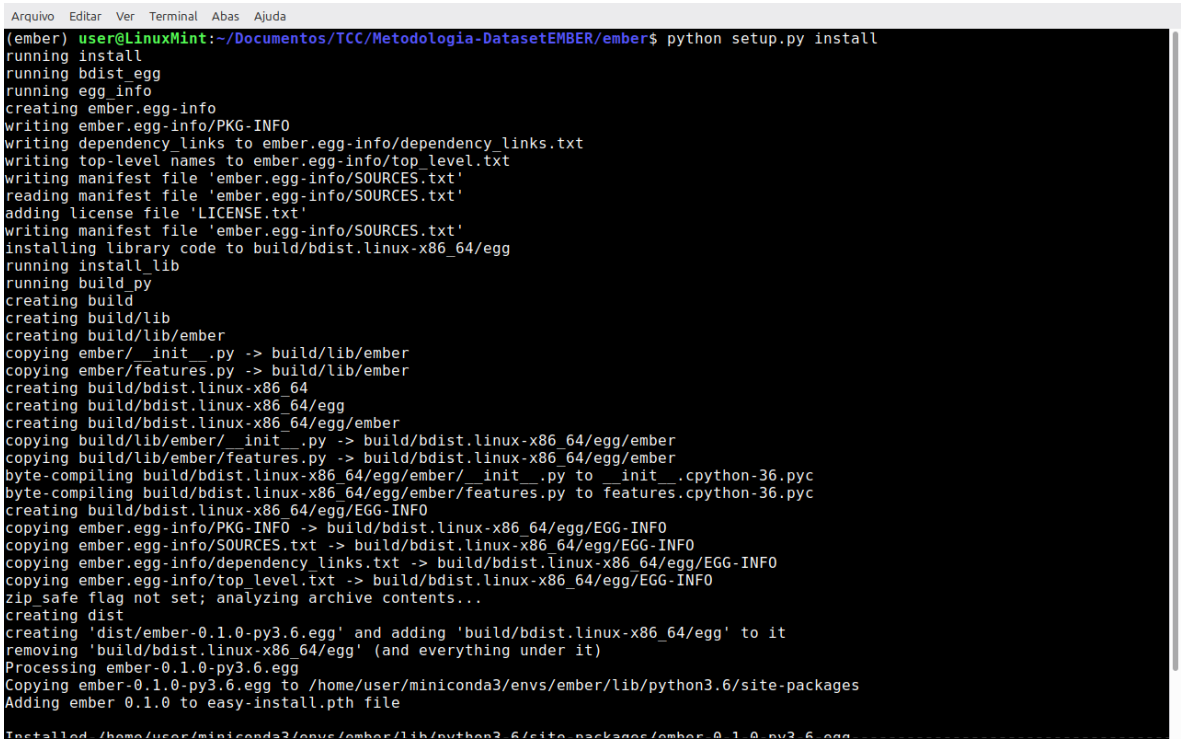
Figura 16 - Instalando as demais dependências



Fonte: Próprios autores a partir do VirtualBox

Após a instalação correta dos pacotes essenciais, foi instalado o módulo EMBER, processo exibido na Figura 17:

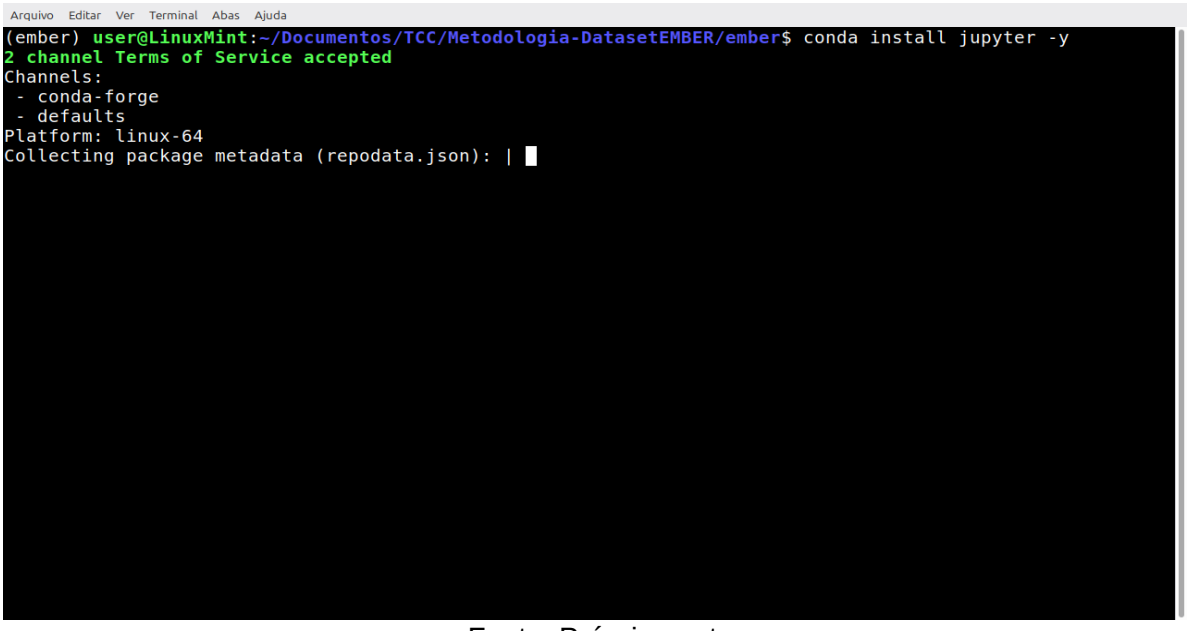
Figura 17 – Instalação do módulo EMBER



Fonte: Próprios autores no VirtualBox

Após a instalação dos arquivos de dependências e do pacote EMBER, foi instalada a interface de programação Jupyter Notebook por meio do comando ”conda install jupyter -y”, conforme a Figura 18:

Figura 18 - Instalando o Jupyter Notebook



Fonte: Próprios autores

A seguir, será abordado sobre o *dataset* escolhido e utilizado na realização do trabalho, juntamente com outros detalhes sobre ele.

3.4.5 DATASET UTILIZADO

Após instalado o pacote EMBER, além das dependências, foi feito o download do *dataset* localizado no próprio Github "https://ember.elastic.co/ember_dataset_2018_2.tar.bz2".

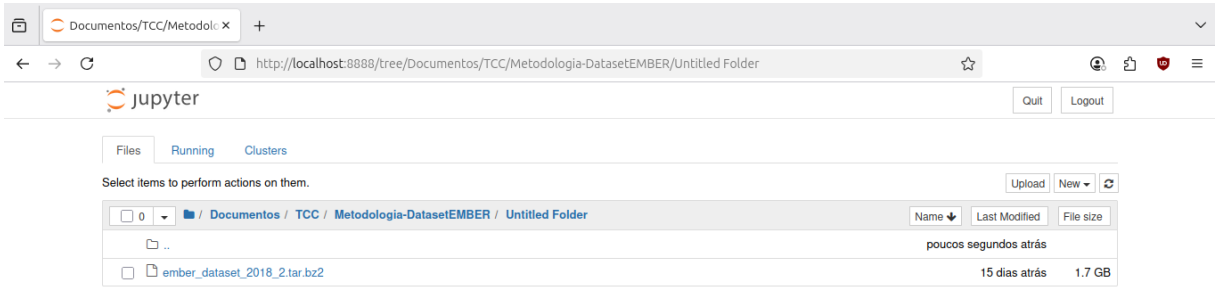
O *dataset* usado neste trabalho é a segunda versão, de 2018, devido ao fato de ele ser de código aberto e gratuito para uso. Para garantir o correto funcionamento dos códigos desenvolvidos nos tópicos seguintes, o *dataset* foi mantido no diretório em que esses códigos estavam localizados. Agora será iniciada a extração de vetores numéricos.

3.5 EXTRAINDO VETORES NÚMERICOS

A partir desse ponto, iniciou-se a utilização dos códigos desenvolvidos. Foi verificado que todos os procedimentos anteriores foram executados corretamente. Dentro do ambiente virtual, o Jupyter Notebook foi aberto dentro do diretório onde o para facilitar a execução dos *scripts*. Para isso, o comando foi executado no terminal no diretório onde o *dataset* havia sido baixado: "jupyter notebook".

Após digitar o comando, foi aberto uma guia no navegador com o Jupyter Notebook, conforme mostrado na Figura 19:

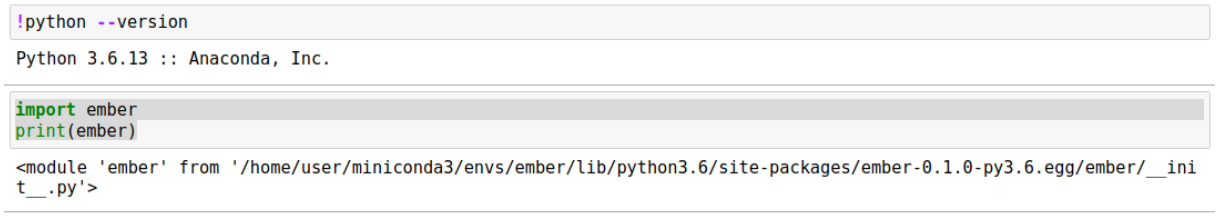
Figura 19 – Guia no navegador com Jupyter Notebook



Fonte: Próprios autores

Após aberto o Jupyter Notebook, foi selecionada a opção “Python 3” no menu “New”, criando-se um documento para a inserção dos códigos. Em seguida, verificou-se se os procedimentos anteriores de configuração foram executados corretamente por meio do comando “!python –version”, usado para verificar a versão do Python, que nesse cenário foi a 3.6, conforme a Figura 20:

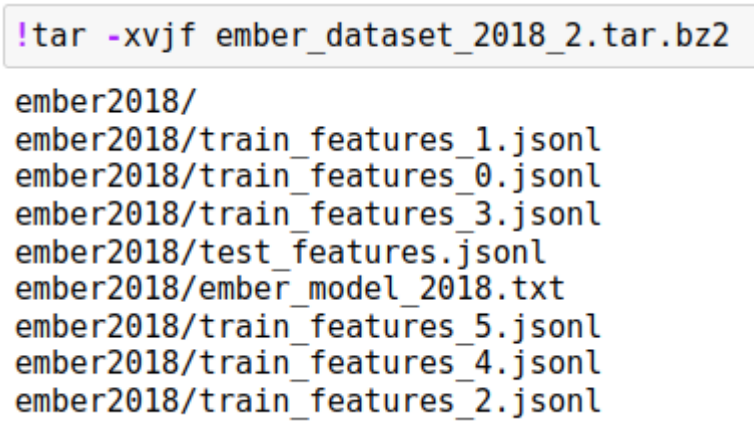
Figura 20 - Testando versão e módulo EMBER



Fonte: Próprios autores a partir da tela do Jupyter Notebook

Agora, dentro de uma nova célula, foi extraído o *dataset* com o seguinte comando: “!tar -xvjf <dataset>”. Este processo levou algum tempo, devido ao tamanho considerável do *dataset*, conforme exibido na Figura 21:

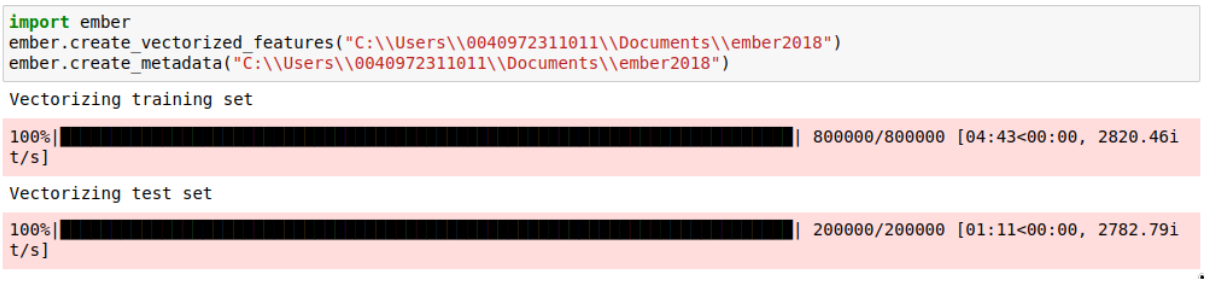
Figura 21 - Extraindo o *dataset*



Fonte: Próprios autores na tela do Jupyter Notebook

Após se extrair, foi executada a seguinte célula de código exibida na Figura 22:

Figura 22 - Vetorização e Geração de Metadados



Fonte: Próprios autores a partir do Jupyter Notebook

Esse passo é obrigatório antes de treinar qualquer modelo, pois os dados precisam estar "vetorizados" (transformados de arquivos binários em vetores numéricos). Os diretórios usados dentro das funções devem ser os mesmos onde o *dataset* estava localizado.

Figura 23 – *Dataframe* obtido após a vetorização

	sha256	appeared	label	avclass	subset
0	0abb4fda7d5b13801d63bee53e5e256be43e141faa077a...	2006-12	0		train
1	c9cafff8a596ba8a80bafb4ba8ae6f2ef3329d95b85f15...	2007-01	0		train
2	eac8ddb4970f8af985742973d6f0e06902d42a3684d791...	2007-02	0		train
3	7f513818bcc276c531af2e641c597744da807e21cc1160...	2007-02	0		train
4	ca65e1c387a4cc9e7d8a8ce12bf1bcf9f534c9032b9d95...	2007-02	0		train
...	...	...	...	...	...
999995	e033bc4967ce64bbb5cafdb234372099395185a6e0280c...	2018-12	1	zbot	test
999996	c7d16736fd905f5fbe4530670b1fe787eb12ee86536380...	2018-12	1	flystudio	test
999997	0020077cb673729209d88b603bddf56b925b18e682892a...	2018-12	0		test
999998	1b7e7c8febabf70d1c17fe3c7abf80f33003581c380f28...	2018-12	0		test
999999	836063f2312b597632bca1f738e68e4d23f672d587a7fc...	2018-12	1	emotet	test

1000000 rows × 5 columns

Fonte: Próprios autores no Jupyter Notebook

A Figura 23 representa o principal *dataframe* com um milhão de amostras que é exibida após a vetorização dos dados. Agora será abordado o carregamento de dados vetorizados e metadados no subtópico a seguir.

3.6 CARREGAMENTO DOS DADOS VETORIZADOS E METADADOS

Nesta etapa, foram carregados os vetores de características “(X_train, X_test)” e os rótulos “(y_train, y_test)” previamente extraídos, além do *dataframe* de metadados contendo informações como *hash*, data de aparecimento, rótulo e tipo de *malware*, conforme exibido na Figura 24:

Figura 24 - Carregamento dos Dados Vetorizados e Metadados

```
import ember
X_train, y_train, X_test, y_test = ember.read_vectorized_features("C:\\Users\\0040972311011\\Documents\\ember2018")
metadata_dataframe = ember.read_metadata("C:\\Users\\0040972311011\\Documents\\ember2018")
```

Fonte: Próprios autores fazendo uso do Jupyter Notebook

Agora, com toda a preparação do ambiente concluída, será iniciado o treinamento de um modelo para teste.

3.7 TREINANDO O MODELO

Para treinar o modelo preditivo, foi utilizado a função “train_model” do pacote EMBER, que implementa um classificador LightGBM. O conjunto de treinamento contém seiscentas mil amostras balanceadas entre arquivos maliciosos e benignos, com mais de duas mil características extraídas de cada arquivo PE. Isso pode ser visto na Figura 25:

Figura 25 – Treinando o modelo

```
import ember
lgbm_model = ember.train_model("C:\\Users\\0040972311011\\Documents\\ember2018")
```

```
[LightGBM] [Info] Number of positive: 300000, number of negative: 300000
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 4.725833 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 212057
[LightGBM] [Info] Number of data points in the train set: 600000, number of used features: 2333
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
```

Fonte: Próprios autores

A função realizou o ajuste dos parâmetros internos do modelo, otimizando sua capacidade de identificar padrões que discriminam *malwares*. No próximo subtópico, será abordada a árvore de decisão gerada a partir do modelo treinado.

3.8 ÁRVORE DE DECISÃO

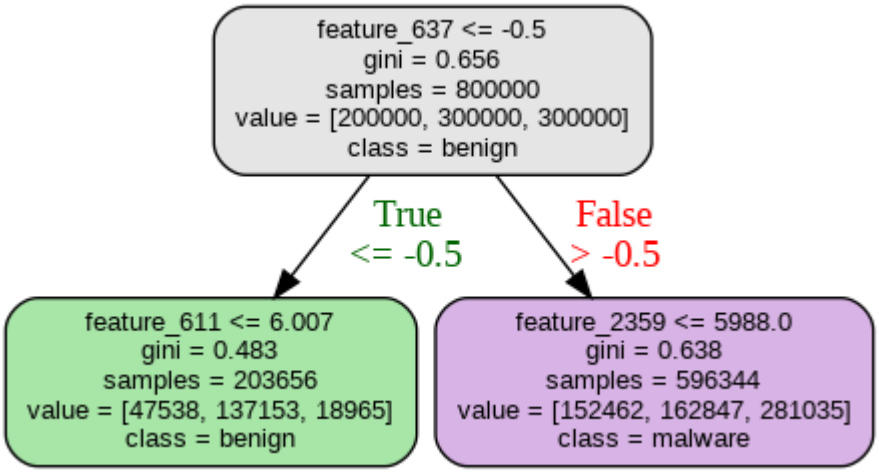
Foi gerada uma árvore de decisão, ilustrada nas Figuras 26, 27 e 28, a partir do modelo treinado, que explica a maneira como ele faz sua escolha de maneira gráfica e permitindo melhor compreensão. Cada nó da árvore de decisão representa uma “regra” ou “teste” que o modelo faz sobre os dados. Esses nós mostram várias informações importantes, cujas informações são explicadas na Tabela 13:

Tabela 13 - Informações dos campos da árvore de decisão

Campo	Significado	Exemplo na árvore
feature	Qual característica foi usada para dividir os dados, representa característica (ou atributo do dado). Cada número (feature_637) é uma coluna da sua matriz “X_train”. Então, “feature_637” é a 637ª coluna do vetor de características (dos milhares gerados pelo EMBER).	feature_637 <= -0.5
gini	Impureza do nó (mistura de classes), ou seja, o quão misturadas estão as classes dentro dele. Gini = 0 → nó puro (todas as amostras são da mesma classe). Gini alto (ex: 0.65) → o nó tem mistura de classes (*malware* e benignos juntos).	gini = 0.656
samples	É o número de amostras (linhas) do *dataset* que chegaram até esse nó durante o treinamento.	samples = 800000
value	Quantas amostras de cada classe estão no nó. class_names = ["unlabeled", "benign", "malware"]	[200000, 300000, 300000]
class	Classe predominante no nó (resultado). decisão final do nó	benign

Fonte: Próprios autores

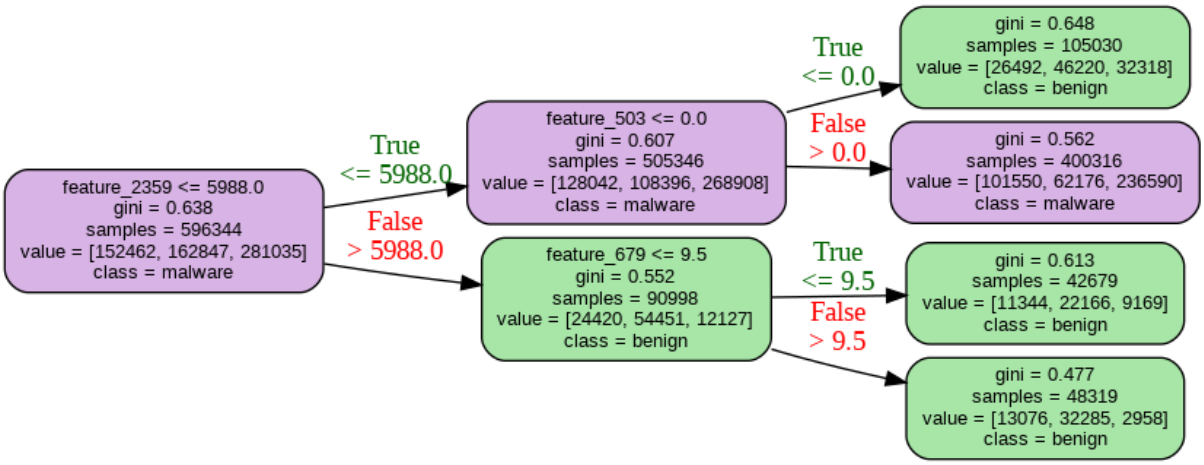
Figura 26 – Sub-árvore de decisão gerada contendo o primeiro nó e sua primeira decisão



Fonte: Próprios autores a partir do código presente no Jupyter Notebook

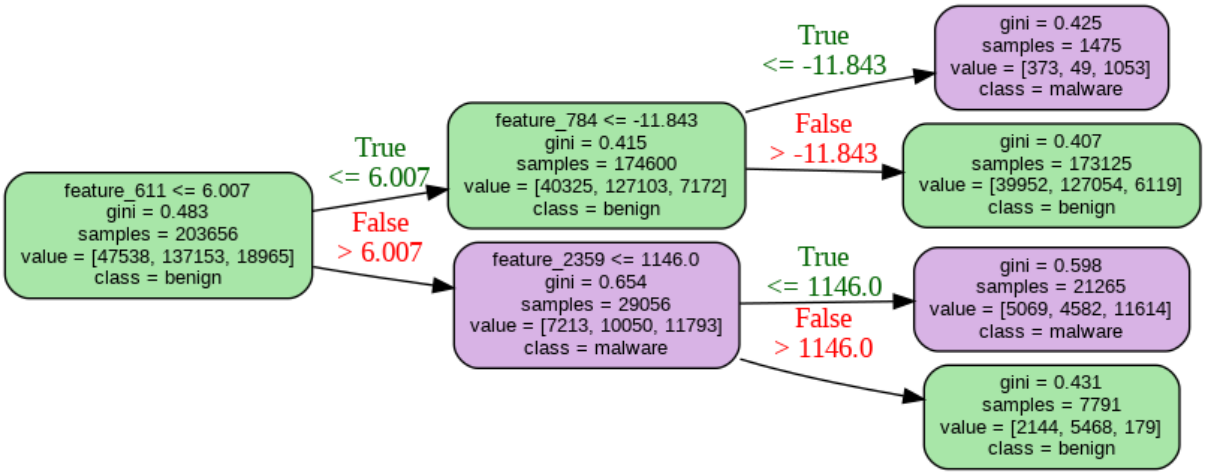
A Figura 26 contém o primeiro nó, de onde se inicia a decisão do modelo, e a primeira decisão que ele deve tomar. A seguir, será exibido o restante das decisões e seus nós em forma de sub-árvores, com a Figura 27 se iniciando a partir da primeira decisão considerando o arquivo malicioso e a Figura 28 o considerando benigno.

Figura 27 – Sub-árvore de decisão gerada considerando o arquivo malicioso



Fonte: Próprios autores a partir do código presente no Jupyter Notebook

Figura 28 – Sub-árvore de decisão gerada considerando o arquivo benigno



Fonte: Próprios autores a partir do código presente no Jupyter Notebook

Agora será finalmente iniciada a classificação de executáveis fazendo uso do modelo treinado para isso.

3.9 CLASSIFICANDO EXECUTÁVEIS

Para realizar a predição em um arquivo binário individual, foi carregado o modelo LightGBM previamente treinado pelo EMBER e aplicando função “predict_sample” que processou o conteúdo bruto do arquivo executável. Isso pode ser observado na Figura 29:

Figura 29 – Classificação do executável

```
import ember
import lightgbm as lgb
lgbm_model = lgb.Booster(model_file="C:\\Users\\0040972311011\\Documents\\ember2018\\ember_model_2018.txt")
putty_data = open("C:\\Users\\0040972311011\\Documents\\putty.exe", "rb").read()
print(ember.predict_sample(lgbm_model, putty_data))
```

```
[LightGBM] [Warning] Ignoring unrecognized parameter 'max_conflict_rate' found in model string.
[LightGBM] [Warning] Ignoring unrecognized parameter 'sparse_threshold' found in model string.
[LightGBM] [Warning] Ignoring unrecognized parameter 'enable_load_from_binary_file' found in model string.
[LightGBM] [Warning] Ignoring unrecognized parameter 'max_position' found in model string.
3.165875703315708e-05
```

Fonte: Próprios autores usando o Jupyter Notebook

O executável usado foi o “putty.exe”, um cliente SSH bem conhecido e claramente benigno. O resultado apresentado foi uma pontuação que representa a

probabilidade de o arquivo ser malicioso, onde valores próximos a zero indicam alta probabilidade de benignidade, e valores próximos a um indicam maior suspeita de malware. Agora será feita a mesma análise, porém em um *ransomware* real. No próximo subtópico será abordada a amostra de *malware* escolhida para o experimento.

3.10 AMOSTRA DE RANSOWARE

Para a amostra de *ransomware*, foi utilizado o WannaCry, muito conhecido pelos ataques realizados no passado fazendo uso dele. Após carregar-se o modelo em “ember_model_2018.txt” realizou-se a predição do malware. A saída representou um número muito próximo de um, indicando que ele foi detectado como altamente malicioso, eventos exibidos na Figura 30:

Figura 30 – Predição do Ransomware WannaCry

```
import ember
import lightgbm as lgb
lgbm_model = lgb.Booster(model_file="/root/Downloads/ember_model_2018.txt")
putty_data = open("/root/WannaCry.EXE", "rb").read()
print(ember.predict_sample(lgbm_model, putty_data))
```

```
0.9999947091796528
```

Fonte: Próprios autores

Agora, no tópico a seguir, será feita a análise de resultados obtidos.

4 ANÁLISE DE RESULTADOS

Nesta seção, foram expostos e analisados os resultados derivados da utilização do modelo EMBER na identificação de arquivos maliciosos. Examinando as métricas gerais de classificação, e elementos específicos, a finalidade foi analisar a capacidade do modelo em diversas situações.

Para análise de resultados, foi utilizado um código em Python, utilizando o Jupyter Notebook. Para iniciar a análise, foram carregados os módulos mostrados na Figura 31:

Figura 31 – Carregamento dos módulos para análise

```
import os
import ember
import numpy as np
import pandas as pd
import altair as alt
import lightgbm as lgb
import matplotlib.pylab as plt
from sklearn.metrics import roc_auc_score, roc_curve
_ = alt.renderers.enable('default')
```

Fonte: Próprios autores do Jupyter Notebook

Após se importar os módulos, foi definida uma variável para armazenar o diretório do *dataset*, conforme demonstrado na Figura 32:

Figura 32 - Variável do diretório do *dataset*

```
data_dir = "/home/user/Documentos/TCC/Metodologia-DataseEMBER/ember2018/"
```

Fonte: Próprios autores fazendo uso do Jupiter Notebook

Realizada essa etapa, foi feita a vetorização dos dados novamente, utilizando as funções do módulo EMBER. Esse processo extrai características como seções do binário, *imports, export tables*, entropia etc. Em seguida, os metadados (*hash*, *data*, *label* e *subset*) são gerados, passo ilustrado na Figura 33:

Figura 33 – Vetorização gerada novamente

```
ember.create_vectorized_features(data_dir)
_ = ember.create_metadata(data_dir)
```

```
Vectorizing training set
100%|██████████| 800000/800000 [20:16<00:00, 657.52it/s]
Vectorizing test set
100%|██████████| 200000/200000 [04:52<00:00, 683.81it/s]
```

Fonte: Próprios autores a partir do Jupyter Notebook

Nesse ponto, o *dataset* já vetorizado foi carregado na memória. “emberdf” recebeu o *dataframe* de metadados, enquanto “X_train”, “X_test”, “y_train” e “y_test” receberam os vetores numéricos e seus respectivos rótulos. O modelo LightGBM pré-treinado também foi carregado, permitindo gerar predições imediatamente. O carregamento das *features* é demonstrado na Figura 34:

Figura 34 – Carregando Features

```
emberdf = ember.read_metadata(data_dir)
X_train, y_train, X_test, y_test = ember.read_vectorized_features(data_dir)
lgbm_model = lgb.Booster(model_file=os.path.join(data_dir, "ember_model_2018.txt"))
```

Fonte: Próprios autores

A seguir será abordada a distribuição do Dataset EMBER além de algumas informações importante sobre ele.

4.1 DISTRIBUIÇÃO DO DATASET EMBER

A base de dados utilizada neste trabalho foi o EMBER 2018, contendo um grande volume de amostras destinadas tanto ao treinamento quanto à validação de modelos de aprendizado de máquina, o que torna possível avaliar o desempenho de detecção em um cenário próximo ao ambiente real.

O primeiro gráfico demonstra a divisão em *train* e *test* totalizando aproximadamente um milhão. Foi possível observar que o *subset* de treino contém a

maior parte dos dados. Cada *subset* foi dividido em três categorias: amostras benignas, maliciosas e não rotuladas, onde as não rotuladas refletem uma característica muito importante, pois aborda a realidade de muitos ambientes de segurança, onde nem todos os arquivos capturados possuem rótulo imediato. Para gerar o primeiro gráfico, na Figura 36, foi usado a célula de código presente na Figura 35:

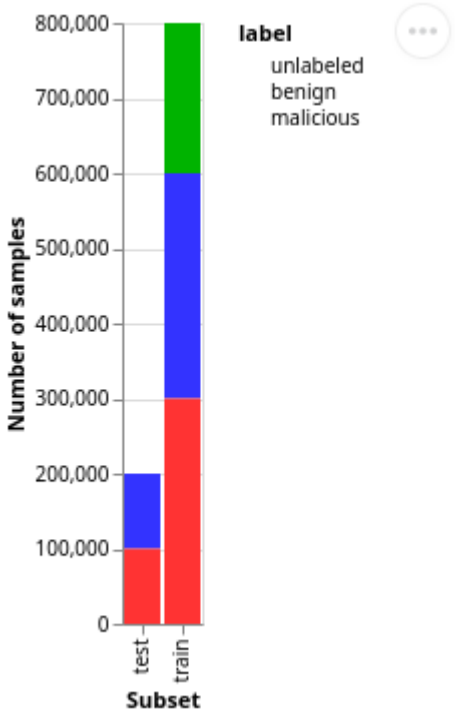
Figura 35 – Código da divisão de amostras

```
plotdf = emberdf.copy()
gbdf = plotdf.groupby(["label", "subset"]).count().reset_index()
alt.Chart(gbdf).mark_bar().encode(
    alt.X('subset:O', axis=alt.Axis(title='Subset')),
    alt.Y('sum(sha256):Q', axis=alt.Axis(title='Number of samples')),
    alt.Color('label:N', scale=alt.Scale(range=["#00b300", "#3333ff", "#ff3333"]),
    legend=alt.Legend(values=["unlabeled", "benign", "malicious"]))
)
```

Fonte: Próprios autores a partir do Jupyter Notebook

Ao rodar a célula de código, o gráfico foi gerado logo em sequência, assim, representou visualmente a divisão:

Figura 36 - Divisão de amostras



Fonte: Próprios autores usando a célula de código descrita no Jupyter Notebook

A existência de uma quantidade significativa de amostras “unlabeled” é especialmente importante, pois aproxima o processo de classificação de um cenário real, onde diversas ameaças emergentes ainda não foram totalmente analisadas.

O segundo gráfico apresenta a distribuição por mês das amostras, evidenciando o mês de aparecimento das amostras durante todo o ano de 2018, além de um conjunto adicional de amostras, anteriores a 2018. Esse gráfico foi gerado através da seguinte célula de comando, conforme a Figura 37:

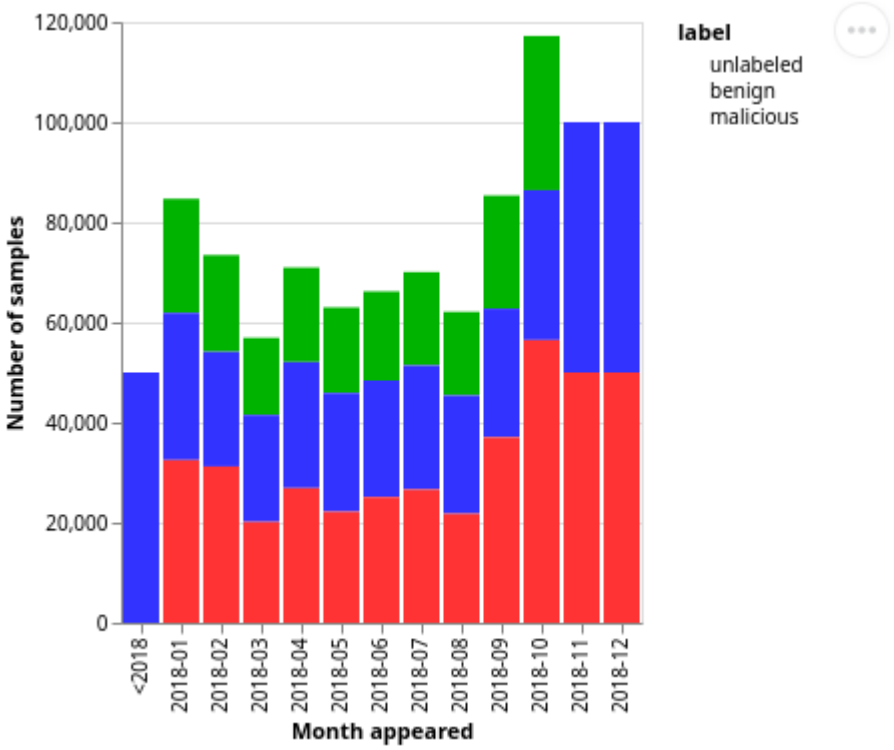
Figura 37 - Código do aparecimento de amostras

```
plotdf = emberdf.copy()
plotdf.loc[plotdf["appeared"] < "2018-01", "appeared"] = " <2018"
gbdf = plotdf.groupby(["appeared", "label"]).count().reset_index()
alt.Chart(gbdf).mark_bar().encode(
    alt.X('appeared:O', axis=alt.Axis(title='Month appeared')),
    alt.Y('sum(sha256):Q', axis=alt.Axis(title='Number of samples')),
    alt.Color('label:N', scale=alt.Scale(range=["#00b300", "#3333ff", "#ff3333"]),
    legend=alt.Legend(values=["unlabeled", "benign", "malicious"]))
)
```

Fonte: Próprios autores a partir do Jupyter Notebook

Feito isso, foi gerado a imagem a seguir que demonstra visualmente o aparecimento das amostras:

Figura 38 - Aparecimento de amostras



Fonte: Próprios autores com uso do Jupyter Notebook

O gráfico exibido na Figura 38 é muito importante, pois evidencia que o *dataset* não é estático, mas sim uma base que contempla uma evolução temporal, o que melhora a capacidade do modelo em reconhecer diferentes padrões de comportamento de *malware* ao longo do tempo. No próximo subtópico será discutido o desempenho com a taxa de falso positivo controlado.

4.2 DESEMPENHO COM FPR CONTROLADO (ENTRE 1% E 0.1)

Nesta etapa, foi feita uma avaliação da taxa de falso positivos. O objetivo dessa análise é observar como o modelo se comporta ao alcançar diferentes níveis de tolerância a falsos positivos.

O *threshold* define o ponto de corte da pontuação de probabilidade predita pelo modelo, determinando se uma amostra será classificada como maliciosa (próximo de um) ou benigna (próximo de zero). Um limiar mais baixo implica em uma detecção mais agressiva (maior taxa de detecção, porém mais falsos positivos), enquanto limiar mais alto torna o modelo mais conservador (menos falsos positivos, mas maior chance

de não detectar alguns malwares). Foi usado a seguinte célula de código, conforme exibido na Figura 39, que iniciou a fase de geração das previsões do modelo:

Figura 39 - Previsões

```
y_test_pred = lgbm_model.predict(X_test)
y_train_pred = lgbm_model.predict(X_train)
emberdf["y_pred"] = np.hstack((y_train_pred, y_test_pred))
```

Fonte: Próprios autores fazendo uso do Jupyter Notebook

Geradas as previsões, foi usado a seguinte célula de código, presente na Figura 40, para a avaliação do modelo:

Figura 40 - Código da avaliação do modelo

```
def get_fpr(y_true, y_pred):
    nbenign = (y_true == 0).sum()
    nfalse = (y_pred[y_true == 0] == 1).sum()
    return nfalse / float(nbenign)


def find_threshold(y_true, y_pred, fpr_target):
    thresh = 0.0
    fpr = get_fpr(y_true, y_pred > thresh)
    while fpr > fpr_target and thresh < 1.0:
        thresh += 0.0001
        fpr = get_fpr(y_true, y_pred > thresh)
    return thresh, fpr

testdf = emberdf[emberdf["subset"] == "test"]
print("ROC AUC:", roc_auc_score(testdf.label, testdf.y_pred))
print()

threshold, fpr = find_threshold(testdf.label, testdf.y_pred, 0.01)
fnr = (testdf.y_pred[testdf.label == 1] < threshold).sum() / float((testdf.label == 1).sum())
print("Ember Model Performance at 1% FPR:")
print("Threshold: {:.4f}".format(threshold))
print("False Positive Rate: {:.3f}%".format(fpr * 100))
print("False Negative Rate: {:.3f}%".format(fnr * 100))
print("Detection Rate: {}%".format(100 - fnr * 100))
print()

threshold, fpr = find_threshold(testdf.label, testdf.y_pred, 0.001)
fnr = (testdf.y_pred[testdf.label == 1] < threshold).sum() / float((testdf.label == 1).sum())
print("Ember Model Performance at 0.1% FPR:")
print("Threshold: {:.4f}".format(threshold))
print("False Positive Rate: {:.3f}%".format(fpr * 100))
print("False Negative Rate: {:.3f}%".format(fnr * 100))
print("Detection Rate: {}%".format(100 - fnr * 100))
```

Fonte: Próprios autores a partir do Jupyter Notebook

Logo após usar o código para avaliação do modelo, foi gerado em sequência a saída com informações para a avaliação, conforme exibido na Figura 41:

Figura 41 – Resultado da avaliação do modelo

```
ROC AUC: 0.9964289467999999

Ember Model Performance at 1% FPR:
Threshold: 0.8336
False Positive Rate: 1.000%
False Negative Rate: 3.502%
Detection Rate: 96.498%

Ember Model Performance at 0.1% FPR:
Threshold: 0.9996
False Positive Rate: 0.098%
False Negative Rate: 13.192%
Detection Rate: 86.80799999999999%
```

Fonte: Próprios autores com uso do Jupyter Notebook

Na saída com "ROC AUC: 0.9964" foi observado que o modelo tem a possibilidade de distinguir perfeitamente amostras malignas e benignas. Apenas um de cada cem arquivos benignos foi classificado erroneamente como *malware* com 1% de Falsos Positivos (FPR = 0.01), o modelo detectou cerca de 96,5% dos malwares, com 1% de Falsos Positivos. Já o modelo mais conservador com 0.1% (FPR = 0.001), apenas um de cada mil arquivos benignos é incorretamente alertado, porém, a taxa de detecção caiu para cerca de 87%.

Com base nos resultados obtidos, diminuir a FPR (ser mais rígido com falsos positivos) aumenta o limiar e reduz a taxa de detecção, sendo o comportamento esperado em qualquer modelo de classificação binária. Agora será abordada a classificação da amostra de *ransomware* realizada.

4.3 CLASSIFICAÇÃO DE AMOSTRA DE RANSOMWARE

Para avaliar a capacidade do modelo em identificar *ransomware*, foi utilizada uma amostra real do WannaCry, conhecida por criptografar arquivos do sistema e exigir pagamento para sua liberação, além de um executável legítimo para comparação de resultados, o Putty, um cliente SSH. O modelo LightGBM,

previamente treinado com o Dataset EMBER, foi carregado a partir do arquivo “ember_model_2018.txt”. Em seguida, a função “predict_sample” do módulo EMBER foi aplicada diretamente sobre o conteúdo binário do executável, processando suas características estáticas e retornando uma pontuação de probabilidade de ser *malware*.

A execução das duas classificações resultou em uma pontuação de 0.9999 na classificação do WannaCry, valor muito próximo de um, indicando alta probabilidade de comportamento malicioso. Durante a classificação do Putty, o resultado retornado foi um valor extremamente baixo, muito próximo de zero. Esse resultado demonstra que o modelo treinado foi capaz de identificar corretamente a amostra WannaCry como um *ransomware*, confirmando sua eficácia na detecção de ameaças conhecidas e perigosas. Agora será comentado sobre algumas limitações identificadas no *dataset* utilizado no trabalho.

4.4 LIMITAÇÕES OBSERVADAS

Durante os testes complementares, foi avaliada uma amostra recente de *ransomware*, não presente no conjunto de dados original EMBER 2018. Ao realizar a predição com o mesmo modelo LightGBM, observou-se que o valor retornado foi muito próximo de zero, o que indica alta probabilidade de benignidade.

Esse comportamento demonstra uma limitação importante do modelo em detectar variantes novas ou amostras de malware que não compartilham características estáticas similares com aquelas utilizadas durante o treinamento.

Como o Dataset EMBER 2018 foi construído com amostras coletadas até o ano de 2018, o modelo tende a apresentar redução de desempenho frente a ameaças mais recentes, especialmente quando o *ransomware* adota técnicas modernas de ofuscação, empacotamento, ou assinaturas de código alteradas. A seguir serão abordadas as considerações finais sobre os resultados obtidos.

5 RESULTADOS E CONSIDERAÇÕES FINAIS

O presente trabalho de conclusão de curso teve como objetivo principal aplicar técnicas de aprendizado de máquina na classificação de executáveis maliciosos utilizando o Dataset EMBER de 2018. Com ele, os testes feitos demostraram que o

modelo treinado teve um ótimo desempenho, alcançando um ROC AUC de 0.9964, o que mostrou alta capacidade de separação entre amostras benignas e maliciosas.

Observou-se que nos testes com controle de taxa de falsos positivos (FPR), o modelo manteve taxas de detecção elevadas mesmo em cenários mais restritivos, a 1% de FPR, o modelo alcançou 96,5% de taxa de detecção. A 0,1% de FPR, ainda obteve 86,8% de taxa de detecção. Os dados apontam que o classificador é eficiente e resistente na detecção de ameaças conhecidas, incluindo amostras de *ransomware* clássico, como o WannaCry, que foi classificado corretamente com uma probabilidade quase igual a um, confirmando que se tratava de um malware.

Entretanto, em testes com amostras recentes, o modelo teve limitações em realizar a classificação do executável como *malwares* de maneira adequada e, nesses casos, foram obtidas pontuações muito próximas de zero, indicando falsos negativos e refletindo a dependência temporal e estática do modelo. Dessa forma, conclui-se que, embora o modelo tenha apresentado excelente desempenho na detecção de malwares conhecidos, sua eficácia pode ser comprometida frente a ameaças modernas ou variações inéditas.

Com isso em mente, fica evidente a necessidade de mais estudos na área além da criação de novos métodos e tecnologias para tornar a abordagem mais confiável e eficaz na classificação de *malwares* mais sofisticados.

Em suma, este trabalho respondeu a questão de pesquisa "como aplicar técnicas de aprendizado de máquina para identificar e classificar executáveis maliciosos de forma eficaz, utilizando o Dataset EMBER como base de treinamento e teste?" e, mesmo com as limitações encontradas, o objetivo geral “aplicar técnicas de aprendizado de máquina na classificação de executáveis maliciosos utilizando o Dataset EMBER” pôde ser alcançado.

REFERÊNCIAS

ABELIUK, A.; GUTIÉRREZ, C. Historia y evoluación de la inteligencia artificial. **Revista Bits de Ciencia**, n. 21, p. 14-21, 2021.

BASTOS, Athena. **Quais são os pilares e as funções da segurança da informação?** Alura Para Empresas, [s. d.]. Disponível em: https://www.alura.com.br/empresas/artigos/seguranca-da-informacao. Acesso em: 28 out. 2025.

BISHOP, C. M. **Pattern recognition and machine learning**. [S.l.]: Springer, 2006.

Cloudflare. **O que foi o ataque de ransomware WannaCry?**. 2025. Disponível em: https://www.cloudflare.com/pt-br/learning/security/ransomware/wannacry-ransomware/. Acesso em: 24 out. 2025

CREVIER, D. **AI:** The tumultuous history of the search for artificial intelligence. [S.l.]: Basic Books, 1993.

DEMETRIO, Rodrigo. **Estatísticas de IA:** 500+ fatos que impulsionam a inovação global. Bureal Works, [S.l], [s.d]. https://www.bureauworks.com/fr/blog/ai-estatisticas-500-fatos-impulsionando-a-inovacao-global. Acesso em: 02 set. 2025

DICIONÁRIO PRIBERAM. **Computar.** 2025. Disponível em: https://dicionario.priberam.org/computar. Acesso em: 30 ago. 2025.

DONDA, Daniel. **Estrutura de arquivos executáveis (Formato PE).** Daniel Donda, [S.l.], [s.d.]. Disponível em: https://danieldonda.com/estrutura-de-arquivos-executaveis-formato-pe/. Acesso em: 15 out. 2025.

DONDA, Daniel. **Hunting com notebook do Jupyter.** [s.d]. Disponível em: https://danieldonda.com/. Acesso em 10 out. 2025.

DYSON, G. **Turing's cathedral:** the origins of the digital universe. [S.l.]: Vintage, 2012.

ESET. **História e evolução do malware desde 1986.** OverBR, [S.l.], [s.d.]. Disponível em: https://overbr.com.br/artigos/historia-e-evolucao-do-malware-desde-1986. Acesso em: 1 out. 2025.

FERLAINO, Pierpaolo. **El Ajadrecista:** the mechanical chess player by Leonardo Torres Y Quevedo, 2022. Disponível em: https://pierpaoloferlaino.medium.com/el-ajadrecista-the-mechanical-chess-player-by-leonardo-torres-y-quevedo-ed8de8a3c06e. Acesso em: 20 out. 2025.

FOUNDERS FORUM GROUP. **AI statistics 2024–2025:** global trends, market growth & adoption data. Londres: Founders Forum Group, 23 jun. 2025. Disponível em: https://ff.co/ai-statistics-trends-global-market/. Acesso em: 01 set. 2025.

FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. **The elements of statistical learning:** Data Mining, Inference, and Prediction. 2. ed. [S.l]: Springer, 2009.

GÉRON, A. **Hands-on machine learning with Scikit-learn, Keras & TensorFlow**. 3. ed. [S.l.]: O'Reilly Media, 2019.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning:** Adaptive Computation and Machine Learning. [S.l.]: The MIT Press, 2016.

KELLY, M. C. et al. **Computer:** a history of the information machine. 3. ed. [S.l.]: Westview Press, 2013.

KOSINSKI, Matthew. **O que é ransomware?** IBM Think, [S.l.]: IBM Corporation, [2023]. Disponível em: https://www.ibm.com/br-pt/think/topics/ransomware. Acesso em: 1 out. 2025.

LECUN, Y.; BENGIO, Y.; HINTON, G. **Deep learning**. [S.l.]: HAL Open Science, 2023.

MICROSOFT. **O que é malware? Definição e tipos.** Segurança da Microsoft, [2025]. Disponível em: https://www.microsoft.com/pt-br/security/business/security-101/what-is-malware. Acesso em: 30 set. 2025.

MITCHELL, M. **Artificial intelligence:** A Guide for Thinking Humans. [S.l]: Farrar, Straus and Giroux, 2019.

MUCCI, T. **A história da IA.** IBM, 2024. Disponível em: https://www.ibm.com/br-pt/think/topics/history-of-artificial-intelligence. Acesso em: 26 set. 2025.

RAMOS, Marien. **Uso de inteligência artificial aumenta e alcança 72% das empresas, diz pesquisa**. CNN Brasil, 08 jun. 2024. Disponível em: https://www.cnnbrasil.com.br/economia/negocios/uso-de-inteligencia-artificial-aumenta-e-alcanca-72-das-empresas-diz-pesquisa/. Acesso em: 02 set. 2025.

ROHR, Altieres. **Primeiro vírus de PCs, 'Brain' completa 25 anos.** G1, 2025. Disponível em: https://g1.globo.com/tecnologia/noticia/2011/01/primeiro-virus-de-pcs-brain-completa-25-anos.html. Acesso em: 12 set. 2025.

ROTH, Phil. **Elastic Malware Benchmark For Empowering Researchers**. Github, 2022. Disponível em: https://github.com/elastic/ember. Acesso em: 30 out. 2025.

ROTH, Phil. **Introducing Ember: An Open Source Classifier And Dataset**. elastic, 2018. Disponível em: https://www.elastic.co/blog/introducing-ember-open-source-classifier-and-dataset. Acesso em: 30 out. 2025.

RUSSEL, S.; NORVIG, P. **Artificial intelligence:** A Modern Approach. 3. ed. [S.l.]: Pearson, 2009.

SANTOS, J. **A incrível saga da inteligência artificial na história da humanidade em 2025.** Disponível em: https://pmp.com.br/sociais/inteligencia-artificial/. Acesso em: 25 ago. 2025.

SOUSA, Priscila. **Segurança da informação - O que é, importância, conceito e definição**. Conceito.de, 2024. Disponível em: https://conceito.de/seguranca-da-informacao. Acesso em: 27 out. 2025.

SCHMIDHUBER, J. **Annotated history of modern AI and deep learning.** [S.l.]: IDSIA, 2022.

SHETTERLY, L. S. **Hidden figures:** the american dream and the untold story of the black women mathematicians who helped win the space race. [S.l.]: William Morrow Paperbacks, 2016.

SULEYMAN, M.; BHASKAR, M. **A próxima onda:** inteligência artificial, poder e o maior dilema do século XXI. [S.l.]: Record, 2023.

TECNOLAN. **Computadores humanos:** as mulheres da NASA, 2025. Disponível em: https://www.tecnolan.com.br/2019/04/26/computadores-humanos-as-mulheres-da-nasa/. Acesso em: 24 out. 2025.

TELEFÔNICA BRASIL S.A. **Entenda o que é malware, os tipos e como se proteger.** São Paulo: Vivo, 2024. Disponível em: https://vivo.com.br/para-voce/por-que-vivo/vivo-explica/para-descomplicar/malware. Acesso em: 7 out. 2025.

THUNDERBIT. **140 estatísticas essenciais de inteligência artificial para 2025**. São Paulo: Thunderbit, 23 maio 2025. Disponível em: https://thunderbit.com/pt/blog/top-artificial-intelligence-stats. Acesso em: 02 set. 2025.

WOILER, S. **Computador: conceitos e aplicações.** Revista de Administração de Empresas, p. 2-5, 1970. Disponível em: https://www.scielo.br/j/rae/a/8FX5tDznG349WPxMJM6Kz7s/. Acesso em: 30 ago. 2025.