CENTRO UNIVERSITÁRIO CARIOCA - UNICARIOCA

VITOR DE ÁVILA FALCÃO

VALIDAÇÃO DE ALGORITMOS DE DETECÇÃO DE ATAQUES EM REDES DE COMPUTADORES COM UTILIZAÇÃO DE INTELIGÊNCIA ARTIFICIAL

RIO DE JANEIRO

2022

VITOR DE ÁVILA FALCÃO

VALIDAÇÃO DE ALGORITMOS DE DETECÇÃO DE ATAQUES EM REDES DE COMPUTADORES COM UTILIZAÇÃO DE INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso apresentado ao Centro Universitário Carioca, como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Fabio Henrique Silva

RIO DE JANEIRO

2022

F178v Falcão, Vitor de Ávila
Validação de algoritmos de detecção de ataques em redes de computadores com utilização de inteligência artificial / Vitor de Ávila Falcão. – Rio de Janeiro, 2022.
104 f.

Orientador: Fábio Henrique Silva
Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Centro Universitário UniCarioca, Rio de Janeiro, 2022.

1.Detecção de ataques. 2. Inteligência artificial. 3. Classificação multi-Label. 4. Python. 5. Sklearn, 6. Skmultilearn. I. Silva, Fábio Henrique, prof. orient. II. Título.

CDD 004.6

VITOR DE ÁVILA FALCÃO

VALIDAÇÃO DE ALGORITMOS DE DETECÇÃO DE ATAQUES EM REDES DE COMPUTADORES COM UTILIZAÇÃO DE INTELIGÊNCIA ARTIFICIAL

Trabalho de conclusão de curso apresentado ao Centro Universitário Carioca, como requisito do grau de Bacharel em Ciência da Computação.

Rio de Janeiro, 14 de junho de 2022.

Banca Examinadora

_____

Prof. Fábio Henrique Silva, MSc. - Orientador
Centro Universitário Carioca

_____

Prof. André Luiz Avelino Sobral, M.Sc. - Coordenador
Centro Universitário Carioca

_____

Prof. Ricardo Pires Mesquita, D.Sc. - Convidado
Centro Universitário Carioca Centro Universitário Carioca

“Incluir é viver acreditando que
como humanos,
Podemos sempre seguir adiante:
Se nossa realidade imediata nos
limita,
Boas doses de sonho alimentam
um outro dia.”
João Beauclair

AGRADECIMENTOS

Aos professores da Unicarioca pela dedicação e ensinamentos transmitidos especialmente em um período difícil de Pandemia de COVID-19 em que todos tiveram que se desdobrar para produzir aulas on-line com novas tecnologias e métodos em tempo muito curto para que os alunos não fossem prejudicados.

RESUMO

O presente TCC visa estudar e implementar as técnicas de Inteligência Artificial, em especial de Detecção de Ataques e de Classificação *Multi-Label*, que estão sendo utilizadas pela comunidade científica e indústria como parte da solução do problema de ataques de segurança de informação às redes de computadores modernas e no desenvolvimento de Sistemas de Detecção de Intrusão. O trabalho utiliza dados gravados de redes reais abertos e disponíveis na Internet, e inclui uma descrição das características desses dados e do processo utilizado para sua geração. É implementado um novo algoritmo para seleção de características (*Features*) gravadas da rede de computadores de forma a melhorar o tempo de execução dessa seleção com baixa perda de qualidade. É implementado um algoritmo para redução controlada do tamanho do arquivo original de gravação para evitar tempos de processamento excessivos. É implementado um algoritmo para preparar o arquivo reduzido de gravação para processamento Multi-Label. São implementados algoritmos para classificação Multi-Label com indicação do tipo de ataque realizado, utilizando técnicas de Transformação do Problema e Adaptadas. O trabalho utiliza a linguagem Python e suas bibliotecas SkLearn e SkMultiLearn. Os resultados obtidos são avaliados através de métricas de desempenho.

Palavras-chaves: **Detecção de Ataques, Inteligência Artificial, Classificação *Multi-Label, Python, SkLearn, SkMultiLearn.***

ABSTRACT

This TCC aims to study and implement Artificial Intelligence techniques, in particular Attacks Detection and Multi-Label Classification, which are being used by the scientific community and industry as part of the solution to the problem of information security attacks on modern computer networks and in the development of Intrusion Detection Systems. The paper uses data recorded from real networks that are open and available on the Internet, and includes a description of the characteristics of this data and the process used to generate it. A new algorithm for selection of recorded computer network characteristics (Features) is implemented in order to improve the execution time of this selection with low loss of quality. An algorithm is implemented for controlled reduction of the original recording file size to avoid excessive processing times. An algorithm is implemented for preparing the reduced file for Multi-Label processing. Algorithms are implemented for Multi-Label classification with indication of the type of attack performed, using Problem Transformation and Adapted techniques. The work uses the Python language and its libraries SkLearn and SkMultiLearn. The results obtained are evaluated using performance metrics.

Keywords: ***Attacks Detection, Artificial Intelligence, Multi-Label Classification, Python, SkLearn, SkMultiLearn.***

.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SÍMBOLOS

5G	Quinta geração de redes celulares de dados móveis
BR-KNN	*Binary Relevance K Nearest Neighbors*
CIC-IDS2017	*Canadian Institute for Cybersecurity – Intrusion Detection Evaluation Dataset 2017*
DARPA	*Defense Advanced Research Projects Agency*
DDoS	*Distributed Denial Of Service*
DoS	*Denial of Services*
HTTP	*HyperText Transfer Protocol*
ID3	Tipo de árrvore de decisão que pode ser utilizada para regressão ou classificação
IDS	*Intrusion Detection Sysrem*
IP	*Internet Protocol*
LGPD	Lei Geral de Proteção de Dados
MIT	*Massachussetz Institute of Technology*
ML-KNN	*Multi-Label K Nearest Neighbors*
R2L	*Remote To Local Attack*
SQL	*Standard Query Language*
TCP	*Transmission Control Protocol*
U2R	*User To Root Attack*

SUMÁRIO

1 INTRODUÇÃO

As redes de computadores de forma geral, como por exemplo a *Internet*, possuem vulnerabilidades que tem sido exploradas há anos para realização de ataques de segurança de informação causando prejuízo a economia e a segurança mundial, sejam às empresas, aos governos e a todos os seus usuários, pela negação do uso da rede ou pelo acesso, roubo ou sequestro de dados privados.

Esses ataques se tornaram bastante frequentes nos dias de hoje e motivo de preocupação por partes dos governos, que emitiram leis que responsabilizam as empresas pela proteção dos dados dos seus cidadãos, como por exemplo a LGPD no Brasil (Lei Geral de Proteção de Dados).

O aumento da escala de utilizadores traz em si maiores vulnerabilidades e um maior risco de novos ataques coordenados a partir de uma quantidade enorme de máquinas conectadas.

A gestão da segurança da informação nesse novo ambiente complexo e de larga escala passa pela automação através da utilização de novas tecnologias como por exemplo a de Inteligência Artificial[1]. A detecção automática de ataques e a utilização de Sistemas de Detecção de Intrusão, com o uso das técnicas de Inteligência Artificial, são medidas de proteção a serem utilizadas nesse cenário.

1.1 Objetivo Geral

O objetivo geral desse trabalho é obter a validação de algoritmos de classificação utilizados na detecção de ataques em redes de computadores através da comparação de várias técnicas encontradas na literatura especializada, e a proposição de funcionalidades que possibilitem incrementar os processos de detecção.

1.1.1. Objetivos específicos

Para alcançar esse objetivo geral, serão estudadas e colocadas em prática algumas técnicas de Inteligência Artificial Supervisionada de Classificação *Multi-Label* através do processamento de dados gravados de dados de ataques reais realizados em rede controlada na Universidade de New Brunswick no Canadá e disponíveis para avaliação na *Internet* em [2].

O trabalho procura atingir as seguintes metas:

- Estudar as técnicas de Inteligência Artificial de Aprendizagem Supervisionada para realização de classificação *Multi-Label* de forma a possibilitar a indicação do ataque realizado ou normalidade a partir do processamento dos dados de rede gravados;
- Implementar um novo algorítmo para seleção de características (*Features*) gravadas da rede de computadores de forma a melhorar o tempo de execução desse processo de seleção com baixa perda de qualidade;
- Implementar um algoritmo para redução controlada do tamanho do arquivo original de gravação para evitar tempos de processamento excessivos.
- Implementar um algoritmo para preparar o arquivo reduzido de gravação para processamento *Multi-Label*;
- Implementar algoritmos para classificação *Multi-Label* com indicação do tipo de ataque realizado, utilizando técnicas de Transformação do Problema: Relevância Binária, Cadeia de Classificadores e *Label PowerSets*; e técnicas de Adaptação: ML-KNN (*Multi-Label* K Vizinhos Mais Próximos) e BR-KNN (Relevância Binária para K Vizinhos Mais Próximos); e
- Analisar e comparar os resultados obtidos na implementação dos algoritmos citados anteriormente.

1.2 Motivação

O trabalho de mestrado desenvolvido em [5] aborda a detecção de anomalias em redes de computadores a partir do processamento por Inteligência Artificial dos dados gravados no CIC-IDS2017 [2], visando a sua classificação binária (existência de ataque ou não), mas não chega a utilizar classificação Multi-Label para indicar o tipo de ataque. Os resultados obtidos para classificação binária foram muito bons, mas o processamento de uma massa de dados de 1 G *Bytes* requer um tempo muito alto de execução, conforme pode-se verificar ao baixar e executar os códigos desse trabalho disponibilizados em [6].

O resultado bem-sucedido do trabalho mencionado no parágrafo anterior e a possibilidade de reuso do conjunto de dados gravados de redes reais já limpos e filtrados [2][6] foram os motivadores para o desenvolvimento desse trabalho, no que tange a extensão do trabalho original de modo a apresentar o processamento dos dados, com intuito de obter uma classificação *Multi-Label*, que indique o tipo de ataque realizado ou normalidade. Para tal, houve uma preocupação em reduzir o seu tempo de processamento pela redução controlada do tamanho do arquivo de gravação de dados.

1.3. Justificativa

O trabalho em questão tem relevante importância ao considerar as vulnerabilidades nos cenários de segurança de informação em redes de grande escala e a possibilidade de utilização de tecnologias de Inteligência Artificial para automatizar a detecção de ataques como medida de proteção.

A realização do TCC é uma oportunidade de aplicar conhecimentos de programação em Python, Inteligência Artificial e de Segurança de Informação adquiridos ao longo do Curso de Ciência da Computação.

1.4 Materiais e Métodos

Os métodos utilizados no trabalho se baseiam em pesquisa bibliográfica e em pesquisa experimental.

A pesquisa bibliográfica consiste do levantamento e estudo da bibliografia utilizando o acervo disponível na Internet, constituído de artigos, teses, livros, repositórios e bibliotecas de software e dados coletados em experimentos realizados por terceiros, com foco no tema escolhido.

A pesquisa experimental é realizada através da implementação de algoritmos de classificação *Multi-label* na linguagem Python, realização de testes de execução em ambiente *PyCharm* no Windows-10 e análise de resultados utilizando métricas de desempenho e gráficos. Na análise de resultados é feita a comparação entre as alternativas utilizadas para os algoritmos e seus parâmetros de ajuste.

1.5 Organização do Trabalho

No Capítulo 2 é apresentada a Fundamentaçãção Básica para realização do TCC, incluindo: a taxonomia dos tipos de ataques, a descrição dos dados obtidos com a gravação, o processo utilizado pela Universidade de New Brunswick no Canadá para geração do conjunto de dados gravados (*Dataset* CIC-IDS2017) em redes reais, o ambiente de teste utilizado (*Test Bed*), a descrição do processo de detecção de ataques com classificação *Multi-Label*, os tipos de ataques realizados, e as ferramentas empregadas para geração dos ataques e para sua gravação.

No Capítulo 3 é apresentada a Fundamentação Específica para a Solução do Problema tratado no presente TCC, incluindo: Técnicas de Seleção de *Features*, Àrvores de Decisão, Florestas Aleatórias (*Random Forests*), Classificação Binária, K Vizinhos mais Próximos, Técnicas de Classificação *Multi-Label*, Métricas de Desempenho para os Classificadores *Multi-Label* de Ataques, Técnicas de Transformação do Problema para Classificação *Multi-Label* eTécnicas de Adaptação para Classificação *Multi-Label*.

No Capítulo 4 são abordados os testes realizados e os resultados obtidos na execução do presente TCC, incluindo: Implementação de Novo Algorirmo de Seleção de Features por Florestas Aleatórias (*Random Forests*), Redução Controlada do

Arquivo de Gravação de Dados de Ataques, Transformação dos arquivos para processamento por classificador *Multi-Label*, Implementação em Python do Classificador *Multi-label* por Transformação do Problema, e Implementação em Python do Classificador *Multi-label* por Adaptação.

No capítulo 5 é registrada a conclusão do trabalho e a indicação para realização de trabalhos futuros.

.

2 FUNDAMENTAÇÃO BÁSICA

Para melhor organização e compreensão dos assuntos que servem de base para esse trabalho, o presente capítulo será dividido em duas partes: (2.1) O processo de geração dos dados para a detecção dos tipos de ataques à Segurança da Informação em ambiente de redes de computadores, e (2.2) as técnicas utilizadas para classificar os dados em busca de padrões anômalos que possam caracterizar ataques à Segurança da Informação.

2.1. O PROCESSO DE GERAÇÃO DOS DADOS PARA A DETECÇÃO DOS TIPOS DE ATAQUES À SEGURANÇA DA INFORMAÇÃO EM AMBIENTE DE REDES DE COMPUTADORES

Neste tópico, serão apresentadas as seguintes fundamentações:

- A taxonomia dos tipos de ataques;
- A descrição dos dados obtidos com a gravação;
- O processo utilizado pela Universidade de New Brunswick no Canadá para geração do conjunto de dados gravados (*Dataset* CIC-IDS2017) em redes reais;
- O ambiente de teste utilizado (*Test Bed*);
- A descrição do processo de detecção de ataques com classificação Multi-Label; e
- Os tipos de ataques realizados, as ferramentas empregadas para geração dos ataques e para sua gravação.

2.1.1 Taxonomia dos Tipos de Ataques

Um ataque ou intrusão é uma sequencia de operações que colocam a segurança da rede ou do sistema computacional em risco. Os ataques às redes de computadores podem ser classificados de passivos e ativos. Os ataques passivos possuem um efeito indireto e são lançados com os seguintes objetivos: monitorar e

gravar o tráfego na rede e coletar informação útil para lançamento de um ataque ativo a posteriori [9].

Os ataques ativos são classificados em 4 categorias de acordo com a Agência de Pesquisa de Defesa dos EUA (DARPA) [9]:

- Negação de Serviço (*DoS – Denial of Service*) – esse ataque procura bloquear o acesso a certos recursos ou tornar algum serviço da rede indisponível como por exemplo o e-mail ou uma conexão. Uma variante desse ataque é a Negação de Serviço Distribuída (*DDoS*) que é realizada de forma coordenada a partir de várias máquinas ao invés de uma única;
- Ataques de Usuário para Raiz (U2R – *User To Root Attack*) – esse ataque busca primeiro ter acesso a senha da conta do usuário seja através de engenharia social ou de varredura de força bruta. Uma vez tendo acessado a conta do usuário ele procura vulnerabilidades do sistema para ter acesso a conta raiz (*root*) ou de administrador com maiores privilégios para utilizar serviços e acessar recursos do sistema;
- Ataque de Remoto Para Local (R2L – *Remote To Local Attack*) – esse ataque é semelhante ao U2R no sentido de descobrir a senha da vítima, porém é realizado de forma remota através de uma solicitação de login (*Remote Login* ou *Secure Shell*) ou uma solicitação de conexão através da rede; e
- Ataque de Examinar e Explorar (*Probe*) – esse ataque faz a varredura da rede em vários modos para obter informações com endereços de IP válidos, serviços oferecidos, sistemas operacionais utilizados, entre outros. O atacante usa essa informação para identificar vulnerabilidades do sistema e lançar ataques contra suas máquinas e serviços.

2.1.2 Descrição das características dos dados gravados e da Ferramenta Utilizada para Gravação

A lista completa das características (*Features*) dos dados gravados no *Dataset CIC-IDS2017* com a descrição de cada uma pode ser encontrada Apêndice A da referência [5], num total de 85 características gravadas. Essas características incluem:

Endereços IP e de portas fonte e destino, indicação da direção do fluxo de dados (entrada ou saída da rede vítima), Protocolo, estatísticas dos pacotes e segmentos transmitidos (máximo, mínimo, média e desvio padrão de tamanho, totais de pacotes transmitidos), taxas de transmissão (em Pacotes/seg e em *Bytes/seg*), estatísticas do fluxo temporal(máximo, mínimo, média e desvio padrão do intervalo de tempo entre chegada de pacotes), Comprimento dos cabeçalhos, *Flags* (PSH, URG, FIN, SYN, RST, ACK, CWE e ECE), o rótulo de ataque ou normalidade(*Label*), *Time Stamp*, entre outros.

A ferramenta que foi utilizada para realização das gravações foi o *CICFlowMeter* do Instituto Canadense de Cybersegurança da Universidade New Brunswick [21].

2.1.3 Processo Utilizado para Geração do Conjunto de Dados Gravados

O processo para geração do conjunto de dados gravados (*Dataset* CIC-IDS2017) está descrito em [3], e usa os seguintes passos:

- Estabelecimento de um ambiente de teste controlado constituído de duas redes interligadas, uma rede para realização dos ataques (rede atacante) e outra rede para sofrer os ataques (rede vítima);
- Seleção de ferramentas de *software* para realização dos ataques a partir da rede atacante;
- Seleção de uma ferramenta de *software* para gravação e análise estatística de dados do tráfego de pacotes no roteador/gateway de entrada/saída da rede vítima;
- Realização de diferentes ataques ao longo de uma semana de segunda a sexta-feira acionando a ferramenta de gravação de forma a gerar diferentes arquivos do tipo separado por vírgula (.csv), um para cada dia da semana;
- Certificação de que cada linha dos dados gravados contenha o rótulo (*Label*) de tipo de ataque ou de normalidade (Benigno). Isso é fácil de se conseguir pois tem-se o controle do período de acionamento das ferramentas de ataque na rede atacante. A gravação dos rótulos se faz

necessária para o processamento de Inteligência Artificial do tipo Supervisionado. Nesse tipo de processamento é realizado um treinamento do classificador que utiliza os rótulos gravados previamente para sua otimização.

Obs: No experimento não foram realizados ataques de forma concorrente, isto é, mais de um ataque em um mesmo intervalo de tempo de gravação. Dessa forma cada rótulo (*Label*) gravado em cada linha vai indicar apenas um tipo de ataque ou normalidade (rótulo de Benigno). Tal característica indica que temos um problema de classificação *Multi-Class* que pode ser tratado por algoritimos de cunho mais geral voltados para classificação *Multi-Label*.

2.1.4 Ambiente de Teste Controlado para Gravação

A Figura 1 mostra o ambiente de teste controlado (*Test Bed*) utilizado para gravação dos ataques no *DataSet CIC-IDS2017* com as suas duas redes interligadas (rede atacante e rede vítima). As máquinas utilizadas nas duas redes utilizaram diversos sistemas operacionais: das linhas Windows, Mac e Linux [3].

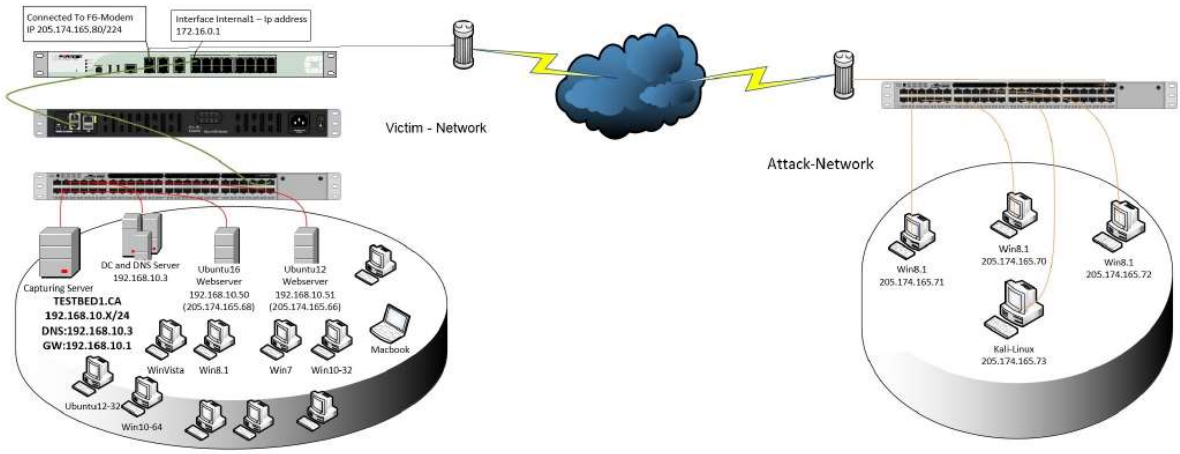


Figura 1: Ambiente de Teste (Retirado de: [3])

Obs: Os ataques realizados em um ambiente de teste controlado apresentam pouca variedade de localização em termos dos endereços IP utilizados. Nesse cenário a utilização dos IPs, apesar de disponíveis na gravação tem pouca significância para

a classificação. Sendo assim, as características (*Features*) de endereçamento poderão ser filtradas antes do processamento.

2.1.5 Descrição do Processo de Detecção de Ataques com Classificação Multi-Label

Esta seção descreve o processo que será utilizado para a Detecção de Ataques com classificação *Multi-Label* em Redes de Computadores a partir do conjunto de dados de ataques gravados (*Dataset*) descrito no item 2.5, incluindo as seguintes etapas (ver Figura 2):

Etapa 1) Pré-processamento;
Etapa 2) Seleção de Características (*Features*);
Etapa 3) Redução Controlada do Arquivo;
Etapa 4) Transformação do Arquivo para Multi-Label;
Etapa 5) Processamento de Inteligência Artificial; e
Etapa 6) Avaliação dos Resultados.

O processo utilizado no desenvolvimento da solução do presente trabalho está baseado na tese de mestrado apresentada em [5]. Em consonância com o objetivo geral do trabalho descrito no item 1.1 são propostas novas funcionalidades que possibilitem incrementar os processos de detecção, especialmente no que ser refere a classificação *Multi-Label*. Através da solução implementada de Classificadores *Multi-Label* é possível indicar o tipo de ataque realizado, além da detecção do ataque.

O trabalho inclui as seguntes melhorias: algoritmo alternativo para seleção de Features de forma mais rápida com baixa perda de qualidade (decorrente do estudo da etapa 2 da Figura 2), Redução controlada do tamanho do arquivo *Dataset* para possibilitar menores tempos de processamento (etapa 3 da figura 2), Tranformação da arquivo reduzido em preparação para o processamento *Multi-Label*(etapa 4 da Figura 2), Estudo e Implementação de algoritmos de classificação *Multi-Label* utilizando as técnicas de Tranformação do Problema e de Adaptação(etapa 5 da Figura 2), e utilizacão de métricas de classificação *Multi-Label* para avaliação de desempenho(etapa 6 da Figura 2).

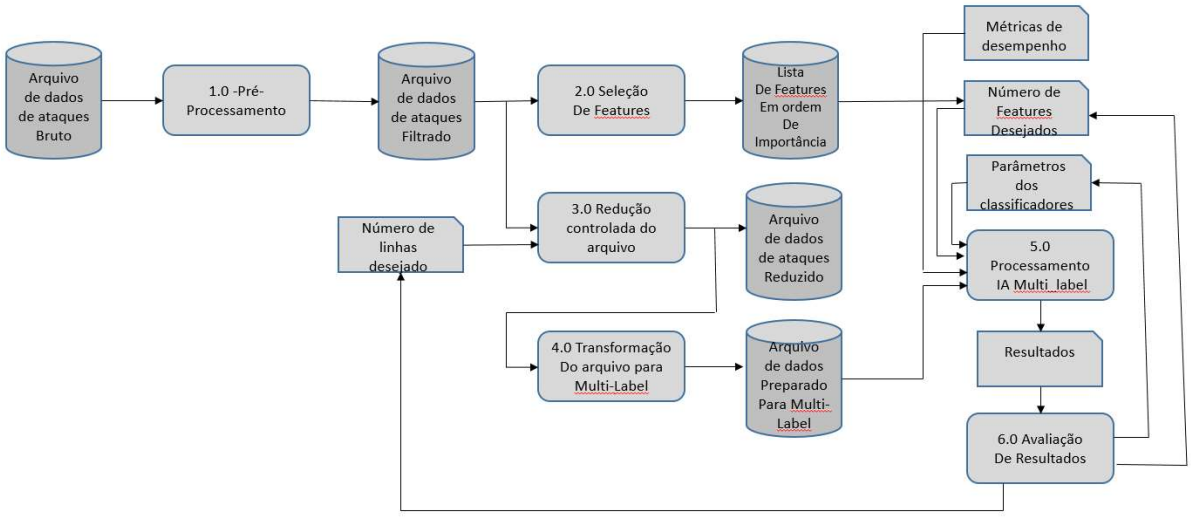


Figura 2: Processo de Detecção de Ataques *Multi-label*(do Autor)

2.1.5.1 *Etapa de Pré-processamento*

A etapa de pré-processamento (Etapa 1.0) trata o arquivo de dados gravados de ataques bruto quanto a possíveis inconsistências, como por exemplo: tipos de dados inadequados para processamento *(string* ao invés de número), número diferente de dados gravados em cada linha, entre outras. Essa etapa não foi realizada no TCC uma vez que os dados de ataques já filtrados foram aproveitados do trabalho desenvolvido por [5][6].

2.1.5.2 *Etapa de Seleção de Features*

A etapa de Seleção de *Features* (Etapa 2.0) estabelece uma ordem de prioridade com base na importância da *Feature* para a classificação do ataque. Essa importância pode ser determinada com base na teoria de correlação estatística ou na teoria da informação. A Seção 3.1 vai tratar das técnicas da Seleção de *Features*, da utilização da técnica de Floresta Aleatória (*Random Forest*) com esse objetivo e apresenta um novo algoritmo para seleção de *Features* com base no processamento

dos arquivos individuais para cada ataque (ver a Tabela 2) ao invés do processamento de um único arquivo com todos os ataques.

2.1.5.3 *Etapa de Redução Controlada do Arquivo*

A etapa de Redução Controlada do Arquivo (Etapa 3.0) possibilita reduzir o tamanho do arquivo gravado a fim de evitar um tempo de processamento muito alto dos classificadores. Como mostrado na tabela 2, o arquivo completo com todos os ataques possui 1.643.992 linhas. A redução busca manter a proporcionalidade de cada ataque sem reduzir o número de linhas dos ataques de menor contagem (*HeartBleed* e *Infiltration*). Essa redução de arquivo é abordada com mais detalhes na Seção 3.2.

2.1.5.4 *Etapa de Transformação do Arquivo para Multi-Label*

A etapa de Transformação do Arquivo para *Multi-Label* (Etapa 4.0) transforma uma única coluna do arquivo original contendo o *Label* de ataque ou Benigno em diversas colunas, uma para cada ataque ou Benigno com indicação binária de sua ocorrência ou não (1 ou 0), em preparação para o processamento pelos classificadores *Multi-Label*. Essa transformação é abordada com detalhes na Seção 3.3.

2.1.5.5 *Etapa de Processamento de Inteligência Artificial Multi-Label*

A etapa de processamento de Inteligência Artificial *Multi-Label* (Etapa 5) aplica as técnicas de classificação *Multi-Label* de Transformação do Problema e de Adaptção. Métricas de desempenho são coletadas para posterior avaliação de resultados. O arquivo para processamento é dividido em 2 conjuntos: um para treinamento e outro para testes. Essa é uma boa prática para evitar vícios do treinamento na realização dos testes. Os seguintes aspectos do problema podem ser variados para o processamento: número de linhas do arquivo reduzido, número de *Features* de acordo com a prioridade de importância, e estabelecimento dos parâmetros dos classificadores. Essa etapa é tratada com detalhes no Capítulo 3.

Devido à restrição de gravação de apenas um ataque por vez no conjunto de dados disponível, o problema de classificação de ataques deste trabalho é do tipo *Multi-Class* e não *Multi-Label*. Apesar disso, como as técnicas de processamento utilizadas se aplicam ao caso mais geral (*Multi-Label*), o processamento de Inteligência Artificial é referenciado no trabalho como *Multi-Label*. A Figura 3 ilustra os conceitos de processamento apresentados.

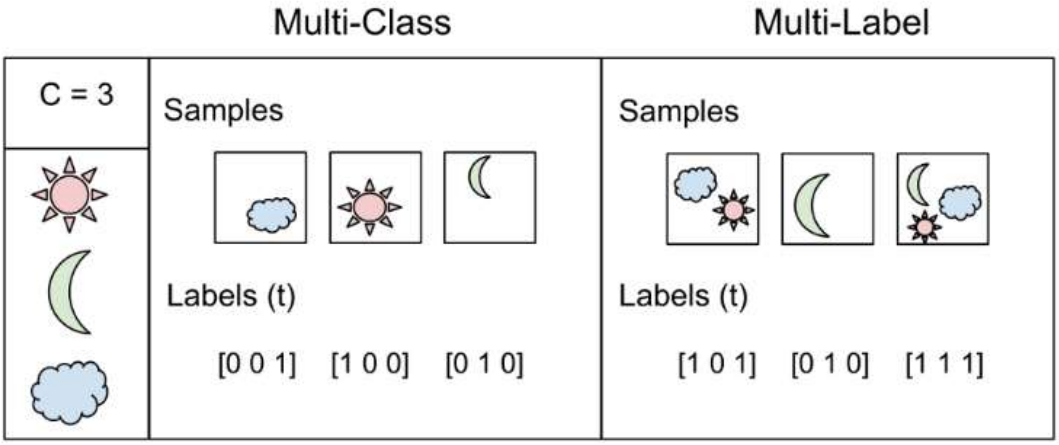


Figura 3: Diferença entre *Multi-Class* e *Multi-Label* (Retirado de [23])

2.1.5.6 *Etapa de Avaliação dos Resultados*

Na etapa de Avaliação dos Resultados (Etapa 6) é realizada a análise das saídas obtidas na Etapa 5 utilizando as métricas de desempenho coletadas e os tempos de processamento observados. Essa etapa é abordada em detalhes no capítulo 4.

.

2.1.6 Tipos de Ataques Realizados e Ferramentas Utilizadas

A Tabela 1 mostra os tipos de ataques realizados em cada dia da semana para gravação do *Dataset CIC-IDS2017* [5].

Dia da Semana	Ataques Realizados
Segunda-feira	Nenhum (tudo Benigno)
Terça-feira	*FTP-Patator, SSH-Patator*

Quarta-feira	*DoS Hulk, DoS GoldenEye, DoS slowloris, DoS Slowhttptest, Heartbleed*
Quinta-feira	*Web Attacks(Brute Force, XSS e Sql Injection) e Infiltration*
Sexta-feira	*Bot, DDoS e PortScan*

Tabela 1: Ataques realizados por dia da semana (Retirado de [5])

O APÊNDICE A apresenta um quadro com as ferramentas utilizadas para geração dos ataques. Os ataques contemplaram todas as categorias do DARPA descritas anteriormente em 2.1.

De forma geral foram gravados aproximadamente 30% de linhas de arquivo de de tráfego com ataques e 70% de linhas de arquivo de tráfego normal (Benigno). A quantidade de linhas geradas para cada tipo de ataque no arquivo de gravação é decorrente do período de tempo utilizado para gravação de cada ataque. A Tabela 2 apresenta a frequência de contagem de linhas para cada tipo de ataque gravado em arquivo separado.

Id do Arquivo	**Arquivo de Ataque**	**Nr de linhas de ataque**	**Nr Total de Linhas**
1	Bot.csv	1966	6567
2	DDoS.csv	41835	139568
3	DoS GoldenEye.csv	10293	34280
4	DoS Hulk.csv	231073	820490
5	DoS Slowhttptest.csv	5499	18162
6	DoS Slowloris.csv	5796	19377
7	FTP-Patator	7938	26483
8	Heartbleed.csv	11	41
9	Infiltration.csv	36	119
10	PortScan.csv	153930	551895
11	SSH-Patator.csv	5897	19791
12	WebAttack.csv	2180	7219
Totais		471454	1643992

Tabela 2: Frequência de contagem de cada ataque (Retirado de [5])

Obs: Nota-se que a amostragem não foi uniforme por cada ataque. É esperado que os ataques com maior contagem tenham maior precisão de classificação por serem treinados com um maior número de amostras. Tal diferença na contagem de ataques por classe pode resultar em problema de desbalanceamento de classes.

2.2 Técnicas utilizadas para classificar os dados em busca de padrões anômalos que possam caracterizar ataques à Segurança da Informação

Neste tópico, será apresentado o conjunto de técnicas que podem ser empregadas na detecção de anomalias no tráfego de dados em uma rede de compuatadores:

- Técnicas de Seleção de *Features*;
- Árvores de Decisão;
- Florestas Aleatórias (*Random Forests*);
- Classificação Binária;
- K Vizinhos mais próximos;
- Técnicas de Classificação *Multi-Label*;
- Métricas de Desempenho para os Classificadores *Multi-Label* de Ataques;
- Técnicas de Transformação do Problema para Classificação *Multi-Label*; e
- Técnicas de Adaptação para Classificação *Multi-Label*.

2.2.1 Técnicas de Seleção de *Features*

A seleção de *Features* tem o objetivo de reduzir a dimensionalidade dos dados a serem processados para classificação. Nesse trabalho, pode ser considerado que o termo "feature" é sinônimo de cararacterísticas ou propriedades disponíveis nas linhas de um *Dataset (*Conjunto de dados gravados.de rede de computador a ser utilizado para o processamento de Inteligência Artificial). A Figura 4 ilustra as colunas com as *Features* (x1 a x85) em um arquivo de forma tabular.

Features									*Labels*
X1	X2	X3	X4	X5	X6	X7	...	X85	y
									Benigno
									DoS
									Web Attack
									Benigno
									Benigno
									SSH-Patator
									Benigno
									Benigno
									DoS
									Web Attack
									SSH-Patator
									Benigno

Figura 4: *Features* em Arquivo *Dataset* na forma Tabular (Do Autor)

O trabalho [9] aponta que a Seleção de *Features* é baseada em medidas como ganho de informação, correlação ou informação mútua e no conhecimento prévio do Analista de Dados para filtrar as *Features* que não sejam importantes ou que sejam redundantes.

O trabalho [24] apresenta o conceito de relevância da *Feature*, que pode ser classificada com forte ou fraca. Uma *Feature* é de forte relevância quando sua retirada do processamento provocar degradação do desempenho do classificador. Por outro lado, uma *Feature* é considerada de relevância fraca quando sendo parte de um conjunto de *Features* retirados do processamento não provocar degradação do desempenho do classificador. As *Features* são irrelevantes se quando retiradas seja isoladamente ou como parte de um sub-conjunto não provocarem degradação do desempenho do classificador.

O trabalho [25] identificam um conjunto de *Features* como relevante se tem alta correlação com a classe e baixa correlação entre si em termos de informação mútua.

2.2.2 Árvores de Decisão

Uma das alternativas possíveis para realização de Classificadores é utilizar Árvores de Decisão [26]. Uma Árvore de Decisão também se depara com a questão de selecionar as *Features* que devem ser consideradas primeiro em sua construção. A Figura 5 mostra um exemplo de Árvore de Decisão.

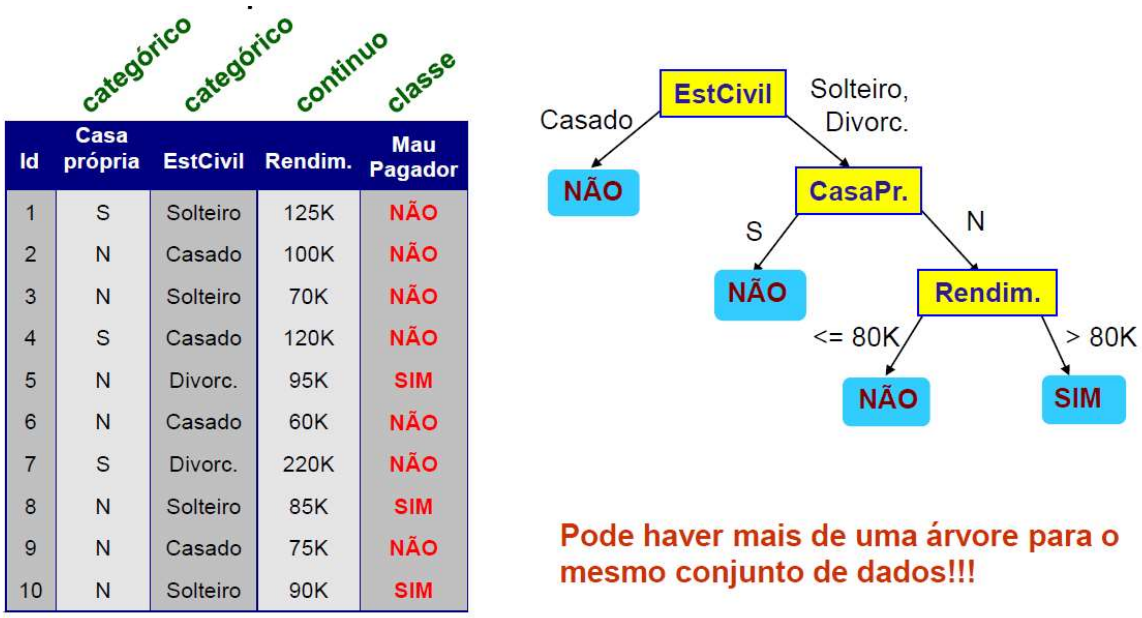


Figura 5: Exemplo de Árvore de Decisão (Retirado de: [26]

O Algorítmo ID3 [26] utiliza seleção de *Features* para construção da Árvore de Decisão. Nesse algoritmo a seleção dos nós é baseada na Teoria da Informação de Shannon, mas especificamente nos conceitos de Entropia e Ganho de Informação.

2.2.3 Florestas Aleatórias (*Random Forests*)

O Autor de [27] define Floresta Aleatória como um classificador que consiste de uma coleção de classificadores baseados em árvores estruturadas. A Floresta Aleatória também pode ser utilizadada para Regressão. A Figura 6 mostra um exemplo de um classificador por Floresta Aleatória.

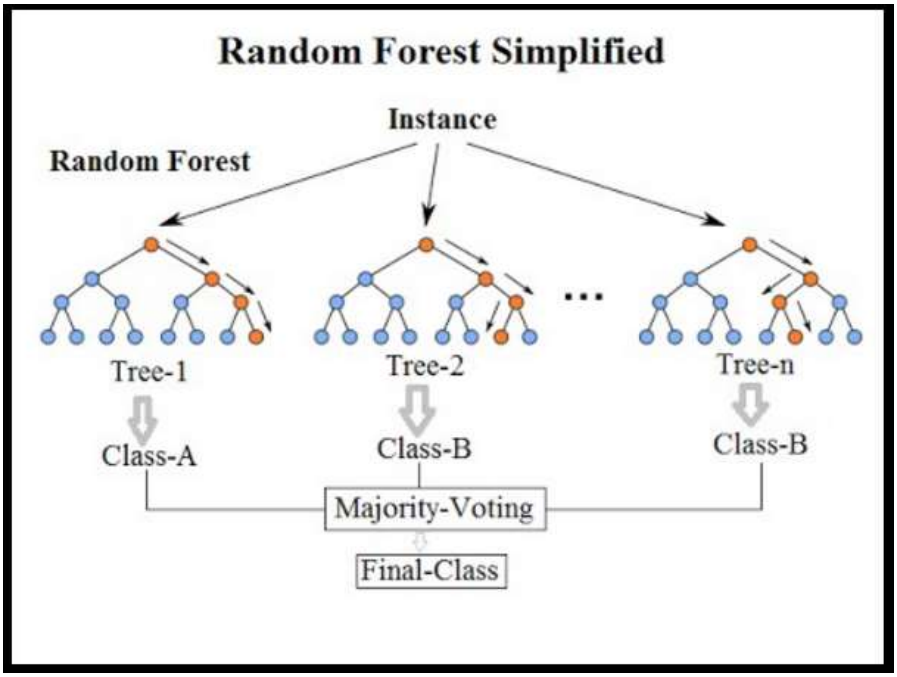


Figura 6: Floresta Aleatória (Extraído de [40])

A Tabela 3 mostra um exemplo de código para o uso da classe *RandomForestRegressor*. Nessa chamada são construídas 250 árvores aleatórias (*n_estimators* = 250), utilizando uma semente zerada (*random_state* = 0) para o gerador aleatório interno à classe e configurando o processador para utilizar todos os seus núcleos em processamento paralelo (*n_jobs* = -1). Após a construção do objeto *RandomForestRegressor*, é realizada a regressão (fit) a partir das matrizes de *Features* X e de *Labels* (y). A propriedade feature_importances devolve a lista de importância para as *Features* de tal forma que a soma dos itens dessa lista seja igual a um.

```
from sklearn.ensemble import RandomForestRegressor


forest = sk.ensemble.RandomForestRegressor(n_estimators=250, random_state=0, n_jobs=-1)
forest.fit(X, y)
importances = forest.feature_importances_
```

Tabela 3: Exemplo de chamada do *RandomForestRegressor* (adaptado de [6])

2.2.4 - Classificação Binária

O processamento de Inteligência Artificial por classificação Binária é do tipo Supervisionado (requer a gravação de labels dos ataques a serem utilizados na fase de treinamento) e tem por objetivo detectar se houve ataque ou não, não se importando em indicar que tipo de ataque foi realizado. A Figura 7 mostra como o arquivo de gravação é preparado para o processamento de classificação binária. O arquivo original possui uma coluna de *Labels* com as *Strings* correspondentes às classes de ataque ou Benigno. No arquivo transformado, a coluna *Labels* apresenta o valor binário 1 na ocorrência de qualquer tipo de ataque e o valor binário 0 para Benigno. Como a informação do tipo de ataque não é utilizada no arquivo transformado para processamento binário, esse classificador não possibilita indicar o tipo de ataque, mas é útil para gerar um alarme sobre a ocorrência de qualquer ataque.

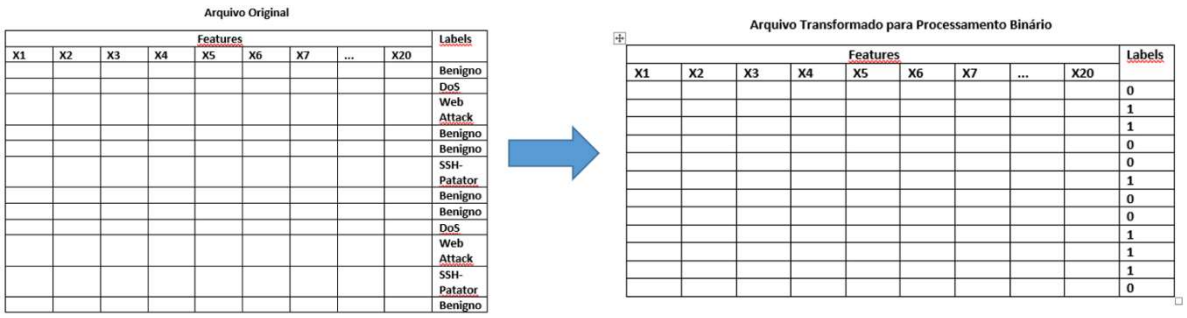


Figura 7: Arquivo Transformado para Processamento Binário (Do Autor)

2.2.5 - K Vizinhos Mais Próximos

O classificador binário do tipo K Vizinhos Mais Próximos procura os K vizinhos mais próximos em distância da amostra a ser classificada aos elementos da Matriz de Treinamento (linhas de *Features*). A classificação se dá pela maior frequência de contagem de um *Label* nos K vizinhos mais próximos.

A Figura 8 mostra a técnica utilizada para K Vizinhos mais próximos: os dados do conjunto treinamento com as *Features* <x1, x2> são plotados no plano bi-dimensional indicando 2 possíveis classes (quadrado azul e triangulo vermelho). Uma nova amostra a ser classinficada é mostrada no centro (circulo verde com interrogação). Dentro do círculo interno há apenas um vizinho mais próximo da

amostra a ser classificada (K =1). Dentro do círculo externo há três vizinhos mais próximos da amostra a ser classificada (K = 3). Se o classificador estabelecer K igual a 1 ele vai decidir pela classe quadrado azul (única possível dentro do círculo interno). Se o classificador estabelecer K igual a 3 ele vai decidir pela classe triangulo vermelho (maior contagem dentro do circulo externo).

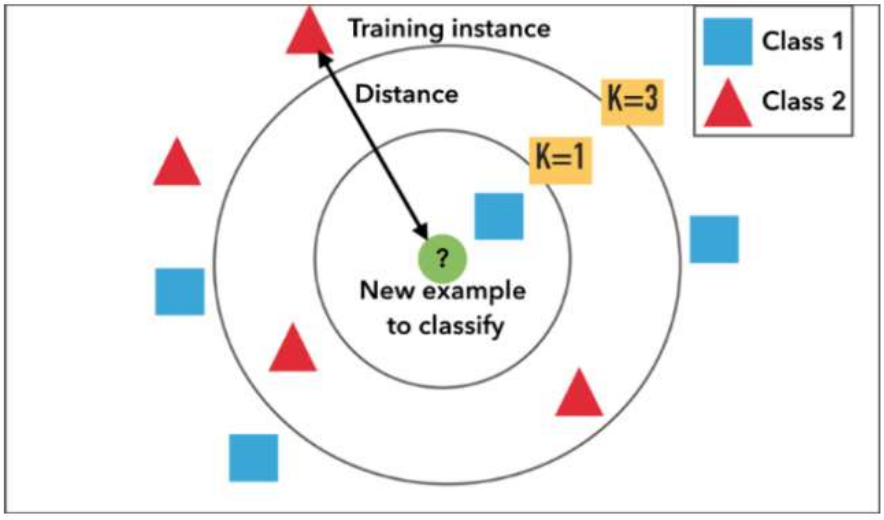


Figura 8: Classificação Binária por K Vizinhos Mais próximos (Retirado de [30])

2.2.6 Classificação *Multi-Label*

O processamento de Inteligência Artificial por classificação *Multi-Label* é do tipo Supervisionado (requer a gravação de labels dos ataques a serem utilizados na fase de treinamento) e tem por objetivo detectar se houve ataque ou não, indicando o tipo de ataque realizado ou normalidade (Benigno).

A Figura 9 apresenta uma visão geral das técnicas de classificação de processamento por Inteligência Artificial *Multi-Label*, indicando:

- Duas técnicas para abordagem da questão de classificação *Multi-label*: Transformação do Problema e Adaptação. Ambas as técnicas são apoiadas pela Biblioteca Python SkMultiLearn [8];
- A técnica de Transformação do Problema possibilita a utilização de três abordagens para essa transformação (Relevância Binária, Cadeia de Classificadores *e Label PowerSets*) que podem ser utilizadas com

diferentes tipos de classificadores binários, entre estes: K Vizinhos Mais Próximos, Floresta Aleatória e Árvore de Decisão ID-3;

- A técnica de Adaptação possibilita a utilização da classe Python *GridSearchCV* para busca automática dos melhores parâmetros de entrada dos classificadores por Adaptaçao;
- Dois algorítimos estão disponíveis para Adaptação: ML-KNN (*Multi-Label K Nearest Neighbors*) e BR-KNN (*Binary Relevance K Nearest Neighbors*);
- O Algorítimo ML-KNN *(Multi-Label K Nearest Neighbors*) combina um classificador K Vizinhos mais próximos com um classificador baseado na aplicação do Teorema de Bayes; e
- O classificador BR-KNN (*Binary Relevance K Nearest Neighbors*) otimiza o tempo de processamento da combinação da transformação por Relevância Binária com um classificador K Vizinhos Mais Próximos.

A técnica de Transformação do Problema procura transformar o problema de classificação *Multi-Label* em um conjunto de problemas de classificação binária. Dessa forma é possível utilizar classificadores binários para solução do problema de classificação *Multi-Label*.

A técnica de Adaptação busca adaptar os algoritmos de classificação binária para a classificação Multi-Label.

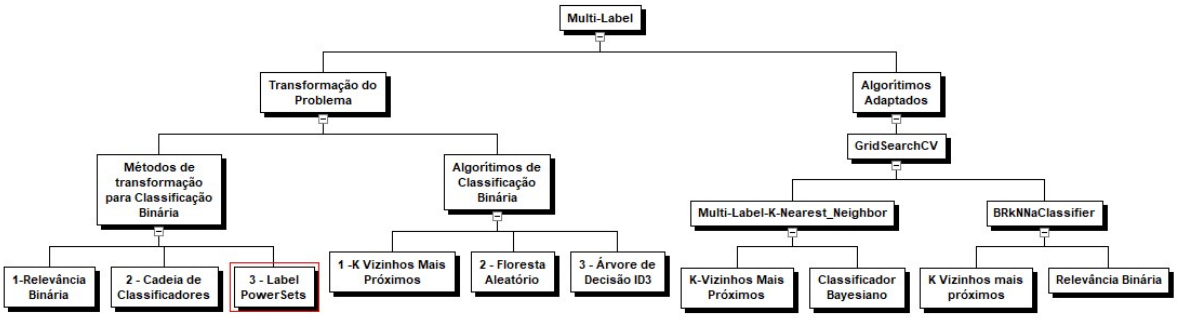


Figura 9: Técnicas para classificação *Multi-Label* (Do Autor)

2.2.7 Métricas de Desempenho para os Classificadores *Multi-Label* de Ataques

A tese de mestrado de [5] descreve as métricas utilizadas para avaliação de desempenho de classificação binária: Precisão, *Recall e F1-Measure*.

O capítulo 7 do livro de referência de [9] descreve as principais métricas utilizadas para medida de desempenho de classificadores *Multi-Label*, entre elas: Matriz de Confusão, Acurácia, e *Hamming Loss*. A referência [33] ressalta que as métricas para classificação *Multi-Label* são mais elaboradas do que as utilizadas para classificação binária.

A Figura 10 apresenta a Matriz de Confusão para um problema de classificação com 4 labels: normal (Benigno), *DoS* Probe e U2R.

Predicted class

Actual class		Normal	DoS	Probe	U2R
	Normal	38	1	2	4
	DoS	4	31	0	0
	Probe	4	0	11	0
	U2R	1	0	1	3

Figura 10: Matriz de Confusão (Retirado de: [9])

Resumidamente, essa matriz indica a contagem de acertos de classificação em sua diagonal e a contagem de erros em suas partes triangulares. A métrica de Acurácia indica o percentual de acertos do classificador enquanto a métrica de *Hamming-Loss* indica a probabilidade de erro de classificação utilizando a Teoria de Códigos de Hamming.

2.2.8 Técnicas de Transformação do Problema para Classificação *Multi-Label*

Esta Seção aborda as seguintes técnicas de Transformação do Problema para classificação *Multi-Label*:

- Relevância Binária;

- Cadeia de Classificadores; e
- *Label PowerSets*.

2.2.8.1 Relevância Binária

A Relevância Binária é um método popular de classificação que procura aprender a partir de um conjunto de classificadores binários, um para cada Label ou rótulo [31]. A Figura 11 ilustra o método de Transformação do Problema por Relevância Binária. Nessa Figura uma Tabela com uma coluna de rótulos de classificação é transformada em 4 tabelas, uma para cada rótulo. A desvantagem desse método é que a correlação entre os *Labels* é desconsiderada pois os *Labels* são tratados em tabelas independentes. Os métodos a serem apresentados em seguida, Cadeia de Classificadores e *Label PowerSets*, buscam corrigir a questão da perda da correlação entre os *Labels*.

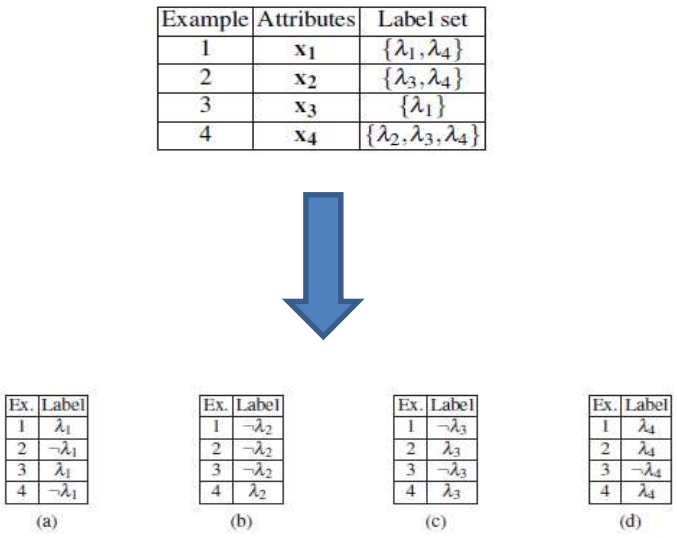


Figura 11: Transformação do Problema por Relevância Binária
(Adaptado de [31])

A tabela 4 apresenta um exemplo de utilização do método de transformação do problema por Relevância Binária em conjunto com um classificador binário de K Vizinhos mais próximos a partir das Bibliotecas *Python SkMultilearn*[8] e *SkLearn*[7].

```
from sklearn.neighbors import KNeighborsClassifier
```

```
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.metrics import accuracy_score,hamming_loss



classif = BinaryRelevance(KNeighborsClassifier(3))

classif.fit(xtrain, ytrain)
classif_predictions = classif.predict(xtest)
acc = accuracy_score(ytest, classif_predictions)
ham = hamming_loss(ytest, classiff_predictions)
```

Tabela 4: Exemplo de chamada Python para Relevância Binária (Extraído de [35])

2.2.8.2 Cadeia de Classificadores

O método de Cadeia de Classificadores busca utilizar a saída de um classificador binário como entrada para o próximo classificador ao adicionar uma coluna de Label (ex: yi, i = 1,2, 3) como uma *Feature*, como mostrado na Figura 12.

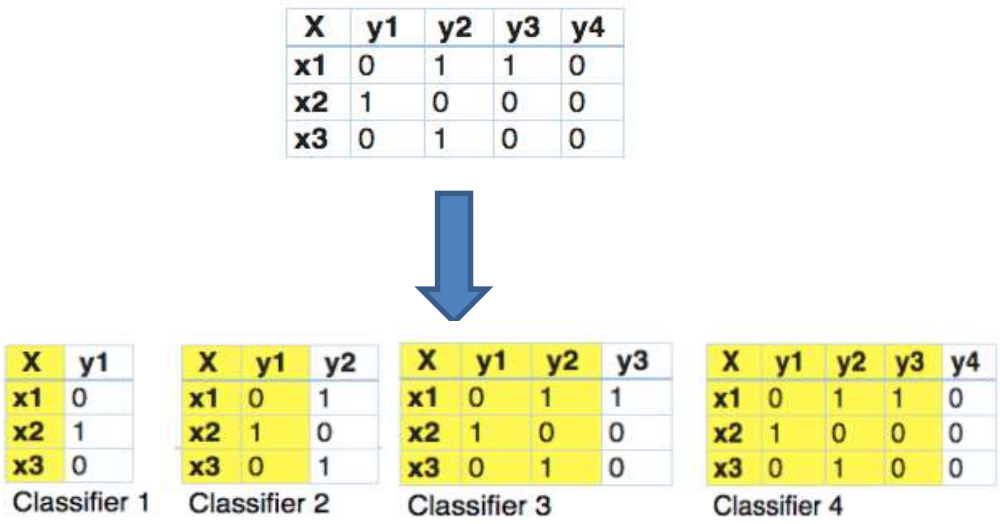


Figura 12: Transformação do Problema por Cadeia de Classificadores (Adaptado de [32])

A tabela 5 apresenta um exemplo de utilização do método de transformação do problema por Cadeia de Classificadores em conjunto com um classificador binário de Árvore de Decisão ID-3 a partir das Bibliotecas Python *SkMultilearn*[8] e *SkLearn*[7].

```
from sklearn.tree import DecisionTreeClassifier
```

```
from skmultilearn.problem_transform import ClassifierChain
from sklearn.metrics import accuracy_score,hamming_loss



classif = ClassifierChain(DecisionTreeClassifier(max_depth=5, criterion="entropy"))

classif.fit(xtrain, ytrain)
classif_predictions = classif.predict(xtest)
acc = accuracy_score(ytest, classif_predictions)
ham = hamming_loss(ytest, classiff_predictions)
```

Tabela 5: Exemplo de chamada Python para Cadeia de Classificadores (Extraído de [35])

2.2.8.3 Label PowerSets

O método de *Label PowerSets* busca agrupar as diferentes combinações de *Labels* individuais. A Figura 13 ilustra o método de Transformação do Problema por *Label PowerSets* na qual observa-se a combinação de classes individuais de *Labels* gerando novas classes por agrupamento.

Example	Attributes	Label set
1	x_1	$\{\lambda_1, \lambda_4\}$
2	x_2	$\{\lambda_3, \lambda_4\}$
3	x_3	$\{\lambda_1\}$
4	x_4	$\{\lambda_2, \lambda_3, \lambda_4\}$

Ex.	Label
1	$\lambda_{1,4}$
2	$\lambda_{3,4}$
3	λ_1
4	$\lambda_{2,3,4}$

Figura 13: Transformação do Problema por *Label PowerSets* (Adaptado de [31])

A tabela 6 apresenta um exemplo de utilização do método de transformação do problema por *Label PowerSets* em conjunto com um classificador binário de Floresta aleatória a partir das Bibliotecas Python *SkMultilearn*[8] e *SkLearn*[7].

```
from sklearn.ensemble import RandomForestClassifier
from skmultilearn.problem_transform import LabelPowerset
from sklearn.metrics import accuracy_score,hamming_loss


classif = LabelPowerset (RandomForestClassifier(max_depth=5, n_estimators=10)


classif.fit(xtrain, ytrain)
classif_predictions = classif.predict(xtest)
acc = accuracy_score(ytest, classif_predictions)
ham = hamming_loss(ytest, classiff_predictions)
```

Tabela 6: Exemplo de chamada Python para *Label PowerSets*(Extraído de [35])

2.2.9 Técnicas de Adaptação para Classificação Multi-Label

Esta Seção aborda as seguintes técnicas de Adaptação para classificação *Multi-Label*:

- *ML-KNN (Multi-Label K Nearest Neighbors)*; e
- *BR-KNN (Binary Relevance K Nearest Neighbors)*

2.2.9.1 Multi-Label K Nearest Neighbors

A técnica de *ML-KNN (Multi-Label K Nearest Neighbors)* utiliza o procedimento de K Vizinhos Mais Próximos combinado com um método de classificação baseado no cálculo da probabilidade da intercessão da ocorrência dos *Labels* pelo Teorema de Bayes. Esse cálculo é utilizado para a classificação ao invés da simples identificação da classe de maior frequência entre as classes dos K Vizinhos Mais Próximos. Essa técnica está descrita em [33].

A Tabela 7 apresenta um exemplo de chamada de código Python para implementação do classificador ML-KNN utilizandos as Bibliotecas *SkLearn*[7] e *MultilLearn*[8].

```
from sklearn.metrics import accuracy_score,hamming_loss
from sklearn.model_selection import GridSearchCV
from skmultilearn.adapt import MLkNN

score = ‘f1_weighted’
parametros = {'k': range(1, 3), 's': [0.5, 0.7, 1.0]}
clf = GridSearchCV(MLkNN(), parametros, scoring=score)
```

```
clf.fit(Xarr_train, yarr_train)
clf_predictions = clf.predict(Xarr_test)
acc = accuracy_score(yarr_test, clf_predictions)
ham = hamming_loss(yarr_test, clf_predictions)
```

Tabela 7: Exemplo de chamada Python para ML-KNN (Adaptado de [36])

2.2.9.2 Binary Relevance K Nearest Neighbors

A técnica de BR-*KNN (Binary Relevance K Nearest Neighbors)* otimiza o tempo de execução da solução obtida por Transformação do Problema ao combinar o classificador binário K Vizinhos mais próximos com o método de transformação do problema por Relevância Binária. Na Transformação do Problema a procura pelos K vizinhos mais próximos é realizada de forma redundante nas 4 tabelas mostradas na Figura 11. Ao realizar essa busca apenas uma vez tem-se uma redução do tempo de processamento da ordem do número de *Labels*. Essa técnica está descrita em [34].

A Tabela 8 apresenta um exemplo de chamada de código Python para implementação do classificador BR-KNN utilizandos as Bibliotecas *SkLearn*[7] e *MultilLearn*[8].

```
from sklearn.metrics import accuracy_score,hamming_loss
from sklearn.model_selection import GridSearchCV
from skmultilearn.adapt import BRkNNaClassifier


score = 'f1_weighted'
parametros = {'k': range(3, 5)}
clf = GridSearchCV(BRkNNaClassifier(), parametros, scoring=score)

clf.fit(Xarr_train, yarr_train)
clf_predictions = clf.predict(Xarr_test)
acc = accuracy_score(yarr_test, clf_predictions)
ham = hamming_loss(yarr_test, clf_predictions)
```

Tabela 8: Exemplo de chamada Python para BR-KNN(Adaptado de [36])

3 TESTES REALIZADOS

Este capítulo aborda os testes experimentais realizados para as propostas de adequações/melhorias das técnicas para a detecção de ameaças em ambiente de redes de computadores, na execução do presente TCC, para:

- Implementação de Novo Algoritmo de Seleção de Features por Florestas Aleatórias (*Random Forests*);
- Redução Controlada do Arquivo de Gravação de Dados de Ataques;
- Transformação dos arquivos para processamento por classificador *Multi-Label*;
- Implementação em Python do Classificador *Multi-label* por Transformação do Problema;
- Implementação em Python do Classificador *Multi-label* por Adaptação.

3.1 Implementação de Novo Algoritmo de Seleção de Features por Florestas Aleatórias (*Random Forests*)

Objetivo: Obter uma lista de prioridade de seleção de *Features* para o *Dataset* utilizado no TCC com menor tempo de processamento possível a partir das importâncias de Features resultantes do processamento por Floresta Aleatória dos arquivos separados por tipo de ataque no trabalho de [5].

Esse teste busca uma alternativa mais rápida e com baixa perda de qualidade em relação a seleção de Features realizada pelo processamento de um arquivo único com todos os ataques [5]. Esse último obtém a Lista de Importâncias do APÊNDICE B, com um tempo de processamento multo alto de 17 horas.

Resultado(s) esperado(s): Uma Lista de importâncias para as *Features* obtida com baixo tempo de processamento adicional em relação ao tempo do processamento por Floresta Aleatória dos arquivos separados por tipo de ataque no trabalho de [5], que é de cerca de 1,7 horas. Espera-se um tempo de processamento adicional muito menor, com baixa perda de importância acumulada para a seleção de 20 Features do *Dataset* utilizado no TCC.

Descrição do teste: Para a realização do teste em questão foi implementado um novo algoritmo em Python cujo pseudocódigo está apresentado na Tabela 10. Esse algoritimo processa as listas de importância obtidas para cada ataque em separado, e calcula a importância de cada *Feature* para todos os ataques pela média ponderada da importância da *Feature* específica para cada ataque com base na frequência de contagem de cada ataque.

A solução obtida em [5] utilizou os pseudocódigos mostrados na Tabela 9. O lado esquerdo dessa tabela mostra a geração da lista de importâncias de *Features* a partir de um arquivo único com todos os ataques. O lado direito dessa tabela mostra a geração de listas de importâncias para cada um dos ataques a partir dos arquivos separados para cada ataque.

Arquivo único com todos os ataques	**Arquivos separados para cada ataque**
Calcular a Regressão por Floresta Aleatória para o arquivo único Salvar a lista de importâncias de *Features* para o arquivo único	Para cada arquivo de ataque: Calcular a Regressão por Floresta Aleatória para cada arquivo Salvar a lista de importâncias de *Features* para cada ataque

Tabela 9: Pseudocódigos da solução obtida em [5] (Do Autor)

```
Para cada ataque:
    Acumular o número total de linhas de ataques
Para cada ataque:
    Obter a contagem do ataque
    Obter o peso de cada ataque em relação ao número total de linhas de ataques
Para cada ataque:
    Obter a lista de importâncias das features
    Para cada feature:
        Acumular a parcela:  importância da feature para o ataque * peso
Ordenar a lista de importância de Features em ordem decrescente de importância
Salvar a lista de importância de Features em arquivo
```

Tabela 10: Pseudo-código para Geração de Lista de Importâncias (Do Autor)

O código Python para essa implementação consta do APÊNDICE C (FeaturesImportance.py). O processamento é realizado em dezenas de segundos e obtem-se a lista de importâncias do APÊNDICE D.

O grau de aderência indica o quanto em percentual o conteúdo da lista original de seleção foi modificado pela nova lista de *Features* selecionadas (100% subtraído do grau de aderência).

A redução da importância acumulada é mais importante que o grau de aderência pois mede o quanto em percentual a nova lista de seleção é menos eficaz para utilização pelos classificadores do que a lista original.

No sentido de obter o grau de aderência e o percentual de redução da importância acumulada para as *Features* selecionadas pela nova alternativa de Seleção e gerar gráficos de avaliação, foi implementado o pseudocódigo da Tabela 11.

```
Selecionar o número de Features a selecionar  igual a 20
Calcular o grau de aderência para a nova Lista de Importâncias
Plotar o Gráfico de Grau de Aderência
Para cada Lista de Importâncias:
  Construir um dicionário Python contendo ( Feature : Importância) em ordem decrescente de
importância da Feature
   Para cada Feature:
     Acumular o valor de importância
   Plotar Gráficos de Avaliação de Importância Acumulada
```

Tabela 11: Pseudo-código para Importância Acumulada de *Features* (Do Autor)

O código Python do APÊNDICE E (FeatureImportaceAnalysis.py) faz a análise das duas listas de importância de Features (APÊNDICEs B e D), calculadas pelo algoritmo original e pelo novo algoritimo apresentado neste TCC.

3.2 Redução Controlada do Arquivo de Gravação de Dados de Ataques

Objetivo: Obter um arquivo com os dados de gravação de todos os ataques com número de linhas reduzido a partir da proporção de cada ataque nos arquivos originais separados por ataque. A redução do arquivo visa possibilitar tempos de

processamento menores para os classificadores de Inteligência Artificial de forma a permitir a observação de resultados sem gastar um tempo excessivo na sua execução, considerando as limitações de tempo para realização deste trabalho e a utilização de um *Home PC* para processamento. Essa redução deverá preservar a contagem mínima dos ataques de menor frequência (*Heartbleed e Infiltration*)

Resultado(s) esperado(s): Dispor de arquivo contendo todos os ataques de forma proporcional em relação a contagem dos ataques nos arquivos originais com um determinado número de linhas pré-selecionado, exemplo (150.000 linhas). A redução mantém a contagem mínima dos ataques *HearttBleed* e *Infiltration* por compensação na contagem dos ataques de maior frequência (*Hulk* e *PortScan*).

Descrição do Teste: A Tabela 12 apresenta o pseudocódigo para geração de arquivos de dados de ataques com número de linhas reduzido de forma controlada a partir da proporção de cada ataque no arquivo original. O APÊNDICE F apresenta o código Python para geração de arquivos reduzidos com os seguintes número de linhas: 50.000 linhas, 100.000 linhas, 150.000 linhas, 200.000 linhas e 250.000 linhas.

```
Gerar um dicionário de ataques (ataque: (contador de linhas de ataque, contador total de linhas))
Calcular o peso de cada ataque:
peso = (contagem de linhas do ataque/ soma da contagem de linhas de todos ataques)
Para cada número de linhas de redução desejado:
  Obter o número de linhas desejadas para cada ataque:
  Número de linhas = peso * 30% tamanho em linhas do arquivo reduzido
  Obter o número de linhas desejadas para Benigno:
  Número de linhas = peso * 70% tamanho em linhas do arquivo reduzido
  Para cada arquivo de ataque:
    Selecionar o número de linhas desejado para cada ataque ou Benigno
    dos arquivos originais (First Come First Served)
  Ajustar para manter o número de linhas dos ataques HeartBleed e Infiltration
  Salvar arquivo reduzido
```

Tabela 12: Pseudo-código para Geração de Arquivos Reduzidos (Do Autor)

3.3 Transformação dos arquivos para processamento por classificador *Multi-Label*

Objetivo: Transformar o arquivo reduzido com todos os ataques, obtido na seção anterior, para um arquivo com formato aceitável pelos classificadores *Multi-Label* da biblioteca Python *SkMultiLearn*

Resultado(s) esperado(s): A Figura 14 mostra como o arquivo de gravação é preparado para o processamento de classificação Multi-Label. Uma única coluna com todos as Strings dos *Labels* no arquivo original é transformada em várias colunas no arquivo transformado com a indicação de presença (binário 1) ou ausência (binário 0) para cada ataque ou Benigno. No exemplo da Figura 19 são geradas colunas para os ataques DoS, Web Attack e SSH-Patator, e também para Benigno.

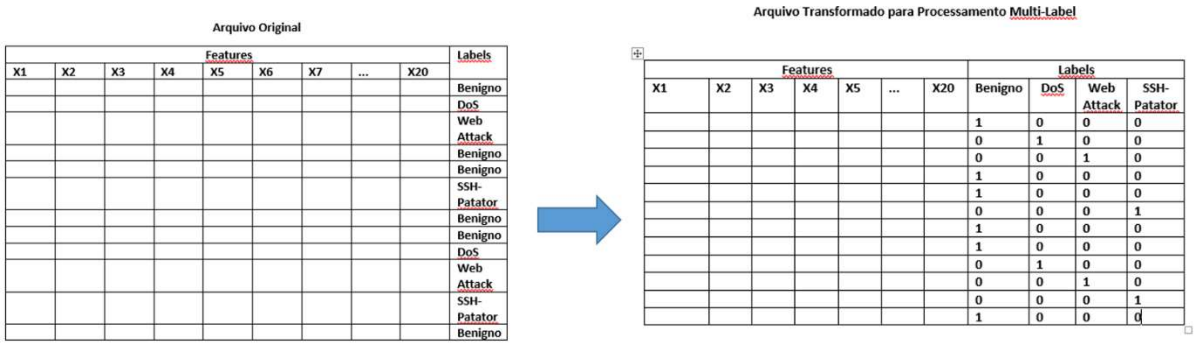


Figura 14: Arquivo Transformado para Processamento Multi-Label (Do Autor)

Descrição do Teste: Foi desenvolvido o programa em Python do APÊNDICE G para realização do teste.

3.4 Implementação em Python de Classificadores *Multi-Label* por Transformação do Problema

Objetivo: Verificar e comparar o desempenho de diversas opções de classificadores *Multi-Label* no processamento dos arquivos transformados da seção anterior, incluindo: a combinação das técnicas de Transformação do Problema

(Relevância Binária, Cadeia de Classificadores e *Label PowerSets*) com os classificadore binários (K Vizinhos Mais Próximos, Floresta Aleatório e Árvore de Decisão ID-3), a variação do número de *Features* utilizadas e a variação do número de linhas do arquivo transformado.

Resultado (s) esperado(s): Serão utilizadas as métricas de desempenho de Acurácia e *Hamming Loss* para avaliação dos resultados de classificadores *Multi-Label* por Transformação do Problema. Espera-se resultados acima de 94% de Acurácia (valor inferido a partir dos resultados do trabalho realizado em [5]). Não há referência para o valor esperado da *Hamming Loss* mas espera-se que ela seja menor que 0,5% na maioria das opções. Também será verificado o desempenho dos classificadores *Multi-Label* por Transformação através da medição do tempo de processamento de cada um deles.

Descrição do Teste: Foi implementado um programa em Python para processamento de arquivos de ataques reduzidos e obtenção de resultados de classificação de *Multi-Label* variando-se os seguintes aspetos do problema: Número de *Features* selecionadas em ordem de importância (20, 40 e 60 Features), número de linhas da arquivo de dados gravados de ataques reduzidos (de 50.000 a 250.000 linhas em passos de 50.000 linhas), a técnica de transformação do problema (Relevância Binária, Cadeia de Classificadores e *Label PowerSets*) e o classificador binário (K Vizinhos Mais Próximos, Floresta Aleatória e Árvore de Decisão ID-3).

A Tabela 13 mostra o pseudocódigo para o programa Python utilizado para essa implementação. Nessa tabela, *Xtrain* e Xteste são as matrizes de *Features*, *ytrain* e yteste são as matrizes com os *labels* de classificação do tipo do ataque ou de Benigno, utilizados, respectivamente, para treino do classificador e para teste de sua utilização como classificador. O APÊNDICE H mostra o código Python para essa implementação.

```
Para cada Número de Features {20, 40, 60}:
    Para cada Número de Linhas de arquivo [50.000, 250.000]:
        Selecionar 70% de dados para treinamento (Xtrain, ytrain)
        Selecionar 30% de dados para testes (Xteste, yteste)
        Para cada técnica de transformação do problema:
            Para cada tipo de classificador binário:
```

Construir o objeto classif baseado na tupla
(Transf do problema, classificador binário)
Treinar o classificador (classif.fit(Xtrain, ytrain))
Testar o classificador (classif.predict(Xtest))
Obter as métricas de Acurácia e Hamming Loss(usar ytest)
Salvar resultados em arquivo para posterior análise

Tabela 13: Pseudocódigo para a Implementação e Teste para Classificação *Multi-Label* por Transformação Do Problema(Do Autor)

3.5 Implementação em Python do Classificador *Multi-label* por Adaptação

Objetivo: Verificar e comparar o desempenho de diversas opções de classificadores *Multi-Label* no processamento dos arquivos transformados da seção 4.3, incluindo: as técnicas de Adaptação ML-KNN (Multi-Label e K Vizinhos Mais Próximos) e BR-KNN (Relevãncia Binária e K Vizinhos Mais Próximos), a variação do número de *Features* utilizadas, a variação do número de linhas do arquivo transformado e a variação dos parâmetros dos classificadores.

Resultado(s) esperado(s): Os resultados esperados são os mesmos do item 4.4.2, utilizando a técnica de Adaptação ao invés de Transformação do Problema para classificação *Multi-Label*.

Descrição do Teste: Foi implementado um programa em Python para processamento de arquivos de ataques reduzidos e obtenção de resultados de classificação de *Multi-Label* por Adaptação variando-se os seguintes aspetos do problema: Número de Features selecionadas em ordem de importância (20, 40 e 60 Features), número de linhas de arquivo de dados gravados de ataques reduzidos (de 50.000 a 150.000 linhas em passos de 50.000 linhas) e a técnica de adaptação (ML-KNN e BR-KNN).

A Tabela 14 mostra o pseudocódigo para o programa Python utilizado para implementação e testes de classificadores *Multi-Label* utilizando a técnica de Adaptação. Nessa tabela, Xtrain e Xteste são as matrizes de Features, ytrain e yteste são as matrizes com os labels de classificação do tipo do ataque ou de Benigno, utilizados, respectivamente, para treino do classificador e para teste de sua utilização

para a classificação. O APÊNDICE I mostra o código Python para essa implementação.

```
Para cada Número de Features {20, 40, 60}:
    Para cada Número de Linhas de arquivo [50.000, 150.000]:
        Selecionar 70% de dados para treinamento (Xtrain, ytrain)
        Selecionar 30% de dados para testes (Xteste, yteste)
         Para cada técnica de adaptação:
            Construir o objeto classif baseado em:
            (Técnica de Adaptação, Parâmetros)
            Treinar o classificador (classif.fit(Xtrain, ytrain))
            Testar o classificador (classif.predict(Xtest))
             Obter as métricas  de Acurácia e Hamming Loss(usar ytest)
              Salvar resultados em arquivo para posterior análise
```

Tabela 14: Pseudocódigo para a Implementação e Teste de Classificação Multi-Label por Adaptação (Do Autor)

4 RESULTADOS DOS TESTES

4.1 Implementação de Novo Algoritmo de Seleção de Features por Florestas Aleatórias (*Random Forests*)

A partir do experimento, observou-se que a redução de tempo de processamento da ordem da razão do tamanho do arquivo com todos os ataques pela média de tamanho dos arquivos de ataque separados. A redução do tempo de processamento é mostrada na Figura 15.

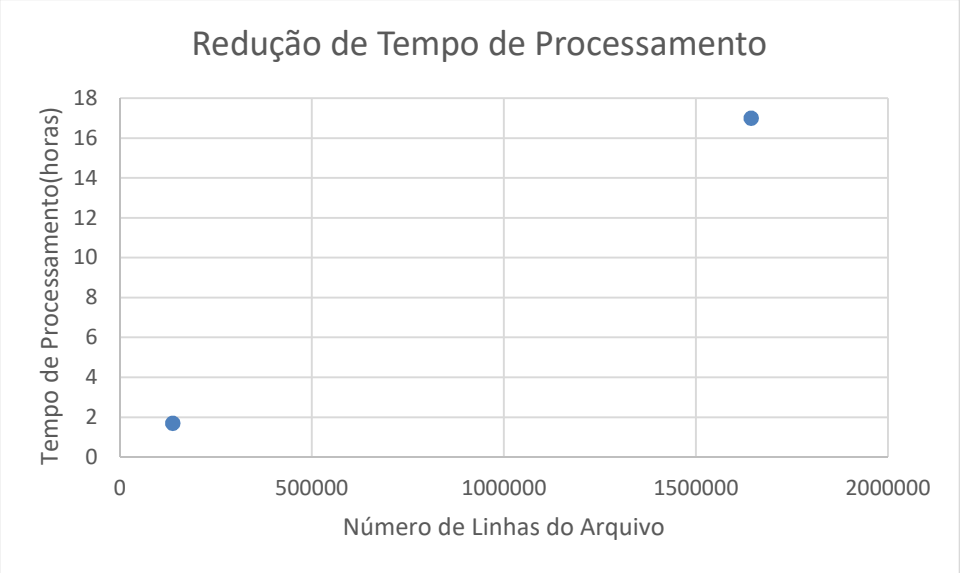


Figura 15: Redução do Tempo de Processamento de Seleção de Features (Do Autor)

O impacto do novo algorítimo de seleção de *Features* na sua importância acumulada do conjunto de Features selecionadas pode se avaliado utilizando o conceito de Grau de Aderência:

$$Grau\ de\ Aderência = \frac{Número\ de\ Features\ Intercessão\ entre\ 2\ Conjuntos\ de\ Seleção}{Número\ de\ Features\ Selecionadas}$$

A Figura 16 ilustra o conceito de grau de aderência. Nesse caso o grau de aderência é calculado pela razão do número de *Features* da Intercessão entre os dois conjuntos de seleção e número de Features selecionadas. Dessa forma, o grau de aderência é de 13/20 ou 65% para 20 Features selecionadas.

Id	Seleção de Features Arquivo único	Seleção de Features Múltiplos Arquivos
1	Bwd Packet Length Std	Bwd Packet Length Std
2	Flow Bytes/s	Subflow Fwd Bytes
3	Total Length of Fwd Packets	Flow Bytes/s
4	Subflow Fwd Bytes	Total Length of Fwd Packets
5	Init_Win_bytes_forward	Average Packet Size
6	Fwd Packet Length Std	Bwd Packets/s
7	Bwd Packets/s	Init_Win_bytes_forward
8	min_seg_size_forward	Fwd Packet Length Std
9	Init_Win_bytes_backward	Init_Win_bytes_backward
10	Fwd IAT Mean	act_data_pkt_fwd
11	Fwd IAT Min	Flow IAT Mean
12	Fwd Header Length	FIN Flag Count
13	Fwd IAT Max	Fwd IAT Mean
14	Flow IAT Std	Flow IAT Max
15	Average Packet Size	Subflow Bwd Packets
16	PSH Flag Count	Total Backward Packets
17	Packet Length Mean	Fwd Header Length
18	Max Packet Length	min_seg_size_forward
19	Flow IAT Min	Fwd Packet Length Max
20	Bwd IAT Mean	PSH Flag Count

Figura 16: Intercessão entre Conjuntos de Seleção de Features (Do Autor)

A Figura 17 ilustra a variação do grau de aderência para um número de *Features* selecionadas no intervalo de 1 a 20 para os dados mostrados na Figura 16. Observa-se que com 1 *Feature* selecionada o grau de aderência é de 100%, pois foi selecionada a mesma *Feature* em ambas as listas de seleção da Figura 16. Para 20 Features selecionadas é de 65% como calculado no parágrafo anterior.

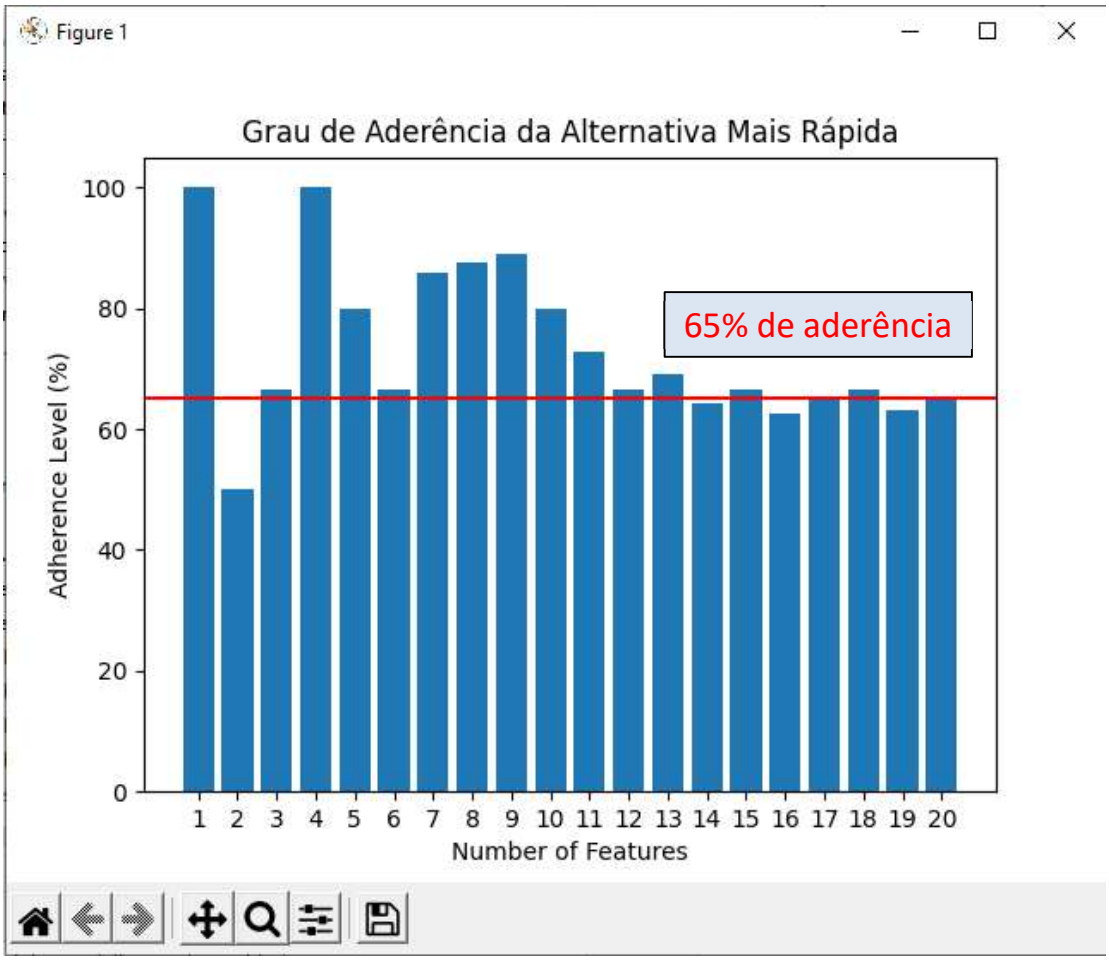


Figura 17: Grau de Aderência para a Alternativa mais rápida (Do Autor)

A Figura 18 e 19, mostram, respectivamente, a Importância acumulada para 20 *Features* selecionadas pelo processamento de um único arquivo de gravação de ataque e selecionadas pelo processamento de arquivos separados de gravação para cada ataque. Nota-se que há uma pequena redução na importância acumulada para 20 Features (de 92,65 % para 87,57%), porém, com ganho no tempo de processamento.

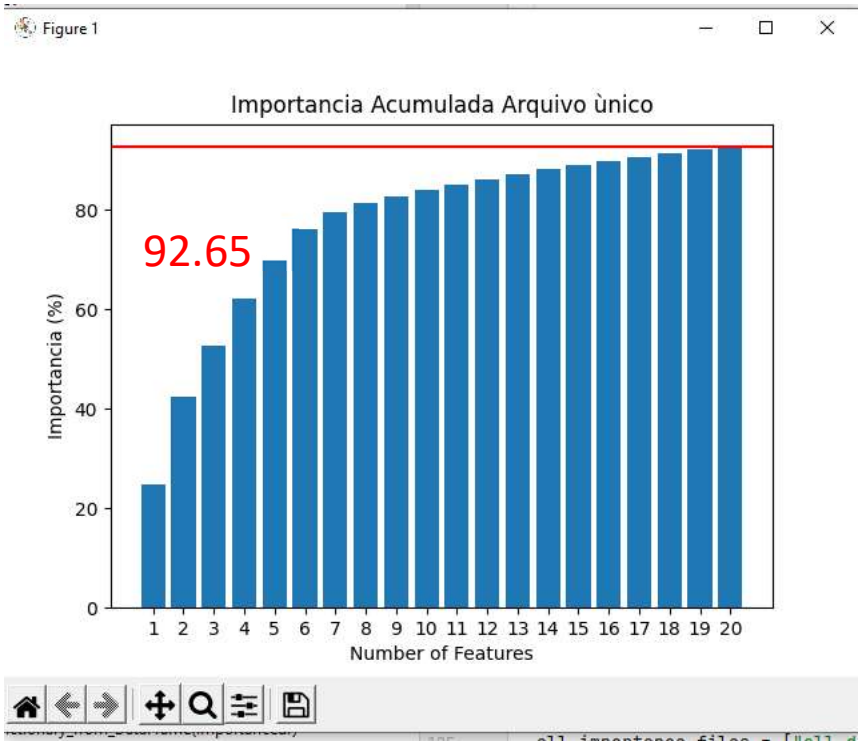


Figura 18: Importância Acumulada -Arquivo Único (Do Autor)

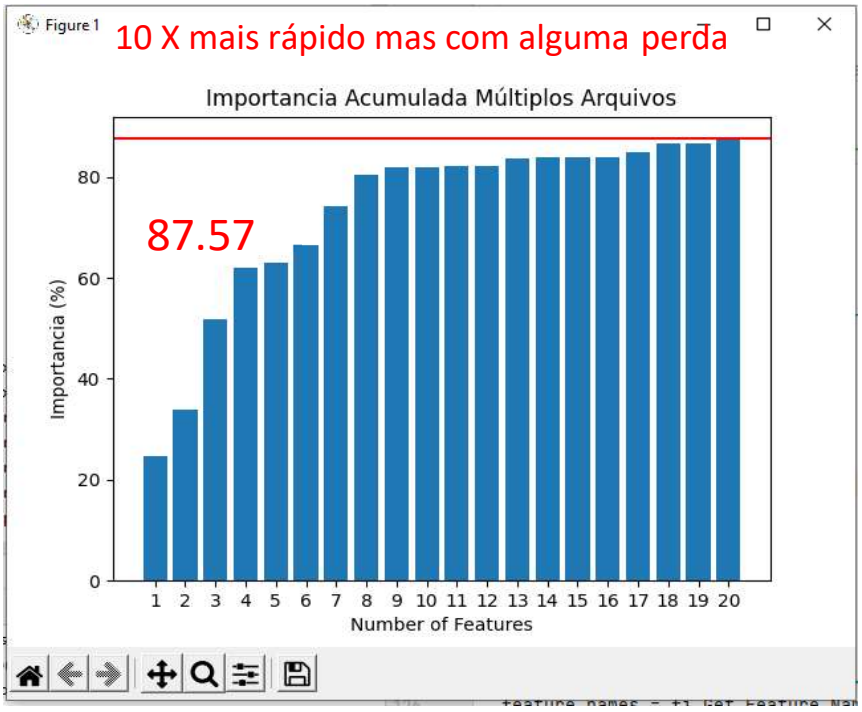


Figura 19: Importância Acumulada -Múltiplos Arquivos (Do Autor)

4.2 Redução Controlada do Arquivo de Gravação de Dados de Ataques

Os arquivos reduzidos foram gerados corretamente no diretório de trabalho/reduzidos da pasta de trabalho utilizada para os programas do TCC.

4.3 Transformação dos arquivos para processamento por classificador *Multi-Label*

Os arquivos transformados foram gerados corretamente no diretório de trabalho/multilabel da pasta de trabalho utilizada para os programas do TCC.

4.4 Implementação em Python de Classificadores *Multi-Label* por Transformação do Problema

As Figura 20 e 21 apresentam os resultados obtidos para o processamento *Multi-Label* por Transformação do Problema de um arquivo de dados gravados de ataques com 150.000 linhas. O classificador K vizinhos mais próximos com transformação do problema pela técnica de cadeia de classificadores atingiu a melhor acurácia (de 99,7422 %) para um número de Features igual a 20, porém com um custo de processamento alto de 927,17 segundos. Nota-se que o classificador K vizinhos mais próximos com transformação do problema pela técnica de *Label PowerSets* atinge uma acurácia de 99.74% com menor custo de processamento de 78,24 segundos.

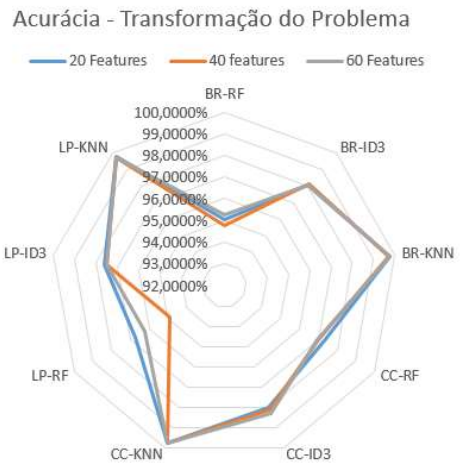
Desse experimento pode-se observar que:

- Aumentar o número de linhas do arquivo melhora a acurácia às custas de maior tempo de processamento, porém essa melhoria vai diminuindo à medida que se aumenta o número de linhas a ser processado. Com a massa de dados utilizada, 150.000 linhas pareceu ser uma opção de bom custo/benefício;
- Aumentar o número de *Features* não necessariamente melhora a acurácia uma vez que pode levar a *overfitting* do modelo. A opção de 20 *Features* pareceu ter um bom custo-benefício;

- O classificador K vizinhos mais próximos resulta em melhor acurácia que os demais classificadores (Árvore de Decisão ID-3 e Floresta Aleatória) porém exige maior tempo de processamento; e
- A maioria dos classificadores testados possuem *Hamming Score* menor que 0,5%.

Consolidação de Resultados para 150.000 Linhas						
Id	Número de Features	Transf do Problema	Model Classif	Acurácia	Hamming Score	Tempo de Execução (segs)
1	20	BinaryRelevance	Random Forest	95,0378%	0,5911%	4,9047
2	20	BinaryRelevance	ID3	98,0222%	0,1815%	5,2133
3	20	BinaryRelevance	K Nearest Neighbors	99,7333%	0,0383%	937,1928
4	20	ClassifierChain	Random Forest	97,2533%	0,2908%	4,3750
5	20	ClassifierChain	ID3	98,0067%	0,2087%	5,0311
6	20	ClassifierChain	K Nearest Neighbors	99,7422%	0,0379%	927,1776
7	20	LabelPowerset	Random Forest	96,7356%	0,4916%	2,9246
8	20	LabelPowerset	ID3	97,6178%	0,3557%	3,4093
9	20	LabelPowerset	K Nearest Neighbors	99,7400%	0,0383%	78,2464
10	40	BinaryRelevance	Random Forest	94,7733%	0,4641%	7,3491
11	40	BinaryRelevance	ID3	98,1022%	0,1735%	11,0446
12	40	BinaryRelevance	K Nearest Neighbors	99,7067%	0,0417%	954,9010
13	40	ClassifierChain	Random Forest	97,0067%	0,3400%	6,9057
14	40	ClassifierChain	ID3	98,1200%	0,1954%	10,2465
15	40	ClassifierChain	K Nearest Neighbors	99,7156%	0,0419%	966,8313
16	40	LabelPowerset	Random Forest	94,8933%	0,7750%	3,6161
17	40	LabelPowerset	ID3	97,4800%	0,3769%	4,4970
18	40	LabelPowerset	K Nearest Neighbors	99,7200%	0,0412%	75,2612
19	60	BinaryRelevance	Random Forest	95,2356%	0,4224%	7,4339
20	60	BinaryRelevance	ID3	98,0333%	0,1735%	15,1614
21	60	BinaryRelevance	K Nearest Neighbors	99,7267%	0,0398%	989,4066
22	60	ClassifierChain	Random Forest	96,9667%	0,3446%	7,1815
23	60	ClassifierChain	ID3	98,3067%	0,1621%	13,7541
24	60	ClassifierChain	K Nearest Neighbors	99,7267%	0,0403%	1026,7343
25	60	LabelPowerset	Random Forest	96,2489%	0,5665%	3,5898
26	60	LabelPowerset	ID3	97,4844%	0,3764%	5,2607
27	60	LabelPowerset	K Nearest Neighbors	99,7267%	0,0403%	78,2367

Figura 20: Resultados Para Tranformação do Problema para arquivo com 150.000 linhas (Do Autor)



BR-RF	Binary Relevance + Random Forest
BR-ID3	Binary Relevance + ID3
BR-KNN	Binary Relevance + K Nearest Neighbors
CC-RF	Chain of Classifiers + Random Forest
CC-ID3	Chain of Classifiers + ID3
CC-KNN	Chain Of Classifiers + K Nearest Neighbors
LP-RF	Label PowerSets + Random Forest
LP-ID3	Label PowerSets + ID3
LP-KNN	Label PowerSets + K Nearest Neighbors

Figura 21: Gráfico Radar para Transformação do Problema 150.000 linhas (Do Autor)

4.5 Implementação em Python do Classificador Multi-label por Adaptação

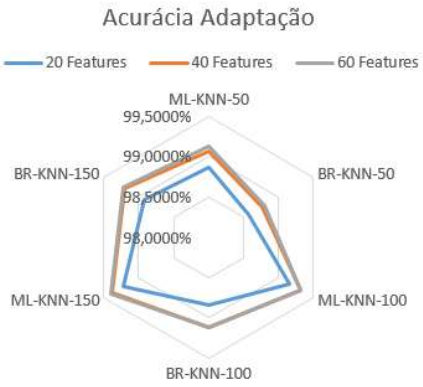
As Figura 22 e 23 apresentam os resultados obtidos para o processamento *Multi-Label* por Adaptação de arquivos de dados gravados de ataques com de 50.000, 100.000 e 150.000 linhas.

O melhor resultado de acurácia igual a 97,40 % foi obtido para adaptação por ML-KNN, 150.000 linhas e 60 features, mas com um tempo de execução muito alto de 8954,28 segundos (mais de duas horas).

O algoritmo BR-KNN de fato reduz o tempo de execução da classificação *Multi-Label* em relação a técnica de Tranformação do Problema que combina Relevância Binária e K Vizinhos mais próximos. Porém, essa redução não chega a 13 (valor do número de *Labels*), além de resultar em uma pequena redução da acurácia.

Id	Número de Features	Número de linhas	Adaptação	Acurácia	Hamming Score	Tempo de Execução (segs)
1	20	50000	ML-KNN	98,8667%	0,1682%	1431,8751
2	20	50000	BR-KNN	98,5667%	0,2056%	62,5369
4	40	50000	ML-KNN	99,0733%	0,1359%	1450,5202
5	40	50000	BR-KNN	98,7600%	0,1774%	68,7013
7	60	50000	ML-KNN	99,1333%	0,1262%	1431,7700
8	60	50000	BR-KNN	98,7933%	0,1738%	68,4275
10	20	100000	ML-KNN	99,1533%	0,1259%	3733,2006
11	20	100000	BR-KNN	98,8400%	0,1721%	229,1552
12	40	100000	ML-KNN	99,3133%	0,1023%	4268,1660
13	40	100000	BR-KNN	99,1200%	0,1305%	237,3990
14	60	100000	ML-KNN	99,3133%	0,1026%	3802,3964
15	60	100000	BR-KNN	99,1100%	0,1318%	247,6996
16	20	150000	ML-KNN	99,2133%	0,1186%	7594,5586
17	20	150000	BR-KNN	98,9267%	0,1590%	540,2305
18	40	150000	ML-KNN	99,3800%	0,0930%	7015,0698
19	40	150000	BR-KNN	99,2089%	0,1164%	566,6058
20	60	150000	ML-KNN	99,4022%	0,0894%	8954,2879
21	60	150000	BR-KNN	99,2267%	0,1135%	522,9266

Figura 22: Resultados Obtidos Por Adaptação (Do Autor)



ML-KNN-50	Multi-Label K Nearest Neighbors 50.000 linhas
BR-KNN-50	Binary Relevance K Nearest Neighbors 50.000 linhas
ML-KNN-100	Multi-Label K Nearest Neighbors 100.000 linhas
BR-KNN-100	Binary Relevance K Nearest Neighbors 100.000 linhas
ML-KNN-150	Multi-Label K Nearest Neighbors 150.000 linhas
BR-KNN-150	Binary Relevance K Nearest Neighbors 150.000 linhas

Figura 23: Gráfico Radar para Adaptação (Do Autor)

5 CONCLUSÕES E TRABALHOS FUTUROS

Esta seção apresenta os seguintes aspectos relacionados ao encerramento do trabalho:

- Conclusões; e
- Trabalhos Futuros.

5.1 Conclusões

A partir dos estudos realizados neste TCC, incluindo-se a experimentação e a avaliação de resultados obtidos, conclui-se o seguinte:

- O processo de Seleção de *Features* por Floresta Aleatória a partir das importâncias obtidas das *Features* para arquivos de ataque individuais com base na ponderação pela frequência de contagem dos ataques ao invés de processar um arquivo único com todos ataques apresenta um ganho em tempo de processamento da ordem de 10 vezes com uma pequena perda em termo de importância acumulada das Features selecionadas;
- A redução controlada do arquivo original de gravação de dados de ataques para arquivos de menores tamanhos, de 50.000 a 250.000 linhas, possibilitou a implementação dos algoritimos de classificação *Multi-Label* com menores tempos de processamento para execução em um *Home-PC* e a observação de seus resultados;
- A memória disponível no *Home-PC* empregado no trabalho (8 Giga bytes de RAM) foi suficiente para realização de todos os testes com os algoritimos de Inteligência Artificial independente do tamanho do arquivo processado. A única restrição para o processamento de arquivos muito grandes (acima de 250.000 linhas) foi o tempo de processamento muito alto em um *Home PC*;
- O processamento de classificação *Multi-Label* exigiu a modificação do arquivo original que continha os *Labels* de ataque ou Benigno em uma

única coluna para um arquivo contendo uma coluna para cada *Label* de ataque ou Benigno;

- A classificação *Multi-Label* por Transformação do Problema foi testada pela combinação da técnica utilizada (Relevância Binária, Cadeia de Classificadores e *Label PowerSet*) com o classificador binário escolhido (Árvore de Decisão ID-3, Floresta Aleatória e K Vizinhos mais próximos). O melhor resultado em termos de Benefício-Custo foi obtido para um arquivo de 150.000 linhas, com 20 Features selecionadas, técnica Label Power Set combinada com o K Vizinhos Mais Próximos: acurácia de 99,74%, *Hamming-Lo*ss de 0,0383% e tempo de processamento de 78,25 segundos;
- A Classificação Muli-Label por Adaptação foi testada com a utilização dos classificadores ML-KNN e BR-KNN, que utilizam como base a classificação binária do tipo K Vizinhos mais próximos. Os resultados de Acurácia obtidos foram um pouco inferiores aos obtidos por Transformação do Problema. No caso do BR-KNN observou-se uma redução do tempo de processamento em relação ao método equivalente de Transformação do Problema, mas inferior ao número de *Labels*;
- A decisão pela escolha do melhor classificador *Multi-Label* depende muito de uma análise de custo-benefício. Sempre é possível aumentar o número de linhas do arquivo a processar para alcançar uma melhor acurária, embora esse ganho seja cada vez menor ao se aumentar o número de linhas a processar, e ainda pode exigir um maior investimento em recursos de processamento e memória. Para o *Dataset* estudado no TCC, típico de classificação *Muti-Class*, a utilização da técnica de Transformação de Resultado por *Label Power Sets* com K Vizinhos mais próximos apresentou melhor custo-benefício.

5.2 Trabalhos Futuros

Considera-se que os seguintes aspectos relacionados ao Tema do TCC poderiam ser abordados em Trabalhos Futuros:

- Desenvolvimento de um Simulador de Ataques a partir de dados gravados em redes reais. A técnica utilizada no TCC para redução controlada do arquivo gravado pode ser expandida para controle da presença de cada ataque de forma individual. A única desvantagem de um Simulador assim é que o padrão das *Features* geradas sempre estaria restrito ao tipo do gravador de dados de redes utilizado na geração dos dados originais;
- Desenvolvimento de um Banco de Dados via *Web* (SQL ou não) para repositório de dados obtidos dos diversos arquivos de gravação de dados reais de ataques em redes disponíveis na Internet. O usuário do banco de dados poderia escolher qual a fonte de gravação, os ataques disponíveis, o tamanho do arquivo e a proporção de cada ataque para geração de um arquivo para processamento, através de consulta a base de Dados. A desvantagem no caso é que os arquivos de gravação de dados são muito grandes e iriam requerer um servidor com boa capacidade. Também pode-se perder muito no pré-processamento dos dados disponíveis na Internet paa filtragem e limpeza;
- Estudo e avaliação das diferentes técnicas de Seleção de *Features* para o processamento de Inteligência Artificial além da técnica de Floresta Aleatória, como por exemplo, utilizando a técnica de Correlação entre as colunas da tabela do *Dataset*;
- Os dados utilizados no TCC são *Multi-Class* devido a restrição da gravação de apenas um ataque por vez no arquivo original. Nesse caso, o processamento por Transformação do Problema já dá um resultado muito bom. Caso se deseje explorar melhor as técnicas de classificação *Multi-Label*, sugere-se um estudo a partir da gravação de dados de outros domínios que tenham características *Multi-Label*, como por exemplo: Classificação de Textos, Classificação de Imagens e Classificação Biológica (genética). Nesse caso caberia verificar a disponibilidade de dados desse tipo de forma aberta na Internet.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] AT&T BUSINESS, 5***G and The Journey to The Edge, 10. ed. Dallas: AT&T Cybersecurity Insights Report***, 2021. Disponível em: <https://cybersecurity.att.com/resource-center/industry-reports/cybersecurity-insights-report-tenth-edition>. Acesso em: 06 abr. 2022.

[2] CANADIAN INSTITUTE FOR CYBERSECURTY, **CIC-IDS2017 – *Intrusion Dataset 2017***, Universidade New Brunswick, Fredericton, Canada, 2017. Disponível em: <https://www.unb.ca/cic/datasets/ids-2017.html>. Acesso em: 06 abr. 2022.

[3] SHARAFALDIN, Iman; LASHKARI, Arash; GHORBANI, Ali. **CIC-IDS2017, *Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization***, *Canadian Institute for Cybersecurty***,** Universidade New Brunswick, Fredericton, Canada, 2018. Disponível em: <https://www.scitepress.org/papers/2018/66398/66398.pdf>. Acesso em: 06 abr. 2022.

[4] THAKKAR, Ankit; LOHIYA, Ritika. ***ICCIDS-2019, A Review of the Advancement in Intrusion Detection DataSets***, Elsevier, Procedia Computer Science, Gujarat, India, 2020. Disponível em:<https://www.sciencedirect.com/science/article/pii/S1877050920307961>.Acesso em: 06 abr. 2022.

[5] KOSTAS, Kahraman. ***Anomaly Detection in Networks Using Machine Learning***, Tese (Mestrado em Segurança e Redes de Computadores), Universidade de Heriot-Watt, Edinburgh, Scotland, ago. 2018.Disponível em:<https://www.researchgate.net/publication/328512658_Anomaly_Detection_in_Networks_Using_Machine_Learning>. Acesso em: 06 abr. 2022.

[6] KOSTAS, kahraman. ***Anomaly Detection in Networks Using Machine Learning***, Tese (Mestrado em Segurança e Redes de Computadores), Universidade de Heriot-Watt, Edinburgh, Scotland, Acervo Git-Hub de Códigos Python, set. 2021. Disponível em: <https://github.com/kahramankostas/Anomaly-Detection-in-Networks-Using-Machine-Learning>. Acesso em: 06 abr. 2022.

[7] BSD, *Berkeley Software Distribution*, **Biblioteca Python para Classificação Binária SkLearn**, 2022. Disponível em: <https://scikit-learn.org/stable/supervised_learning.html#supervised-learning>. Acesso em: 06 abr. 2022.

[8] BSD, *Berkeley Software Distribution*, **Biblioteca Python para Classificação Multi-Label SkMultiLearn**, 2022. Disponível em: <http://scikit.ml>. Acesso em: 06/04/2022.

[9] BHATTACHARYYA, Dhruba; KALISTA, Jugel. ***Network Anomaly Detection – A Machine Learning Perspective***, Boca Raton, Fl., USA: CRC PRESS, 2014.

[10] SECURITY ON-LINE, ***Ares: Python Botnet or Backdoor***, 2018. Disponível em: <https://securityonline.info/ares-python-botnet-backdoor/>. Acesso em: 06 abr. 2022.

[11] WIKIPEDIA***, Low Orbit Ion Cannon,*** Disponível em: <https://en.wikipedia.org/wiki/Low_Orbit_Ion_Cannon>. Acesso em: 06 abr. 2022.

[12] MAZEBOLT, ***Golden Eye HTTP Flood***, 2022. Disponível em: <https://kb.mazebolt.com/knowledgebase/goldeneye-http-flood/>. Acesso em: 07 abr. 2022.

[13] THE DARK SOURCE, ***Hulk DoS Tool – How to use a Web Server DoS Tool***, 2022. Disponível em: <https://thedarksource.com/hulk-dos-tool/>. Acesso em: 07 abr. 2022.

[14] SHEKYAN, Gergey. ***GITHUB, SlowHTTPTest***, 2022. Disponível em: <https://github.com/shekyan/slowhttptest>. Acesso em: 07 abr. 2022.

[15] NETSCOUT, ***SloLoris DDoS Attacks***, 2022. Disponível em: <https://www.netscout.com/what-is-ddos/slowloris-attacks>. Acesso em: 07 abr. 2022

[16] LANJELOT, ***GITHUB,* Patator**, 2012. Disponível em:
< https://github.com/lanjelot/patator >.
Acesso em: 07 abr. 2022

[17] HEARTBLEED, ***The HeartBleed Bug***, 2022. Disponível em:
<https://heartbleed.com/>. Acesso em: 07 abr. 2022

[18] METASPLOIT, ***The World´s Most Used Penetration Testing Framework***, 2022. Disponível em: <https://www.metasploit.com/>. Acesso em: 08 abr. 2022

[19] CISCO PRESS, ***Penetration Testing and Network Defense: Performing Host Reconnaissance***, 2022. Disponível em:
<https://www.ciscopress.com/articles/article.asp?p=469623&seqNum=4/>.Acesso em: 08 abr. 2022

[20] TECHBEACON, ***How to scale Your Web Application Security Testing with Selenium***, 2022. Disponível em: <https://techbeacon.com/security/how-scale-your-web-app-security-testing-selenium/>. Acesso em: 08 abr. 2022

[21] CANADIAN INSTITUTE FOR CYBERSECURTY, ***CICFlowMeter***, 2022. Disponível em: <https://www.unb.ca/cic/research/applications.html>. Acesso em: 08 abr. 2022

[22] BSD, *Berkeley Software Distribution*, ***Metrics and Scoring: Quantifying the Quality of Predictions***, 2022. Disponível em: <https://scikit-learn.org/stable/modules/model_evaluation.html>. Acesso em: 11 abr. 2022

[23] SHARAT, Shandra. ***Hamming Score for Multi-Label Classification***, 2022. Disponível em: <https://www.linkedin.com/pulse/hamming-score-multi-label-classification-chandra-sharat/>. Acesso em: 12 abr. 2022

[24] KOHAVI, R.; John***,*** G***. Wrappers for Feature Subset Selection. Artificial Intelligence***, 1997. Disponível em: < https://ai.stanford.edu/~ronnyk/wrappersPrint.pdf>. Acesso em: 12 abr. 2022

[25] HALL, M. A.; SMITH, L. A; ***Feature Subset Selection: A Correlation Baased Iterative Approach. In Proc. of the International Conference on Neural Information Processing and Intelligent Information Systems***, Springer, 1997. Disponível em: < https://researchcommons.waikato.ac.nz/handle/10289/1515>. Acesso em: 12 abr. 2022

[26] MESQUITA, Ricardo. **Árvores de Decisão - Material de Apoio**, Unicarioca, 2021.

[27] BREIMAN, Leo; ***Random Forests***, 2001. Disponível em: < https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf>. Acesso em: 12 abr. 2022

[28] BSD, *Berkeley Software Distribution*, ***Class Sklearn ensemble Random Forest Regressor***, 2022. Disponível em: <https://scikitlearn.org/0.16/modules/generated/sklearn.ensemble.RandomForestRegressor.html>. Acesso em: 12 abr. 2022

[29] XU, Frank. ***Towards Data Science, Which one is more important? Be careful before you made decisions with RandomForest***, 2020. Disponível em: <https://towardsdatascience.com/a-relook-on-random-forest-and-feature-importance-2467dfab5cca> Acesso em: 12 abr. 2022

[30] ALGORÍTMOS E MACHINE LEARNING, **K Vizinhos Mais Próximos, Algoritmos de Classificação**, 2022.Disponível em: <https://aimlsite.wordpress.com/2018/01/05/k-vizinhos-mais-proximos-algoritmo-de-classificacao/>.Acesso em: 22 abr. 2022

[31] MAIMOM, Oded; ROKACH, Lior. ***Data Mining and Knowledge Discovey Handbook***, 2. Ed., New York: Springer, 2010.

[32] ANALYTICS VIDHYA, ***Solving Multi-Label Classification Problems***, 2017. Disponível em: <https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/>.Acesso em: 22 abr. 2022

[33] ZHANG, Min-Ling; ZHOU, Zhi-Hua. ***ML-KNN: A Lazy Learning Approach to Multi-Label Learning***, 2007. Disponível em: <https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/pr07.pdf>. Acesso em: 23 abr. 2022

[34] SPIROMITROS, E.; TSOUMAKAS, G.; VLAHAVAS, I***. An Empirical Study of Lazy Multi_label Classification Algorithms***, 2008. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-540-87881-0_40>. Acesso em: 23 abr. 2022

[35] KARIUKI, Charles. ***Multi-Label Classification with Scikiti-MultiLearn***, 2021. Disponível em: <https://www.section.io/engineering-education/multi-label-classification-with-scikit-multilearn/> Acesso em: 29 abr. 2022

[36] TEN, David. ***Multi-Label Classification with Scikiti-MultiLearn***, GITHUB, 2002. Disponível em: <https://xang1234.github.io/multi-label/> Acesso em: 29 abr. 2022

[37] WIKIPEDIA, ***Glossary of Artificial Intelligence***, 2022. Disponível em: <https://en.wikipedia.org/wiki/Glossary_of_artificial_intelligence> Acesso em: 30/04/2022

[38] BSD*, Berkeley Software Distribution*; ***Emsemble Methods, Parameters***, 2022. Disponível em: <https://scikit-learn.org/stable/modules/ensemble.html#parameters> Acesso em: 30 abr. 2022

[39] HAYKIN, Simon**. Redes Neurais: Princípios e Prática**, 2. Ed. Porto Alegre: Bookman, 2001.

[40] WIKIPEDIA, ***Random Forest***, 2022.

Disponível em: <https://en.wikipedia.org/wiki/Random_forest>.

Acessso em: 12 mai. 2022

GLOSSÁRIO

Termo	Definição
Acurácia	Métrica de desempenho da classificação Multi-Label que indica o percentual de acertos
Adaptacão	A técnica de classificação *Multi-Label* por Adaptação busca adaptar os algoritmos de classificação binária para a classificação Multi-Label pela utilização das seguintes alternativas de classificadores: ML-KNN e BR-KNN.
Aprendizado	É um processo pelo qual os parâmetros livres de um classificador são adaptados através de um processo de estimulação pelo ambiente no qual o classificador está inserido (Adaptado de [39])
Aprendizado Não Supervisionado	Tipo de aprendizado que não requer a rotulação a priori de cada linha do *Dataset*.
Aprendizado Supervisionado	Tipo de aprendizado que exige a rotulação a priori de cada linha do Dataset a ser utilizado para treinamento. Esses rótulos indicam classes que normalmente são obtidas por opinião de especialista.
Árvore de Decisão	Árvore que é construída para auxílio a decisão com base na priorização das *Features* em ordem de importância a partir daTeoria da Informação. Essas árvores podem ser utilizadas para Regressão ou classificação. E podem ser de contrução aleatória (ex: *Random Forests*) ou não (ex: ID3)
Binary Relevance K Nearest Neighbors(BR-KNN)	É uma técnica de adaptação para classificação *Multi-Labe*l que otimiza a busca dos K vizinhos mais próximos em combinação com Relevância Binária.
Cadeia de Classificadores	Técnica de Transformação do Problema para classificação Multi-Label em que as tabelas construidas para cada Label dependem dos Labels anteriores

Termo	Definição
Classificação	A classificação se refere a um problema de modelagem preditiva onde um rótulo (*Label*) de classe é prevista para um determinado exemplo de dados de entrada. (Adaptado de [37])
Classificação Binária	Refere-se à previsão de uma de duas classes, que podem ser codificadas de forma binária (0 ou 1). Exemplo: Ataque na rede ou Normalidade (Benigno). (Adaptado de [37])
Classificação Multi-Class	Envolve a previsão de zero ou mais rótulos (Labels) de classe, nas quais os rótulos de classe são mutuamente exclusivos. (Adaptado de [37])
Classificação *Multi-Label*	Envolve a previsão de zero ou mais rótulos (Labels) de classe, nas quais os rótulos de classe não são mutuamente exclusivos. (Adaptado de [37])
Dataset	Um conjunto de dados correspondente ao conteúdo de uma única tabela de banco de dados, ou de uma única matriz de dados estatísticos no qual cada coluna da tabela representa uma determinada variável, e cada linha corresponde a um determinado membro do conjunto de dados em questão. O conjunto de dados lista valores para cada uma das variáveis, tais como altura e peso de um objeto, para cada membro do conjunto de dados. Cada valor é conhecido como um dado. O conjunto de dados pode incluir dados para um ou mais membros, correspondendo ao número de linhas.(Adaptado de [37]). A ref [4] apresenta uma revisão de *Datasets* aplicáveis a detecção de ataques em redes de computadores.
Desbalanceamento de Classes	Problema que ocorre quando o *Dataset* contem diferentes frequências de contagem para suas classes. Quando isso acontece as predições das

Termo	Definição
	classes de menor frequência de contagem apresentam menor acurácia.
Features	Cararacterísticas ou propriedades disponíveis nas linhas de um *Dataset*
First Come First Served	Estratégia de fila que foi utilizada na coleta de amostras dos ataques para redução de lnhas do arquivo para processamento *Multi-Label*
Hamming Loss	Métrica de desempenho da classificação Multi-Label que indica o percentual de erros. Utiliza a Teoria dos Códigos de Hamming no cálculo na média das distâncias entre o código da classe prevista e o código da classe atual (distâncias de Hamming)
Índice de Gini	Índice de dispersão estatística proposto em 1912 pelo estatístico italiano Corrado Gini. È utilizado na obtenção das importâncias das *Features* por um Regressor de Floresta Aleatória.
Inteligência Artificial	Tecnologia que possibilita que uma máquina emule funções "cognitivas" as quais os humanos associam a outras mentes humanas, tais como "aprendizagem" e "resolução de problemas.
Intrusion Detection System	Sistema automatizado de detecção de intrusão em Redes de Computadores
K Vizinhos Mais Próximos	O classificador binário do tipo K Vizinhos mais próximos procura os K vizinhos mais próximos em distância da amostra a ser classificada aos elementos da Matriz de Treinamento (linhas de *Features*).
Label PowerSets	Técnica de Transformação do Problema para classificação *Multi-Label* que considera novas classes pela combinação entre as ocorrências dos Labels.

Termo	Definição
Labels	Rótulos, classes associadas às Features de um Dataset, são necessários a priori em caso de Aprendizado Supervisionado de Inteligência Artificial
Matriz de Confusão	A Matriz de Confusão para um problema de classificação *Multi-Label* apresenta a contagem de acertos e erros da classe prevista pelo classificador. A contagem de acertos é registrada na diagonal da Matriz enquanto a contagem de erros fica registrada das partes triangulares superior e inferior da Matriz.
Métricas de desempenho	As métricas de desempenho são utilizadas na avaliação dos resultados dos classificadores. Para o classificador *Multi-Label* podem ser utilizadas as seguintes: Matriz de Confusão, Acurácia e Hamming Loss.
Multi-Label K Nearest Neighbors (ML-KNN)	É uma técnica de adaptação para classificação *Multi-Labe*l que combina K Vizinhos Mais Próximos com a Teoria de Probabilidade Condicional de Bayes para o cálculo da probabilidade da intercessão da ocorrência dos *Labels*.
Network Slicing	Tecnologia empregada nas redes 5 G para dividir a rede em camadas horizontais virtuais que alocam recursos de forma dinâmica conforme os requisitos do cliente
Python	Linguagem de Programação muito utilizada para implementação e teste de algoritmos de Inteligência Artificial, sendo apoiada por diversas bibliotecas de *software* livre
Random Forest (Floresta Aleatória)	Um método de aprendizagem em que um conjunto de árvores de decisão é construído de forma aleatória (amostragem com reposição) para classificação e regressão através de treinamento e predição de saída da (classificação) ou da média (regressão) .

Termo	Definição
Regressão	É uma técnica para investigar a relação entre variáveis ou características independentes (*Features*) e uma variável ou resultado dependente. É usada como um método de modelagem preditiva na aprendizagem de máquinas, no qual um algoritmo é usado para prever resultados contínuos.
Relevância Binária	Técnica de Transformação do Problema para classificação *Multi-Label* em que são construídas tabelas independentes para a classificação binária de cada *Label*
Rótulo de Ataque	Rótulo ou *Label* de uma linha do *Dataset* que indica o tipo de ataque realizado na rede: *Bot, DDoS, DDoS GoldenEye, DoS Hulk, DoS slowhttptest, Dos slowloris, FTP-Patator, Heartbleed, Infiltration, PortScan, SSH Patator e Web Attack*
Rótulo de Benigno	Label ou rótulo de uma linha do Dataset em que não há realização de ataque na rede. O rótulo de Benigno indica normalidade.
Software Defined Networks	È uma abordagem de gerenciamento de redes que possibilita a configuração dinâmica da redes de computadores possibltando melhoria de desempenho , monitoração e agilidade para atender novos requisitos do cliente.
Test Bed	Ambiente de Teste composto com computadores e softwares interligados através de redes. Nesse ambiente são realizados ataques de Segurança de Informação de forma controlada e dados são gravados na rede para geração de um *Dataset*
Transformação do Problema	É uma das técnicas de abordagem da classificação *Multi-Label*. Essa técnica procura transformar o problema de classificação Multi-Label em um conjunto de problemas de classificação binária, através de um

Termo	Definição
	dos métodos: Relevância Binária, Cadeia de Classificadores e *Label PowerSets*.

APÊNDICE A – DESCRIÇÃO DOS ATAQUES E FERRAMENTAS

A Tabela A.1 apresenta a descrição dos ataques realizados para geração do conjunto de dados CIC-IDS2017 [3] incluindo as ferramentas utilizadas para sus realização:

Ataque	**Descrição do Ataque**	**Ferramenta**	**Descrição da Ferramenta**
Botnet (*Bot*)	Ataques de *SPAM* em e-mails ou *DDoS* [5].	Ares[10]	Linguagem Python, Administrador e agentes *malware*
DDoS	Negação de serviço distribuído	*LOIC* (*Low Orbit ION Cannon*)[11]	linguagem C#
DoS GoldenEye	Negação de serviço por inundação HTTP	*Golden Eye*[12]	Requisição *HTTP GET*
DoS Hulk	Negação de serviço	*Hulk* [13]	Linguagem Python Sobrecarrega servidores *Web*
DoS Slowhttptest	Negação de serviço	*Slowhttptest*[14]	Linguagem C++ Torna o tempo de conexão *HTTP* lento
DDoS SlowLoris	Negação de serviço distribuído	*SlowLoris*[15]	Torna o tempo de conexão *HTTP* lento
FTP-Patator	Ataque de Força Bruta	*Patator*[16]	Linguagem Python Atua no *login FTP*
Heartbleed	Roubo de informação protegida	*Heartbleed*[17]	Explora um *Bug* da biblioteca *SSL*
Infiltration	Ataque de coleta de informação	*Metasploit* [18]	*Download* ou cópia de arquivo infectado com virus Virus executa varredura e coleta informação.
PortScan	Ataque de coleta de informação	*NMap switches*[19]	Executa varredura e coleta informação.
SSH-Patator	Ataque de Força Bruta	*Patator*[16]	Linguagem Python Atua no login SSH

Ataque	Descrição do Ataque	Ferramenta	Descrição da Ferramenta
Web Attack	Força Bruta, *XSS(Cross-site scripting) e Sql Injection*	*Selenium Framework*[20]	IDE de scripts de testes automatizados

Tabela A.1: Descrição dos Ataques e Ferramentas (CCIDS-2017)

Obs: Algumas das ferramentas utilizadas são livres e estão disponíveis para download na Internet.

APÊNDICE B – LISTA DE IMPORTÂNCIAS DE FEATURES RESULTANTE DO PROCESSAMENTO DE UM ÚNICO ARQUIVO COM TODOS OS ATAQUES

Features, importance

Bwd Packet Length Std,0.2466260237992073

Flow Bytes/s,0.17871978576198877

Total Length of Fwd Packets,0.1023694474508433

Subflow Fwd Bytes,0.09314768253163362

Init_Win_bytes_forward,0.07631861465579869

Fwd Packet Length Std,0.06389937818054914

Bwd Packets/s,0.035002648028040755

min_seg_size_forward,0.017390338338523673

Init_Win_bytes_backward,0.01375737612726867

Fwd IAT Mean,0.012776308641861562

Fwd IAT Min,0.01108386235381743

Fwd Header Length,0.010111752347370448

Fwd IAT Max,0.009989605757854188

Flow IAT Std,0.009879408186687293

Average Packet Size,0.009218603826683902

PSH Flag Count,0.008824950024108373

Packet Length Mean,0.007956017215916376

Max Packet Length,0.007074450739701624

Flow IAT Min,0.0069464115559786386

Bwd IAT Mean,0.005386852721131183

Idle Min,0.005181836679975601

Fwd IAT Total,0.005136234356302765

Fwd IAT Std,0.004420714432607411

Flow Duration,0.0041500750892723565

Bwd Packet Length Max,0.004013511459186213

Flow IAT Max,0.0035337132303905126

Idle Mean,0.0034577534442026747

Fwd Packets/s,0.0033767542439156343

Bwd IAT Std,0.003367833257482618

Flow IAT Mean,0.0032748910052164647

Bwd IAT Min,0.0025443033537097337

Idle Max,0.0024956833807651428

Active Std,0.0024560689698030344

Bwd Header Length,0.002346657322800124

Bwd IAT Max,0.002110144204800857

Active Max,0.002005917588312588

Active Mean,0.0016846061934557973

Active Min,0.0016822106196160112

Bwd IAT Total,0.0015394487402014582

Fwd PSH Flags,0.0014638093162482207

Total Length of Bwd Packets,0.0013251239727437365

Subflow Bwd Bytes,0.0013128886963928612

SYN Flag Count,0.0012867214859894102

Packet Length Std,0.0011774596744599328

Packet Length Variance,0.000996009298742627

URG Flag Count,0.0007193633565487237

Fwd Packet Length Min,0.0006861795807750826

ACK Flag Count,0.0006682985758411072

Down/Up Ratio,0.0006092330228881779

Avg Bwd Segment Size,0.0006038213838464541

Min Packet Length,0.000571014447770622

Flow Packets/s,0.000541171732133121

Fwd Packet Length Mean,0.0005367934458602009

Bwd Packet Length Mean,0.0005259864448691013

Avg Fwd Segment Size,0.000487406988140044

Subflow Bwd Packets,0.0001956497404200605

Total Backward Packets,0.0001773958326720122

FIN Flag Count,0.00016058736041763813

Idle Std,0.0001493998631267193

Fwd Packet Length Max,0.00013812510027173873

Total Fwd Packets,0.0001271617420293934

Subflow Fwd Packets,0.00011929315480566611

act_data_pkt_fwd,8.712335669894894e-05

Bwd Packet Length Min,7.57567824291602e-05
CWE Flag Count,1.4620010263812037e-07
Fwd URG Flags,8.78721009984257e-08
RST Flag Count,0.0
Bwd Avg Bulk Rate,0.0
Bwd Avg Packets/Bulk,0.0
Bwd Avg Bytes/Bulk,0.0
Fwd Avg Bulk Rate,0.0
Fwd Avg Packets/Bulk,0.0
Bwd URG Flags,0.0
Fwd Avg Bytes/Bulk,0.0
Bwd PSH Flags,0.0
ECE Flag Count,0.0

Tabela B.1: Lista de Importâncias de *Features* pelo Processamento de Arquivo único de Ataques

APÊNDICE C – CÓDIGO PYTHON PARA OBTENÇÃO DE LISTA ALTERNATIVA DE IMPORTÂNCIAS PELO PROCESSAMENTO DAS LISTAS DE IMPORTÂNCIA OBTIDAS PARA CADA ATAQUE

```
# Calculo Alternativo de Importância das Features para Detecção de
Anomalias/Ataques em Redes de Computadores
# Nome: FeaturesImportance.py
# Nome do Aluno: Vitor de Avila Falcão
# Matricula: 2019201206
# Curso: Ciência da Computação - Unicarioca - Rio Comprido
# Disciplina: TCC 2022.1
# pré-condições:
# Os arquivos com dados gravados para cada ataque deverão estar no
diretório ./attacks
# Os arquivos com as listas de importância processadas para cada ataque
deverão estar no diretótio ./importances
# Data: 13/04/2022
# Orientador: Prof Fabio Henrique Silva

import os
import pandas as pd
import time
import operator
import warnings

warnings.filterwarnings("ignore")

# Gerar um dicionário de ataques:
# key -  o tipo de ataque e o
# valor - tuples (Contagem de ataques, Contagem de linhas do arquivo)
# entrada:
# attacks - lista de ataques
def Generate_Dictionary_Attacks(attacks):
  attacks_dict = {}
  for atq in attacks:
     # obter o nome do arquivo de ataque
     atq_file = atq + ".csv"
     sample_file = r'.\\attacks\\'+atq_file
     df = pd.read_csv(sample_file, dtype={'Label': 'str'})
     total_count = len(df)
     atq_count = total_count-df["Label"].value_counts().BENIGN
     attacks_dict[atq] = (atq_count, total_count)
  del df
  return attacks_dict

# Calcular uma lista de pesos para cada ataque a partir do dicionário de
ataques
# entrada:
# attacks_dict - dicionário de ataques obtido por
Generate_Dictionary_Attacks
def Generate_Weights_For_Attacks(attacks_dict):
  lcontrib = [attacks_dict[key][0] for key in attacks_dict]
  total = sum(lcontrib)
  lweights = [x/total for x in lcontrib]
  return lweights
```

```
# Gerar um dicionário de importâncias
# sendo a key o tipo de ataque e
# o valor a importância do ataque
# Entradas:
# attacks - lista de ataques
# features - lista de features
# lweights - lista de pesos para cada ataque
def Generate_Dictionary_Importances(attacks, features, lweights):
  print("\n Generate_Dictionary_Importances")
  features_importance_dict = {x: 0.0 for x in features}
  count_attacks = 0
  for atq in attacks:
    atq_imp_file = atq + ".csvimportance.csv"
    sample_file = r'.\\importances\\' + atq_imp_file
    df = pd.read_csv(sample_file)
    for feat in features:
      lindice = df[df["Features"] == str(feat)].index.values
      if len(lindice): # if not empty list
        indice = lindice[0]
        valor = df.iloc[indice][1]
        features_importance_dict[feat] = features_importance_dict[feat]\
        +valor*lweights[count_attacks]
    count_attacks += 1
    del df
  features_importance_dict =dict(sorted(features_importance_dict.items()\
  ,key=operator.itemgetter(1), reverse=True))
  return features_importance_dict


# Salvar lista de importâncias alternativa para arquivo
# Entrada:
# features_importance_dict - dicionário de imporâncias das features
obtido por Generate_Dictionary_Importances
def save_features_importance_to_file(features_importance_dict):
  df = pd.DataFrame(list(features_importance_dict.items()))
  df.to_csv("alt all data.csvimportance.csv")
  del df


# Obter lista de ataques a partir dos nomes dos arquivos de ataque
# Entrada:
# attack_files - lista de arquivos e ataque
def Get_List_of_Attacks(attack_files):
  lencsv = len(".csv")
  attacks = [x[0:len(x) - lencsv] for x in attack_files]
  return attacks


# Obter a lista com o nome das Features
# Essa lista está gravado na primeira linha de todos os arquivos de
ataque
def Get_Feature_Names():
  shortestattackfile = r'.\\attacks\\Heartbleed.csv';
  df = pd.read_csv(shortestattackfile)
  # Eliminar a coluna 'Label' de df
  del df["Label"]
  feature_names = list(df.columns.values)
  del df
  return feature_names
```

```
if __name__ == "__main__":
  # Obter o tempo inicial
  seconds = time.time()

  # Obter a lista de nomes de arquivos de ataque
  # Cria lista de arquivos de ataque  no diretório ./attacks
  attack_files = os.listdir("attacks")

  #Obter a lista de ataques
  attacks = Get_List_of_Attacks(attack_files)

  # Obter a lista de arquivos de importância para cada ataque
  # Cria uma lista com os nomes dos arquivos no diretorio ./importances.
  importance_files = os.listdir("importances")

  # Extrai a lista de nomes das Features a partir do menor arquivo
  # de  ataque
  feature_names = Get_Feature_Names()

  # Gera dicionário de ataques contendo tuplas
  # (contagem do ataque, contagem total do arquivo de ataque)
  # O dicionário será utilizado para ponderar importâncias
  attacks_dict = Generate_Dictionary_Attacks(attacks)
  print("dicionario de ataques")
  print(attacks_dict)

  # Gera a lista de pesos para todos os ataques
  lweights = Generate_Weights_For_Attacks(attacks_dict)
  print("lweights")
  print(lweights)

  # Constrói o dicionário de importâncias a partir da média ponderada
  # das importâncias obtidas pelo processamento de cada arquivo
  # em separado
  features importance dict = Generate Dictionary Importances(attacks,\
  feature_names, lweights)

  # Salvar a lista de importâncias calculada por média ponderada
  #para um arquivo
  save_features_importance_to_file(features_importance_dict)

  print("mission accomplished!")
  print("Total operation time: = ", time.time() - seconds, "seconds")
```

Tabela C.1: Código FeaturesImportance.py

APÊNDICE D – LISTA DE IMPORTÂNCIAS DE FEATURES RESULTANTE DO PROCESSAMENTO DE ARQUIVOS SEPARADOS PARA CADA ATAQUE

,0,1
0,Bwd Packet Length Std,0.29502432959378844
1,Subflow Fwd Bytes,0.12272265359166866
2,Flow Bytes/s,0.10644050655366508
3,Total Length of Fwd Packets,0.10298131828285055
4,Average Packet Size,0.08476867449650351
5,Bwd Packets/s,0.05197910448345264
6,Init_Win_bytes_forward,0.041841939130030564
7,Fwd Packet Length Std,0.03949778640532554
8,Init_Win_bytes_backward,0.020134062150337584
9,act_data_pkt_fwd,0.018207061307077468
10,Flow IAT Mean,0.014178389181230855
11,FIN Flag Count,0.011853087253448278
12,Fwd IAT Mean,0.011511979892378328
13,Flow IAT Max,0.010631624294404103
14,Subflow Bwd Packets,0.009460763292907374
15,Total Backward Packets,0.009273738075281542
16,Fwd Header Length,0.007396653789359562
17,min_seg_size_forward,0.006015490723600586
18,Fwd Packet Length Max,0.0036398833944879297
19,PSH Flag Count,0.0027114998218546998
20,Flow IAT Min,0.0025987607163231704
21,Fwd IAT Min,0.002204668932029439
22,Avg Bwd Segment Size,0.0020267474970239144
23,Bwd Packet Length Mean,0.0020046781509666933
24,Flow Duration,0.0016989007726183373
25,Subflow Bwd Bytes,0.001365611967276298
26,Bwd Header Length,0.001182445076777603
27,Fwd Packets/s,0.0011712425401818555
28,Avg Fwd Segment Size,0.0011120064267538827
29,Total Length of Bwd Packets,0.0010712786106659834
30,Fwd IAT Total,0.0010591677854171265
31,Fwd Packet Length Mean,0.0009647577850978214
32,Bwd IAT Min,0.0009048198219702767
33,Packet Length Mean,0.000893237781713905

34,Fwd IAT Max,0.0007925105898548722
35,Fwd IAT Std,0.0007356155364778568
36,Idle Min,0.0007057531233712535
37,Flow IAT Std,0.0005752360965954876
38,Bwd IAT Mean,0.000561276172788522
39,URG Flag Count,0.0005273550936355398
40,SYN Flag Count,0.00045016647863208855
41,Bwd IAT Total,0.0004276078106546844
42,Down/Up Ratio,0.0004237022260301514
43,Fwd PSH Flags,0.0003694537097488112
44,Bwd IAT Max,0.00034294450651903974
45,Idle Max,0.00031106328480531007
46,Subflow Fwd Packets,0.00030659626985706943
47,Total Fwd Packets,0.0003025296391781437
48,Idle Mean,0.00029606865344620413
49,Fwd Packet Length Min,0.0002701699594415839
50,Max Packet Length,0.00025166216282994456
51,Active Min,0.0002319290878322813
52,ACK Flag Count,0.0002280052049120891
53,Bwd IAT Std,0.00022639931645404318
54,Active Mean,0.00022428676000591736
55,Active Max,0.00022366754246700227
56,Flow Packets/s,0.0001795499267573351
57,Packet Length Std,0.00013553711877341583
58,Bwd Packet Length Max,0.00012714466989702009
59,Packet Length Variance,0.0001248077636242747
60,Bwd Packet Length Min,7.78041656403224e-05
61,Idle Std,2.102612531765161e-05
62,Min Packet Length,1.7106928978329428e-05
63,Active Std,2.6838170225302945e-06
64,CWE Flag Count,6.66626709213606e-07
65,Fwd URG Flags,4.870119636170004e-07
66,RST Flag Count,1.8809567756685373e-07
67,ECE Flag Count,1.2894562998284523e-07
68,Flow ID,0.0
69,Source IP,0.0
70,Source Port,0.0
71,Destination IP,0.0
72,Destination Port,0.0

73,Protocol,0.0
74,Timestamp,0.0
75,Bwd PSH Flags,0.0
76,Bwd URG Flags,0.0
77,Fwd Avg Bytes/Bulk,0.0
78,Fwd Avg Packets/Bulk,0.0
79,Fwd Avg Bulk Rate,0.0
80,Bwd Avg Bytes/Bulk,0.0
81,Bwd Avg Packets/Bulk,0.0
82,Bwd Avg Bulk Rate,0.0
83,External IP,0.0

Tabela D.1: Lista de Importâncias de Features pelo Processamento de Arquivos Separados de Ataques

APÊNDICE E – CÓDIGO DE ANÁLISE DAS LISTAS DE IMPORTÂNCIAS

```
# Análise da Importância das Features para Detecção de
# Anomalias/Ataques em Redes de Computadores
# Nome: FeatureImportanceAnalysis.py
# Nome do Aluno: Vitor de Avila Falcão
# Matricula: 2019201206
# Curso: Ciência da Computação - Unicarioca - Rio Comprido
# Disciplina: TCC 2022.1
# Data: 13/04/2022
# Orientador: Prof Fabio Henrique Silva
# pré-condições:
# Os arquivos com as listas de importâncias devem estar no diretório ./
# arquivo all_data.csvimportance.csv - Lista de importâncias para
# o processamento de um único arquivo com todos os ataques
# arquivo alt_all_data.csvimortance.csv - Lista de importâncias para
# o processamento de arquivos separados para cada ataque

import pandas as pd
import time
import FeaturesImportance as fi
import matplotlib.pyplot as plt

# Obter lista de DataFrames para cada uma das Listas de Importâncias
# em um total de 2 listas de importância
# Entrada:
# Lista de Arquivos de Importância
def Get_Importance_DataFrames(all_importance_files):
  df = [None, None]
  for ind in range(len(all_importance_files)):
    sample_file = r'.\\' + all_importance_files[ind]
    df[ind] = pd.read_csv(sample_file)
    if (ind == 1):
      del df[1]['Unnamed: 0']
    print(df[ind].head())

  return df

# Gerar dicionário de importâncias a partir do Data Frame de Importâncias
# Entrada:
# Data-Frame de importâncias
def Generate_Importance_Dictionary_from_DataFrame(importancedf):
  importance_dict = {importancedf.iloc[ind][0]:\
  importancedf.iloc[ind][1] for ind in importancedf.index}
  return importance_dict

# Obter percentual de coincidência entre as duas listas de importância
# Entradas:
# lf - Lista de importâncias para arquivo único com todos os ataques
# altlf - lista de importâncias para arquivos separados para cada ataque
def GetPercentOfFeaturesMatches(lf, altlf):
  lenlf = len(lf)
  lenaltlf = len(altlf)
  assert(lenlf==lenaltlf), "Lista Must Have The Same Size"
  count = 0
  for x in lf:
    for y in altlf:
```

```
      if (x == y):
        count = count+1
  return (count/ lenlf)*100.0

# Obter lista de aderencia variando com o número de features selecionadas
# Entradas:
# lf - Lista de importâncias para arquivo único com todos os ataques
# altlf - lista de importâncias para arquivos separados para cada ataque
# numfeatures - número de features a considerar
def Compute_Adherence(lfeatures, altlfeatures, numfeatures):
  lenfeatures = len(lfeatures)
  lenaltfeatures = len(altlfeatures)
  lenmin = min(lenfeatures, lenaltfeatures)
  assert(numfeatures <= lenmin), 'Excessive Numer of features'
  percadher = []
  for ind in range(numfeatures):
    lf = lfeatures[0:ind+1]
    altlf = altlfeatures[0:ind+1]
    perc = GetPercentOfFeaturesMatches(lf, altlf)
    percadher.append(perc)
  return percadher

# Obter Importancia acumuladda por numero de features utilizado
# Calculo para solução com um unico arquivo de ataques dicimp[0]
# e com varios arquivos de ataque separados dictimp[1]
#Entradas:
# dictimp - Lista de dicionário de importância
# numfeatures - número de features a considerar
def GetAccImportance(dictimp, numfeatures):
  lfeatures, limportances =\
Generate_Separated_Lists_from_Importancedictionary\
  (dictimp[0])
  altlfeatures, altlimportances = \
Generate_Separated_Lists_from_Importancedictionary \
  (dictimp[1])
  lenfeatures = len(lfeatures)
  lenaltfeatures = len(altlfeatures)
  lenmin = min(lenfeatures, lenaltfeatures)
  assert (numfeatures <= lenmin), 'Excessive Number of features'

  # calculate acc list of importance for all attackks file solution
  lacc = []
  acc = 0.0
  for x in limportances[0:numfeatures]:
    acc = acc + x
    lacc.append(acc*100.0)

  laltacc = []
  altcacc = 0.0
  for ind in range(numfeatures):
    y = dictimp[0][altlfeatures[ind]]
    altcacc = altcacc + y
    laltacc.append(altcacc*100.0)

  # calculate acc list of importance for separated files attacks solution
  return lacc, laltacc


# Plotar Gráfico de Barras
# Entrada:
```

```
# lista - lista de valores a plotar
def Plotar_Barras(lista, title = "Grafico de Barras", xlabel = "x",
ylabel = "y", hline = 0):
  inds = range(len(lista))
  vals = lista
  labels = [str(ind + 1) for ind in inds]
  fig, ax = plt.subplots()
  rects = ax.bar(inds, vals)
  ax.set_xticks([int(ind) for ind in inds])
  ax.set_xticklabels(labels)
  ax.set_xlabel(xlabel)
  ax.set_ylabel(ylabel)
  ax.set_title(title)
  if (hline != 0):
    ax.axhline(y=hline, color='r')
  plt.show()


f __name__ == "__main__":
  # Get initial Time
  seconds = time.time()

  all_importance_files = ["all_data.csvimportance.csv",\
  "alt_all_data.csvimportance.csv"]
  feature_names = fi.Get_Feature_Names()
  df = Get_Importance_DataFrames(all_importance_files)

  dictimp = [Generate_Importance_Dictionary_from_DataFrame(x)\
  for x in df]

  lfeatures, limportances =
Generate_Separated_Lists_from_Importancedictionary\
  (dictimp[0])
  print(lfeatures)
  print(limportances)

  altlfeatures, altlimportances =
Generate_Separated_Lists_from_Importancedictionary\
  (dictimp[1])
  print(altlfeatures)
  print(altlimportances)

  numfeatures = 20
  percadher = Compute_Adherence(lfeatures, altlfeatures, numfeatures)
  print(percadher)

  # Plotar aderencia de soluções
  Plotar_Barras(percadher, title="Grau de Aderência da \
  Alternativa Mais Rápida", xlabel="Number of Features", \
  ylabel="Adherence Level(%)", hline=65)

  lacc, laltacc = GetAccImportance(dictimp, numfeatures)
  print(lacc)
  print(len(lacc))
  Plotar_Barras(lacc, title="Importancia Acumulada\
  Arquivo ùnico", xlabel="Number of Features",\
  ylabel="Importancia (%)", hline=lacc[-1])

  print(laltacc)
```

```
print(len(laltacc))
Plotar_Barras(laltacc, title="Importancia Acumulada\
Múltiplos Arquivos", xlabel="Number of Features",\
ylabel="Importancia (%)", hline=laltacc[-1])

print("mission accomplished!")
print("Total operation time: = ", time.time() - seconds, "seconds")
```

Tabela E.1: Código FeatureImportanceAnalysis.py

APÊNDICE F – CÓDIGO PYTHON PARA OBTENÇÃO ARQUIVO REDUZIDO DE GRAVAÇÂO DE DADOS DE ATAQUES

```
# Redução controlada do arquivo de gravação de dados de ataques
# para um número linhas especificado
# Nome: FeatureImportanceAnalysis.py
# Nome do Aluno: Vitor de Avila Falcão
# Matricula: 2019201206
# Curso: Ciência da Computação - Unicarioca - Rio Comprido
# Disciplina: TCC 2022.1
# Data: 17/04/2022
# Orientador: Prof Fabio Henrique Silva
# pré-condições:
# Os arquivos com os dados de cada ataque deverão estar no diretório
,/attacks/
# Saída:
# o arquivo reduzido será escrito no diretório ./reduzidos/

import os
import time
import FeaturesImportance as fi
import numpy as np


# Função utilizada para gerar o novo diretório ./reduzidos/
def folder(f_name):
  try:
    if not os.path.exists(f_name):
      os.makedirs(f_name)
  except OSError:
    print("The folder could not be created!")

# Calcula lista de pesos para cada ataque a partir dos arquivos do
dicionário ./attacks/
# Entrada:
# attacks_dict: dicionário de ataques
# key -  o tipo de ataque e o
# valor - tuples (Contagem de ataques, Contagem de linhas do arquivo)
# Saida:
# Lista de pesos de cada ataque
def Generate_Weights_For_Attacks_Files(attacks_dict):
  lcontrib = [attacks_dict[key][1] for key in attacks_dict]
  total = sum(lcontrib)
  lweights = [x/total for x in lcontrib]
  return lweights

# Gerar dicionário de ataques com contagem reduzida para
# o número de linhas estipulado utilizando
# redução proporcional ao número de linas de ataque
# nos arquivos de gravação de ataques originais
# Preserva a contagem nínima doa ataques HeartBleed
# e Infiltration
# Para tal é feita uma compensação em DoS Hulk
# Entradas:
# attacks: lista de ataques
# numlinhas: Meta de redução do número de linhas
# lweights: lista de pesos de cada ataque
# Saída:
```

```
# dicionário de ataques
# key -  o tipo de ataque e o
# valor - tuples (Contagem de ataques, Contagem de linhas do arquivo)
def Generate_Dictionary_Reduced_Attacks(attacks, numlinhas, lweights):
  lweightsarray = np.asarray(lweights)
  # Criterio de seleção de Linhas - FCFS - First Come First Served
  linhasporarquivoataque = numlinhasreduz * lweightsarray
  attacks_red_dict = {}
  Ind = 0
  for atq in attacks:
    attacks_red_dict[atq] = (round(0.30 *\
    (linhasporarquivoataque[Ind])),\
    round(linhasporarquivoataque[Ind]))
    Ind = Ind + 1

  # ajustes para manter os minimos de HeartBleed e Infiltration
  # sem redução
  # Tirar dos maiores, i.e. DoS Hulk e PortScan
  compens2 = 41 - attacks_red_dict["Heartbleed"][1]
  compens1 = 119 - attacks_red_dict["Infiltration"][1]
  attacks_red_dict["Heartbleed"] = (11, 41)
  attacks_red_dict["Infiltration"] = (36, 119)
  compensado1 = attacks_red_dict["DoS Hulk"][1]-compens1
  attacks_red_dict["DoS Hulk"] = (round(0.3*(compensado1)), compensado1)
  compensado2 = attacks_red_dict["PortScan"][1] - compens2
  attacks_red_dict["PortScan"] = (round(0.3 *\
  (compensado2)), compensado2)

  #ajuste final pela soma total, altera o valot total de Dos Hulk
  soma = ObterSomaAttackDict(attacks_red_dict)
  delta = numlinhas-soma
  numataques = attacks_red_dict["DoS Hulk"][0]
  numtotal = attacks_red_dict["DoS Hulk"][1]
  attacks_red_dict["DoS Hulk"] = (numataques, numtotal+delta)

  return attacks_red_dict

# Obtem a contagem total de linhas do dicionário de ataques
# Entrada:
# attacks_dict: dicionário de ataques
# key -  o tipo de ataque e o
# valor - tuples (Contagem de ataques, Contagem de linhas do arquivo)
# Saída:
# Contagem do total de linhas do dicionário de ataque
def ObterSomaAttackDict(attacks_dict):
  lcontrib = [attacks_dict[key][1] for key in attacks_dict]
  soma = sum(lcontrib)
  return soma


# Gerar arquivo único com todos os ataques e contagem reduzida
# Entrada:
# attacks_dict: dicionário de ataques
# key -  o tipo de ataque e o
# valor - tuples (Contagem de ataques, Contagem de linhas do arquivo)
# attacks_red_dict: dicionário de ataques
# key -  o tipo de ataque e o
# valor - tuples (Contagem de ataques, Contagem de linhas do arquivo)
# Saída:
# Arquivo único com todos os ataques e contagem reduzida
```

```
# salvo no diretório ./reduzidos/.
def GeneratedReducedFile(attack_files, attacks_red_dict, numlinhasreduz):
  # Loop para  os arquivos de ataques
  strlinhas = str(numlinhasreduz)[0:-3]
  outfile = open(".//reduzidos//reducedallattackfiles"+strlinhas+\
  ".csv", "w")
  print("reducedallattackfiles"+strlinhas+".csv"+" is opened")
  conttotallin = 0
  FirstFile = True
  # attack_files = ["Bot.csv"]
  for attackfile in attack_files:
    infile = open(".//attacks//" + attackfile, "r")
    atq = attackfile[0:-4]
    # obter limites de ataques e de benignos
    limatq = attacks_red_dict[atq][0]
    limbenigno = attacks_red_dict[atq][1] - limatq
    countlin = 0
    countatq = 0
    countbenign = 0

    while (True):
      line = infile.readline()

      if (countlin == 0):
        if (FirstFile):
          outfile.write(line)
          FirstFile = False
      else:
        listlinestr = line.split(',')
        #if (conttotallin == 0):
        #print(listlinestr)
        # if atq in listlinestr:
        if atq in listlinestr or \
        ((atq == "Web Attack")
        and (("Web Attack - Brute Force" in listlinestr)
        or ("Web Attack - XSS" in listlinestr)
        or ("Web Attack - Sql Injection" in listlinestr))
        ):
          # escrever linhas de ataques se o contador não suplantou
          # o limite
          if (countatq < limatq):
            outfile.write(line)
            countatq = countatq + 1
            conttotallin = conttotallin+1
        else:
          # escrever linha de benignos se contador não suplantou o limite
          if (countbenign < limbenigno):
            outfile.write(line)
            countbenign = countbenign + 1
            conttotallin = conttotallin+1

      countlin = countlin + 1
      if not line:
        break

    infile.close()
  outfile.close()
  print("reducedallattackfiles"+strlinhas+".csv"+" is closed")
  print("Numero total de Linhas:")
  print(conttotallin)
```

```
if __name__ == "__main__":
  # Get initial Time
  seconds = time.time()

  # linhas de 50.000 até 250.000 passo de 50.000
  linhas = list(range(50000, 300000, 50000))
  print("linhas")
  print(linhas)

  # Obter a lista de arquivos de ataque a partir do diretório ./attacks
  attack_files = os.listdir("attacks")

  folder("./reduzidos/")
  # Obter a lista de ataques
  # removendo .csv de cada arquivo de ataque da lista
  attacks = fi.Get_List_of_Attacks(attack_files)

  # Extrair a lista de nomes das Features do menor arquivo de ataque
  feature_names = fi.Get_Feature_Names()

  # Gerar dicionário de ataques com as tuplas (contagem de cada ataque,
contagem total do arquivo de ataque)
  attacks_dict = fi.Generate_Dictionary_Attacks(attacks)

  # Gerar pesos a partir do dicionario de ataques
  lweights = Generate_Weights_For_Attacks_Files(attacks_dict)

  for numlinhasreduz in linhas:
    attacks_red_dict = Generate_Dictionary_Reduced_Attacks\

    (attacks,numlinhasreduz, lweights)

    soma = ObterSomaAttackDict(attacks_red_dict)
    print("Confirmando número de linhas do arquivo a ser gerado:")
    print(soma)

    # Gerar arquivo de ataques reduzido
    GeneratedReducedFile(attack_files, attacks_red_dict, numlinhasreduz)

  print("mission accomplished!")
  print("Total operation time: = ", time.time() - seconds, "seconds")
```

Tabela F.1: Código ReduzAllAttacksFiles.py

APÊNDICE G – CÓDIGO PYTHON PARA TRANFORMAÇÃO DO ARQUIVO REDUZIDO PARA PROCESSAMENTO MULTI-LABEL

```
# Produzir arquivos reduzidos com labels em binário para uso por
classificadores Multi_label
# Nome: ReducedFilesToMultiLabel.py
# Nome do Aluno: Vitor de Avila Falcão
# Matricula: 2019201206
# Curco: Ciência da Computação - Unicarioca - Rio Comprido
# Disciplina: TCC 2022.1
# Data: 18/04/2022
# Orientador: Prof Fabio Henrique Silva
# pré-condições:
# Os arquivos com os dados de cada ataque devem estar no diretório
,/attacks/
# Os arquivos reduzidos devem esta no diretório ./reduzidos/
# Saída:
# o arquivo reduzido será escrito no diretório ./multilabels/

import os
import time
import pandas as pd
import FeaturesImportance as fi

# Função utilizada para gerar o novo diretório ./multilabels/
def folder(f_name):
  try:
    if not os.path.exists(f_name):
      os.makedirs(f_name)
  except OSError:
    print("The folder could not be created!")

if __name__ == "__main__":
  # Get initial Time
  seconds = time.time()

  # Cria lista de nomes de arquivos de ataque a partir do
  # diretório ./attacks
  attack_files = os.listdir("attacks")

  # Cria lista de nomes de arquivos de ataque reduzidos do diretório
./reduzidos
  reduced_files = os.listdir("reduzidos")

  # Obter a lista de ataques
  # removendo .csv para cada arquivo de ataque da lista
  attacks = fi.Get_List_of_Attacks(attack_files)
  print(attacks)

  # Cria o diretório ./multilabels
  folder("./multilabels/")

  # Inicia a tranformação para cada arquivo reduzido
  for arquivo in reduced_files:

    # Criar um dataframe para o arquivo reduzido
    df = pd.read_csv(".//reduzidos//"+arquivo, low_memory=False)
    print(df.head())
```

```
    # Acertar colunas dos ataques
    for atq in attacks:
      if atq != "Web Attack":
        df.loc[df['Label'] == atq, atq] = 1
        df.loc[df['Label'] != atq, atq] = 0
      else:
        df.loc[df['Label'] == "Web Attack - Brute Force" , atq] = 1
        df.loc[df['Label'] == "Web Attack - XSS", atq] = 1
        df.loc[df['Label'] == "Web Attack - Sql Injection", atq] = 1
        df.loc[df['Label'] != "Web Attack - Brute Force", atq] = 0
        df.loc[df['Label'] != "Web Attack - XSS", atq] = 0
        df.loc[df['Label'] != "Web Attack - Sql Injection", atq] = 0

    # Acertar a coluna Benign
    df.loc[df['Label'] == "BENIGN", "Benign"] = 1
    df.loc[df['Label'] != "BENIGN", "Benign"] = 0


    print(df.head())

    # Salvar arquivo transformado para multilabel
    df.to_csv(".//multilabels//"+arquivo[0:-4]+".ml.csv",\
    encoding='utf-8')


print("mission accomplished!")
print("Total operation time: = ", time.time() - seconds, "seconds")
```

Tabela G.1: Código ReducedFilesToMultiLabel.py

APÊNDICE H – CÓDIGO PYTHON PARA IMPLEMENTAÇÂO E TESTES DE CLASSIFICADORES MULTI-LABEL POR TRANSFORMAÇÃO DO PROBLEMA

```
# Processamento dee Inteligência Artificial Multi-Label para detecção
# de Ataques
# através da técnica de transformação do problema
# Nome: ProblemTransformation.py
# Nome do Aluno: Vitor de Avila Falcão
# Matricula: 2019201206
# Curco: Ciência da Computação - Unicarioca - Rio Comprido
# Disciplina: TCC 2022.1
# Data: 13/04/2022
# Orientador: Prof Fabio Henrique Silva
# pré-condições:
# Os arquivos com ataques separados devem estar no diretório ./attacks
# O arquivo com as lista de importâncias devem estar no diretório ./
# arquivo all_data.csvimportance.csv - Lista de importâncias para o
processamento de um único arquivo com todos os ataques
# Os arquivos de ataques reduzidos preparados para o processamento Multi-
Label devem ser colocardos no Diretório ./multilabels
# ex: reducedallattackfiles100.ml.csv


import os
import time
import pandas as pd
import FeaturesImportance as fi
import json

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,hamming_loss
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

# Import Transformation Problem Alternatives
from skmultilearn.problem_transform import BinaryRelevance
from skmultilearn.problem_transform import ClassifierChain
from skmultilearn.problem_transform import LabelPowerset

# Testar o modelo de classificação com um método de transformação
# do problema de classificação multi-label para classificação binária
# Entradas:
# model - classificador binário a ser utilizado
# transf_problem - método de tranformação do problema:
# BinaryRelevance, ClassifierChain ou Label Powerset
# Matrizes de entrada e saída para treinamento (xtrain, ytrain)
# Matrizes de entrada e saída parra teste (xtest, ytest)
# Saída:
# result - Dicionário de resultados com acurácia, hamming loss e tempo de
execução
def TestarModelo(model_value, transf_problem_value, xtrain, ytrain,
xtest, ytest):
  model = model_value[1]
  transf_problem = transf_problem_value[1]
```

```
  inicio = time.time()
  clf = transf_problem(model)
  # normalizar os dados
  scaler = MinMaxScaler()
  xtrain = scaler.fit_transform(xtrain)
  xtest = scaler.transform(xtest)
  clf.fit(xtrain, ytrain)
  clf_predictions = clf.predict(xtest)
  acc = accuracy_score(ytest, clf_predictions)
  ham = hamming_loss(ytest, clf_predictions)
  result = {"transf problema": transf_problem_value[0],\
  "modelo": model_value[0],\
  "acuracia:": acc, "hamming_score": ham, \
  "tempo execucao(segs)": time.time()-inicio}
  print(result)
  with open('resultados_transf_problema.txt', 'a') as file:
      file.write(json.dumps(result)) # use `json.loads` to do the reverse
      file.write("\n")
  return result


# Obter Matrizes de Treino (X train, y train)  e Teste (X test, y test)
# Entradas:
# NumeroDeFeatures - utiizado como filtro de colunas das amostras
# attacks - lista de ataques
# df imp - data frame das features em ordem de importância
# df_ataques - data frame de ataques
# Saidas:
# Matrizes de treino - X_train, y_train
# Matrizes de Teste - X_test, y_test
def ObterMatrizesTreinoeTeste(NumerodeFeatures, attacks, df_imp,
df_ataques):
    # Obter lista das Features em ordem de importância
    ImpFeatures = df_imp['Features'].to_list()[0:NumerodeFeatures]
    # print(ImpFeatures)

    # Selecionar Xfeatures pela ordem de importância
    Xfeatures = df_ataques[ImpFeatures]
    # print(Xfeatures)

    # Obter as colunas correspondentes aos ataques
    y = df_ataques[attacks]
    # print("y.head()")
    # print(y.head())

    # Obter a coluna correspondente a Benigno
    y1 = df_ataques['Benign']
    # print("y1.head()")
    # print(y1.head())

    # Concatenar as colunas de ataques e de Benigno
    y = y.join(y1)
    # print("y.head()")
    # print(y.head())

    # Separar test sets de treinamemto e teste (70% para treino e
    # 30% para teste)
    X_train, X_test, y_train, y_test = train_test_split(Xfeatures, \
    y, test_size=0.3, random_state=42)
```

```
    print("type(X_train")
    print(type(X_train))

    print("X_train.shape")
    print(X_train.shape)

    print("y_train.shape")
    print(y_train.shape)

    print("X_test.shape")
    print(X_test.shape)

    print("y_test.shape")
    print(y_test.shape)

    return X_train, y_train, X_test, y_test

# Código retirado de
# https://www.geeksforgeeks.org/python-program-right-rotate-list-n/
# Faz a rotação circular de uma lista a direita
def rightRotate(lists, num):
    output_list = []

    # Will add values from n to the new list
    for item in range(len(lists) - num, len(lists)):
        output_list.append(lists[item])

    # Will add the values before
    # n to the end of new list
    for item in range(0, len(lists) - num):
        output_list.append(lists[item])

    return output_list

if   name   == "  main  ":
    # Obter o tempo inicial
    seconds = time.time()

    # dicionário de classificadores de IA
    dict_classif_bin = {"Nearest Neighbors":\
    KNeighborsClassifier(3),\
    "Random Forest":  RandomForestClassifier(max_depth=5,\
    n_estimators=10, n_jobs=-1),\
    "ID3": DecisionTreeClassifier(max_depth=5, criterion="entropy")}

    # dicionário de métodos de transformação do problema
    dict_prod_transf = {"BinaryRelevance": \
    BinaryRelevance, "ClassifierChain": \
    ClassifierChain, "LabelPowerset": LabelPowerset}

    # Lista de Numero de Features a serem consideradas
    ListaNumeroDeFeatures = list(range(20, 80, 20))
    print(ListaNumeroDeFeatures)

    # Obter lista de nomes de arquivos de ataque
    attack_files = os.listdir("attacks")

    # Obter lista de arquivos reduzidos de ataque prontos
    # para processamento Multi-Label
    reduced_attack_files = os.listdir("multilabels")
```

```
    # Ajustar ordem dos arquivos de 50.000 linhas para 250.000 linhas
    reduced_attack_files = rightRotate(reduced_attack_files, 1)

    # Obter a lista de ataques
    # removendo .csv para cada arquivo da lista de arquivos de ataque
    attacks = fi.Get_List_of_Attacks(attack_files)
    # print(attacks)

    # Criar um Data Frame para as Features da  lista de
    # importancias dos ataques
    df1 = pd.read_csv(".//all_data.csvimportance.csv", low_memory=False)
    # print(df1.head())

    # Se necessário apagar o arquivo de resultados
    # remanescente da execução anterior
    if (os.path.isfile('./resultados_transf_problema.txt')):
        os.remove('./resultados_transf_problema.txt')

    # Aqui começa o loop para testar os modelos para
    # todas as combinações de:
    # NumeroDeFeatures, Número de linhas do Arquivo de Ataque,
    # Método de Transformação do Problema e Classificador
    for NumeroDeFeatures in  ListaNumeroDeFeatures:
      for arquivo_ataque in reduced_attack_files:
        print("\nProcessando "+arquivo_ataque+"\n")
        if arquivo_ataque == "reducedallattackfiles50.ml.csv":
          Str_arquivo_ataque = arquivo_ataque[-9:-7]
        else:
          Str_arquivo_ataque = arquivo_ataque[-10:-7]
        # Criar um data frame parao arquivo de ataque reduzido
        #preparado para processamento Multi_label
        df2 = pd.read_csv(".//multilabels//"+\
        arquivo_ataque, low_memory=False)

        # Obter Matrizes de Treinamento (X_train, y_train)
        # e Teste (X_test, y_test)
        X_train, y_train, X_test, y_test =\
        ObterMatrizesTreinoeTeste(NumeroDeFeatures,
        attacks, df1, df2)

        with open('resultados_transf_problema.txt', 'a') as file:
          # Escrever cabeçalho no arquivo de saída
          StrCabecalho = "Numero de Features = "+\
          str(NumeroDeFeatures)+" Numero de Linhas = "+\
          Str_arquivo_ataque+".000\n"
          file.write(StrCabecalho)

        # Testar Modelo combinando técnicas de tranformação de
        # problema e classificador de IA
        for value_pt in dict_prod_transf.items():
          print("\n")
          for value_clf in dict_classif_bin.items():
            # Evita executar Nearest Neighbors para alto
            # número de linhas do arquivo
            # devido ao tempo de alto de processamento
            if (value_clf[0] != "Nearest Neighbors") or\
            ((value_clf[0] == "Nearest Neighbors")\
            and (arquivo_ataque != 'reducedallattackfiles250.ml.csv')\
            and (arquivo_ataque !=
```

```
            'reducedallattackfiles200.ml.csv'))     or\
            ((value_clf[0] == "Nearest Neighbors") and\
            (arquivo_ataque == \
            'reducedallattackfiles200.ml.csv')
            and (NumeroDeFeatures == 20)):
              TestarModelo(value_clf, value_pt, X_train, \
              y_train, X_test, y_test)

    tempo = time.time() - seconds
    Strtempo = str(tempo)
    StrFecho = "Total operation time: = "+ Strtempo+" seconds"
    with open('resultados_transf_problema.txt', 'a') as file:
        # Escrever fecho do arquivo de saída
        file.write(StrFecho)

    print("mission accomplished!")
    print(StrFecho)
```

Tabela H.1: ProblemTransform.py

APÊNDICE I – CÓDIGO PYTHON PARA IMPLEMENTAÇÂO E TESTES DE CLASSIFICADORES MULTI-LABEL POR ADAPTAÇÂO

```
# Processamento de Inteligencia Artificial Multi-Label para detecção de
Ataques
# utilizando a técnica de adaptação
# Nome: MultiLabel.py
# Nome do Aluno: Vitor de Avila Falcão
# Matricula: 2019201206
# Curco: Ciência da Computação - Unicarioca - Rio Comprido
# Disciplina: TCC 2022.1
# Data: 23/04/2022
# Orientador: Prof Fabio Henrique Silva
# pré-condições:
# Os arquivos com ataques separados devem estar no diretório ./attacks
# O arquivo com as lista de importâncias devem estar no diretório ./
# arquivo all_data.csvimportance.csv - Lista de importâncias para o
processamento de um único arquivo com todos os ataques
# Os arquivos de ataques reduzidos preparados para o processamento Multi-
Label devem ser colocardos no Diretório ./multilabels
# ex: reducedallattackfiles100.ml.csv


import os
import time
import pandas as pd
import FeaturesImportance as fi
import ProblemTransformation as pt
import json


from sklearn.metrics import accuracy_score,hamming_loss
from sklearn.model_selection import GridSearchCV
from skmultilearn.adapt import MLkNN
from skmultilearn.adapt import BRkNNaClassifier



def ConverterDftoArrays(X_train, y_train, X_test, y_test):
  Xarr_train = X_train.to_numpy()
  print(type(Xarr_train))
  print(Xarr_train.shape)
  Xarr_test = X_test.to_numpy()
  print(type(Xarr_test))
  print(Xarr_test.shape)
  yarr_train = y_train.to_numpy()
  print(type(yarr_train))
  print(yarr_train.shape)
  yarr_test = y_test.to_numpy()
  print(type(yarr_test))
  print(yarr_test.shape)

  return Xarr_train, yarr_train, Xarr_test, yarr_test
```

```
# Entradas:
# model_Value - classificador  a ser utilizado
# parametros - parametros do classificador
# score - estratégia para cáculo do score de treinamento
# Matrizes de entrada e saída para treinamento (Xarr_train, yarr_train)
# Matrizes de entrada e saída parra teste (Xarr_test, yarr_test)
# Saída:
# result - Dicionário de resultados com acurácia, hamming loss e tempo de
execução
def TestarModeloAdaptado(modelo_value, parametros, score, Xarr_train,
yarr_train, Xarr_test, yarr_test):
  modelo = modelo_value[1]
  start = time.time()
  clf = GridSearchCV(modelo, parametros, scoring=score)
  clf.fit(Xarr_train, yarr_train)
  time_training = time.time() - start
  print('training time taken: ', time_training, 'seconds')
  print('best parameters :', clf.best_params_,\
 'best score: ', clf.best_score_)
  training_result = {'best parameters' : clf.best_params_,\
  'best score':  clf.best_score_, \
  'training time taken' :  time_training}
  clf_predictions = clf.predict(Xarr_test)
  acc = accuracy_score(yarr_test, clf_predictions)
  ham = hamming_loss(yarr_test, clf_predictions)
  result = {"modelo": modelo_value[0], \
            "acuracia:": acc, "hamming_score": \
            ham, "tempo execucao(segs)": time.time() - start}
  print(result)
  with open('resultados_adaptados.txt', 'a') as file:
      file.write("\n")
      file.write(json.dumps(training_result))
      file.write(json.dumps(result))
      file.write("\n")
  return result



if __name__ == "__main__":

    # Obter o tempo inicial
    seconds = time.time()

    # dicionário de classificadores de IA Muiti-Label adaptados
    dict_classif_ml_adapted = {"ML-KNN": MLkNN(),\
    "BR-KNNa": BRkNNaClassifier()}

    dict_classif_ml_parametros = {"ML-KNN": \

    {'k': range(1, 3), 's': [0.5, 0.7, 1.0]},\
    "BR-KNNa": {'k': range(3, 5)}}

    # Lista de Numero de Features a serem consideradas
    ListaNumeroDeFeatures = list(range(20, 80, 20))
    print(ListaNumeroDeFeatures)

    # Obter lista de nomes de arquivos de ataque
    attack_files = os.listdir("attacks")

    # Obter lista de arquivos reduzidos de ataque prontos para
```

```
    # processamento Multi-Label
    reduced_attack_files = os.listdir("multilabels")

    # Ajustar ordem dos arquivos de 50.000 linhas para 250.000 linhas
    red_attack_files = pt.rightRotate(reduced_attack_files, 1)

    # Pegar até 150.000 linhas
    reduced_attack_files = red_attack_files[0:3]


    # Obter a lista de ataques
    # removendo .csv para cada arquivo da lista de arquivos de ataque
    attacks = fi.Get_List_of_Attacks(attack_files)


    # Criar um Data Frame para as Features da  lista de
    # importancias dos ataques
    df1 = pd.read_csv(".//all_data.csvimportance.csv", low_memory=False)


    # Se necessário apagar o arquivo de resultados
    # remanescente da execução anterior
    if (os.path.isfile('./resultados_adaptados.txt')):
      os.remove('./resultados_adaptados.txt')

    # Aqui começa o loop para testar os modelos para todas
    # as combinações de:
    # NumeroDeFeatures, Número de linhas do Arquivo de Ataque,
    # e Classificador por Adaptação
    for NumeroDeFeatures in ListaNumeroDeFeatures:
      for arquivo_ataque in reduced_attack_files:
        print("\nProcessando " + arquivo_ataque + "\n")
        if arquivo_ataque == "reducedallattackfiles50.ml.csv":
            Str_arquivo_ataque = arquivo_ataque[-9:-7]
        else:
            Str_arquivo_ataque = arquivo_ataque[-10:-7]
        # Criar um data frame parao arquivo de ataque
        # reduzido preparado para processamento Multi_label
        df2 = pd.read_csv(".//multilabels//" + \
        arquivo_ataque, low_memory=False)

        # Obter Matrizes de Treinamento (X_train, y_train)
        # e Teste (X test, y test)
        X_train, y_train, X_test, y_test = \
        pt.ObterMatrizesTreinoeTeste(NumeroDeFeatures,
        attacks, df1, df2)

        Xarr_train, yarr_train, Xarr_test, yarr_test = \
        ConverterDftoArrays(X_train, y_train, \
        X_test, y_test)

        with open('resultados_adaptados.txt', 'a') as file:
            # Escrever cabeçalho no arquivo de saída
            StrCabecalho = "Numero de Features = " + str(\
            NumeroDeFeatures) + " Numero de Linhas = "+\

           Str_arquivo_ataque + ".000\n"
           file.write(StrCabecalho)

        for value_ml in dict_classif_ml_adapted.items():
```

```
            print("\n")
            #score = 'f1_micro'
            score = 'f1_weighted'
            parametros =  dict_classif_ml_parametros[value_ml[0]]
            result = TestarModeloAdaptado(value_ml,\
            parametros, score, Xarr_train, yarr_train,\
            Xarr_test, yarr_test)

tempo = time.time() - seconds
Strtempo = str(tempo)
StrFecho = "Total operation time: = " + Strtempo + " seconds"
with open('resultados_adaptados.txt', 'a') as file:
  # Escrever fecho do arquivo de saída
  file.write(StrFecho)

print("mission accomplished!")
print(StrFecho)
```

Tabela I.1: Multilabel.py

Obs IMPORTANTE: Foi utilizada a versão 0.2.0 da biblioteca scikit-multilearn do Python. Essa versão apresenta um *bug* que deverá ser corrigido nas próximas versões. Por enquanto, deve-se incluir a seguinte correção de forma manual na linha 165 do arquivo mlknn.py e na linha 39 do arquivo brknn.py:

- Substituir:
- self.knn_ = NearestNeighbors(self.k).fit(X)
- Por:
- self.knn_ = NearestNeighbors(n_neighbors=self.k).fit(X)