Seleção de Características Multiobjetivo para Detecção de Malwares Android

Philipe Fransozi[1], Jhonatan Geremias[1], Eduardo K. Viegas[1], Altair O. Santin[1]

[1]Programa de Pós-graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná (PUCPR)
Caixa Postal 17.315 – 80.242-980 – Curitiba – PR – Brasil

{philipe.hfransozi, jgeremias, eduardo.viegas, santin}@ppgia.pucpr.br

Abstract. *This paper proposes a malware detection model for Android using the multi-view technique and multi-objective feature selection. Initially, a set of multiple features, referred to as multi-view, is extracted from an Android application, which is then used to build a feature vector for the classification task. Then, a multi-objective optimization algorithm is applied to select a subset of features that reduces the model's error rate and inference time. Two classification models are applied for each feature subset using the ensemble method with majority voting. Experiments demonstrated the feasibility of our proposal. Compared to a single-view model without feature selection, our method improved the true positive rates by an average of 4.4 while requiring up to 65% less in inference processing costs.*

Keywords— *Android, Malware Detection, Machine Learning, Multi-objective Optimization*

Resumo. *Este artigo propõe um modelo de detecção de malware para Android utilizando a técnica de multi-view e a técnica de seleção de características com multiobjetivo. Inicialmente, um conjunto de múltiplas características, chamado de multi-view, é extraído de um aplicativo Android, com o qual constrói-se um vetor de características que é utilizado na tarefa de classificação do aplicativo. Em seguida, aplica-se um algoritmo de otimização multiobjetivo para selecionar um subconjunto de características que reduza a taxa de erro do modelo e o tempo de inferência. Assim, para cada subconjunto de características aplica-se dois modelos de classificação utilizando o método ensemble com voto majoritario. Experimentos demonstraram a viabilidade de nossa proposta. Comparando com um modelo de única view e sem seleção de características, o nosso método melhorou as taxas de verdadeiro positivo em uma média de 4,4, exigindo até 65% dos custos com processamento de inferência.*

Plavras-Chave— *Android, Detecção de Malware, Aprendizado de Máquina, Otimização Multiobjetivo*

1. Introdução

À medida que o sistema operacional para dispositivos móveis *Android* tornou-se o mais utilizado do mundo com cerca de 3 bilhões de usuários ativos, totalizando quase três quartos do atual mercado [AndroidStats 2024], também se tornou popular os aplicativos maliciosos, conhecidos como *malware*, ao longo do tempo. Considerando um relatório de cibersegurança [Kaspersky 2023], em 2023 detectou-se um aumento de 52% no número

de amostras de *malware* para essa plataforma, sendo muitas dessas amostras oriundas da loja oficial de aplicativos *Google Play*.

Os métodos de detecção de *malware* para *Android* seguem, em geral, duas abordagens, quais sejam, uma implementação dinâmica ou uma implementação estática. Os modelos que utilizam uma implementação dinâmica, por um lado, buscam por indícios maliciosos durante a execução de um aplicativo. Trata-se, com isso, de uma técnica que exige um significativo esforço na tarefa de detecção. Já os modelos que utilizam uma implementação estática, por outro lado, analisam características do aplicativo, tais como, permissões, chamadas de API e códigos de operação. Trata-se, portanto, de uma abordagem geralmente preferida na literatura em virtude da sua fácil condução, ou seja, velocidade e escalabilidade, e resultados promissores [Geremias et al. 2022].

Considerando a técnica de análise estática, geralmente ela é implementada analisando os arquivos que compõe um arquivo APK (*Android Application Package*), podendo ser o arquivo *manifest.xml*, desde o qual são extraídas as permissões da aplicação ou o arquivo *.dex*, desde o qual é extraído o código binário com os códigos de operação e as chamadas de API [Darwaish and Nait-Abdesselam 2020, dos Santos et al. 2023]. Inúmeras abordagens para a tarefa de classificação de malware para *Android* foram introduzidas na literatura nos últimos anos, dentre os quais temos os modelos baseados em aprendizado de máquina. Trata-se de uma abordagem, em geral, que utiliza um vetor de características do aplicativo como entrada para um modelo de aprendizado de máquina, com o qual é feita a classificação do aplicativo em benigno (*goodware*) ou malicioso (*malware*).

Embora possamos encontrar resultados importantes na literatura, que mostram alta precisão na classificação, as técnicas atuais baseadas em aprendizado de máquina são raramente implementadas em ambientes de produção [Smith et al. 2020]. Isso ocorre porque o comportamento de um malware em dispositivos Android é frequentemente caracterizado por múltiplos indícios maliciosos, o que geralmente exige a análise de diversos arquivos para uma classificação precisa. Por exemplo, um arquivo APK pode solicitar diversas permissões altamente sensíveis e utilizá-las de maneira apropriada durante a análise do código fonte [Geremias et al. 2023]. Se a classificação se basear apenas nas permissões solicitadas, o APK pode ser erroneamente classificado como malware. Na literatura, observa-se que a maioria das abordagens atuais depende da análise de um único arquivo APK para realizar a classificação [Santos et al. 2023].

A combinação de múltiplos conjuntos de características para a classificação de *malware* em *Android* tem sido amplamente explorada na literatura [Darwaish and Nait-Abdesselam 2020]. Em geral, a maioria dos modelos propostos utiliza redes neurais profundas (DNN), que tendem a melhorar a precisão do sistema, embora impliquem desvantagens significativas em termos de requisitos de memória e processamento [Ravi et al. 2022]. No entanto, a seleção de características em um ambiente multi-view é desafiadora, pois cada classificador deve considerar o desempenho de classificação obtido durante o processo de seleção de características.

Diante disso, este artigo apresenta um modelo de seleção de características *multi-view* com otimização multiobjetivo para a classificação de *malware* em *Android*. O processo inicia-se com a extração de um conjunto de características provenientes de diferentes partes de um arquivo APK Android, incluindo permissões, chamadas de API (*API*

Calls) e códigos de operações (*Opcodes*). Esses três elementos de um arquivo APK constituem o que denominamos neste artigo de *multi-view*, com os quais se cria um vetor de características complementares do aplicativo. Posteriormente, esse vetor é utilizado como base para a seleção de um subconjunto características através de um algoritmo de otimização multiobjetivo. Os subconjuntos selecionados são então empregados como entrada para os modelos de classificação (DT, RF), que são desenvolvidos utilizando a técnica ensemble com voto majoritário. Os objetivos de otimização são reduzir a taxa de erro dos modelos e diminuir o tempo de inferência. Como resultado, nossa abordagem aproveita subconjuntos de características complementares para a classificação de *malware* em *Android*, minimizando os custos de processamento.

Ademais, as questões de pesquisa que orientam este trabalho são:

- **QP1**: Qual é a precisão na detecção tradicional de *malware* em *Android* utilizando aprendizado de máquina com uma única *view*?
- **QP2**: De que maneira nossa abordagem de otimização multiobjetivo aprimora a acurácia da classificação?
- **QP3**: Quais são as vantagens e desvantagens do nosso modelo em termos de custo de processamento, medido pelo tempo de inferência?

Por fim, as principais contribuições deste trabalho são:

- *Dataset* com 40 mil amostras de aplicativos *Android*, disponível publicamente, composto por características relacionadas às permissões, *API calls* e *Opcodes*;
- Modelo de detecção de *malware* para *Android* implementado com *multi-view* e estratégia de otimização multiobjetivo. Nossa proposta pode melhorar a taxa verdadeiro positivo em uma média de 4,4, exigindo até 65% dos custos com processamento.

2. Trabalhos Relacionados

A detecção de *malware* em dispositivos *Android* utilizando técnicas baseadas em aprendizado de máquina tem se tornado um tema amplamente estudado na literatura nos últimos anos [Qiu et al. 2020, Horchulhack et al. 2024]. As metodologias propostas buscam alcançar alta acurácia em determinados conjuntos de dados de teste. Por exemplo, D. O. Sahin *et al.* [Şahin et al. 2021] aplicaram uma técnica de seleção de características em um modelo de detecção de *malware* baseado em permissões, utilizando aprendizado de máquina. Essa abordagem resultou em uma melhora na precisão da classificação, porém, foi realizada com um conjunto de dados desatualizado e com um número limitado de amostras. S. Seraj *et al.* [Seraj et al. 2022] introduziram um conjunto de dados mais recente, contendo uma maior quantidade de amostras de *malware*, e utilizaram uma técnica baseada em redes neurais profundas (DNN), o que aprimorou a acurácia em comparação com estudos anteriores. No entanto, essa pesquisa não explorou o potencial de *multi-view* para melhorar a generalização do modelo. A. Pektas *et al.* [Pektaş and Acarman 2020] propuseram o uso de *opcodes* para a detecção de *malware*, conseguindo uma melhora na acurácia ao combinar essa abordagem com a seleção de características. Ainda assim, a pesquisa não considerou como a abordagem *multi-view* poderia ser utilizada para aumentar a confiabilidade do modelo. A. Darwaish *et al.* [Darwaish and Nait-Abdesselam 2020] sugeriram outra técnica que traduz o código binário em uma imagem que é, em seguida,

usada na classificação. Apesar dessa proposta ter melhorado a acurácia, ela também não explorou o uso de conjuntos de características múltiplas.

Nos últimos anos, a combinação de múltiplos conjuntos de características para a detecção de *malware* em dispositivos *Android* também tem sido objeto de diversas pesquisas [dos Santos et al. 2021]. S. Millar *et al.* [Millar et al. 2021] propuseram o uso de *opcodes*, permissões e pacotes de API para a tarefa de classificação. Embora essa abordagem tenha melhorado a precisão da classificação, ela não investigou como as diferentes *views* poderiam ser combinadas.

A seleção de características foi sugerida como uma maneira de aprimorar a classificação em um contexto *multi-view*. Y. Wu *et al.* [Wu et al. 2023] aplicaram aprendizado por reforço na tarefa de seleção de características, o que aumentou a acurácia, mas não considerou a aplicação de *multi-view*. M. Azad *et al.* [Azad et al. 2022] propuseram o uso de algoritmos de otimização por enxame de partículas para a seleção de características em um classificador baseado em redes neurais profundas (DNN). Embora essa abordagem tenha melhorado a precisão, ela também negligenciou a aplicação de *multi-view*. De forma semelhante, H. Hawks *et al.* propuseram uma abordagem de seleção de características combinada com diferentes classificadores de aprendizado de máquina para a detecção de *malware*. Embora essa metodologia possa aumentar a precisão da classificação, ela não explorou a aplicação de *multi-view*. Assim, as abordagens atuais na literatura geralmente não consideram como o uso de *multi-view* pode ser explorado para aumentar a confiança da detecção de *malware* em dispositivos *Android*.

3. Modelo de Detecção de *Malware* para *Android* com *Multi-view*

Para resolver o desafio acima mencionado, qual seja, a detecção de *malware* para *Android* em um cenário *multi-view*, o esquema proposto é implementado através de uma abordagem de otimização multiobjetivo. A Figura 1 ilustra a operação do modelo proposto.

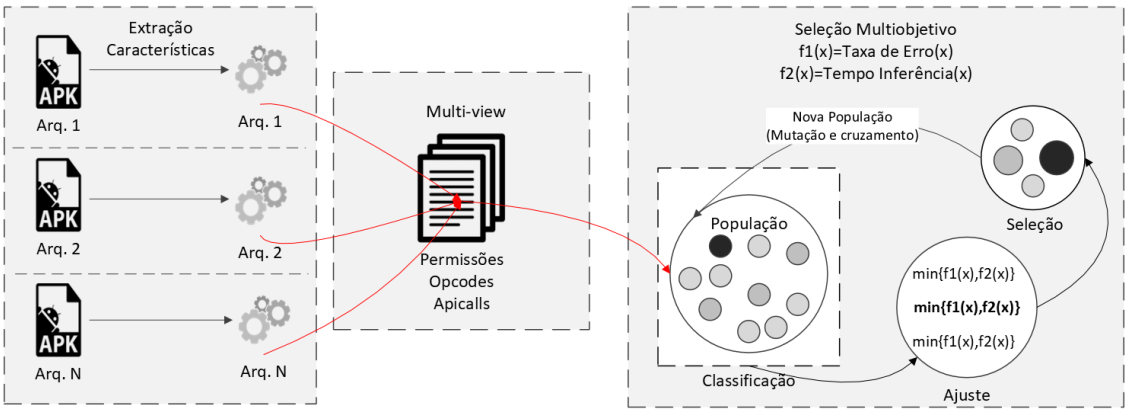


Figura 1. Um modelo de detecção com *multi-view*, no centro, e otimização multiobjetivo, no lado direito.

A proposta considera inicialmente a fase de extração das características por meio da análise estática, através de um módulo de extração (Figura 1, *Extração de Características*), que gera um conjunto de características do aplicativo com informações sobre as permissões, *API Calls* e *Opcodes*. O nosso principal pressuposto é que essas características, que compõem um conjunto *multi-view*, podem ser utilizadas para melhorar a

generalização e a confiança da detecção de *malware* para *Android*. Em seguida, o conjunto multi-view alimenta um pipeline de classificação baseado em aprendizado de máquina e seleção de caracterísicas com otimização multiobjetivo (Figura 1, *Seleção Multiobjetivo*). Cada uma das *views* do conjunto *multi-view* é classificada por um modelo de aprendizado de máquina (DT e RF), produzindo um resultado de classificação para cada *view*, os quais são combinados por uma votação por maioria, produzindo o resultado final da classificação (Figura 1, *Classificação*).

A abordagem de otimização multiobjetivo é utilizada para enfrentar o desafio de combinar múltiplas *views*. O nosso objetivo é otimizar as características utilizadas por cada *view* com base na precisão de classificação combinado com os custos computacionais (Figura 1, textitOtimização Multiobjetivo). A nossa principal conclusão é que a seleção de características pode ser utilizada para melhorar a precisão da detecção de *malware* ao mesmo tempo que viabiliza um processo de classificação com *multi-view*.

As subsecções seguintes descrevem mais pormenorizadamente o modelo proposto, incluindo os seus componentes de implementação.

3.1. Classificação *multi-view*

As abordagens atuais na literatura para a detecção de *malware* em *Android* costumam se basear em uma única *view* para a tarefa de classificação. Isso resulta em modelos que não apresentam a generalização necessária para aplicação em ambientes de produção. Para superar esse desafio, o modelo proposto adota um processo de classificação *multi-view*, implementado por meio de classificadores *ensemble*.

O funcionamento do nosso modelo está ilustrado na Figura 1. Ele começa com a análise de arquivos APK de *Android*. Assim, os arquivos usados no processo de extração das características são obtidos. Na prática, o nosso modelo considera três *views*: permissão (*manifest.xml*), *Opcode* (*dex*) e *API calls* (*dex*) (descritos na Seção 4.1). Assim, um vetor de características é criado a partir de um conjunto de características de cada arquivo. Cada vetor de características resultante é classificado por um classificador. Por fim, para combinar os múltiplos resultados das classificações, o nosso modelo utiliza um procedimento de votação por maioria.

Como resultado, o nosso modelo funciona seguindo um método de *multi-view* com o objetivo de melhorar a generalização e a confiança do sistema resultante. Além disso, ele combina os resultados dos classificadores através de um simples procedimento de votação por maioria, mantendo assim os custos de processamento de inferência mínimos. A próxima seção descreve detalhadamente como foi feito o melhoramento da combinação de *multi-view*.

3.2. Otimização multiobjetivo

Para combinar múltiplas *views*, utilizamos uma abordagem de otimização multiobjetivo que consiste em otimizar o tempo de processamento de inferência e a precisão resultante do grupo de classificadores simultaneamente. Para atingir tal objetivo, implementamos a otimização multiobjetivo como uma seleção de características baseada em *wrapper* através de um algoritmo de busca genético multiobjetivo (Figura 1, Ajuste, Seleção, Mutação, e Cruzamento). A implementação e os parâmetros utilizados são descritos com maior detalhe na Secção 4.3).

Seja N o número de *views* usadas e seja h o número de classificadores, onde cada *view* $x_i \subseteq \mathbb{R}^n$, de modo que cada *view* x_i é um espaço único de características. A tarefa de otimização multiobjetivo visa encontrar um subconjunto de características para cada *view*, de modo a minimizar o tempo de processamento e a taxa de erro. A função objetivo $obj_{\text{proc}}(h, x)$ é definida como a soma do tempo de processamento $tempoProcessamento(h_i(x_i))$ para cada i variando de 1 até N. Isso significa que o tempo total de processamento é obtido somando-se os tempos individuais de processamento de cada componente $h_i(x_i)$. Já a função objetivo $obj_{\text{erro}}(h, x)$ é definida como o máximo entre a soma das saídas $h_i(x_i)$ para a classe *goodware* e a soma das saídas $h_i(x_i)$ para a classe *malware*, onde i varia de 1 até N.

Em que obj_{proc} mede o tempo total de processamento de inferência do classificador h para toda *view* x_i, e obj_{erro} mede a taxa de erro do conjunto de classificadores com base nos votos de confiança máxima do conjunto. obj_{erro} mede o resultado da classificação do conjunto de acordo com a soma máxima dos valores de confiança da classificação individual para cada classe, em que $h_i(x_i)^{goodware}$ denota a confiança do classificador h_i para $goodware$ e $h_i(x_i)^{malware}$, o classificador h_i para $malware$. O objetivo é encontrar o vetor de variáveis $\tilde{x_i}, ..., \tilde{x_N}$ que minimiza simultaneamente as duas seguintes expressões. A primeira expressão é a soma dos valores da função objetivo $obj_{\text{proc}}(h, x_j)$ para cada j variando de 1 até $\mathcal{D}$. A segunda expressão é a minimização de $1-$ a soma da acurácia $acurácia(x_j, obj_{\text{erro}}(h, x_j))$, novamente para cada j variando de 1 até $\mathcal{D}$.

Em que $\mathcal{D}$ é um conjunto de dados de teste e $\tilde{x}_i$ é um subespaço de características da *view* x_i. Por conseguinte, o modelo visa encontrar o subespaço de características de cada *view* que otimiza o conjunto resultante do tempo de processamento de inferência e a acurácia. Nesse último caso, o nosso objetivo é minimizar o inverso da acurácia obtida do sistema $(1 - acurácia)$. Como resultado, o modelo proposto pode considerar a aplicação de detecção de *malware* para *Android* com *multi-view*, levando em conta a acurácia resultante do conjunto. A nossa ideia é selecionar cada subespaço de características da *view* enquanto medimos a precisão do conjunto resultante e os custos de processamento.

4. Avaliação

Nesta seção, investigamos o desempenho do modelo proposto. Especificamente, visamos responder as seguintes *Questões de Pesquisa* (QP):

- **QP1**: Qual é a precisão na detecção tradicional de *malware* em *Android* utilizando aprendizado de máquina com uma única *view*?
- **QP2**: De que maneira nossa abordagem de otimização multiobjetivo aprimora a acurácia da classificação?
- **QP3**: Quais são as vantagens e desvantagens do nosso modelo em termos de custo de processamento, medido pelo tempo de inferência?

As seções seguintes descrevem em pormenor o nosso procedimento de construção do modelo e o seu desempenho.

4.1. Um *dataset* realista de *malware* para *android*

Para avaliar de forma confiável o desempenho do nosso modelo em um contexto de *multi-view*, criamos um *dataset* com 40 mil amostras de arquivos APK para dispositivos *Android*. Essas amostras foram obtidas da plataforma *AndroZoo* [Allix et al. 2016] e estavam disponíveis entre 2022 e 2024, dentre os quais 20 mil são de *goodware* e 20 mil de

malware. Cada um dos arquivos foi categorizado através da API do *VirusTotal* [Virustotal 2024], sendo considerado *malware* sempre que ao menos 2 soluções no *VirusTotal* marcaram-na como tal. Caso contrário, elas foram consideradas *goodware*.

As características de cada arquivo APK foram extraídas utilizando a ferramenta *AndroPyTool* [Martín et al. 2019]. Além disso, desenvolvemos um protótipo que padroniza essas características extraídas, tornando-as apropriadas para aplicações de aprendizado de máquina. Para nossas experimentações, consideramos três *views*, que são:

- ***API Calls***. Um total de 63.460 características que representam a contagem de cada chamada de API realizada pelo código-fonte binário (*dex*);
- ***Opcode***. Um conjunto de 224 características que contabiliza o número de ocorrências de cada Opcode no código-fonte binário (*dex*);
- ***Permissões***. Um total de 19.083 características que identifica as permissões solicitadas no arquivo *manifest.xml*.

O conjunto de dados resultante foi dividido em três subconjuntos preliminares: treino, validação e teste, contendo respectivamente 40%, 30% e 30% dos arquivos APK, selecionados aleatoriamente sem reposição.

O conjunto de treino foi utilizado para construir os classificadores desenvolvidos nesta pesquisa. O conjunto de teste, por sua vez, foi empregado no processo de otimização multiobjetivo. Já o conjunto de validação foi utilizado para avaliar a precisão final do sistema. Como resultado, nosso conjunto de dados abrange uma ampla gama de características dos arquivos APK, permitindo validar a capacidade de generalização das técnicas de detecção de malware baseadas em aprendizado de máquina.

4.2. Construção do modelo

Para avaliar nossa abordagem, utilizamos dois classificadores amplamente empregados na detecção de *malware* para *Android*: *Decision Tree* (DT) e *Random Forest* (RF). O classificador DT foi configurado utilizando o critério $gini$ para medir a qualidade dos nós, sem impor um limite máximo à profundidade da árvore. Já o classificador RF foi parametrizado com 100 árvores de decisão como *base learner*, cada uma utilizando os mesmos parâmetros aplicados ao classificador DT.

O *dataset* foi normalizado utilizando o método *min-max*. Os classificadores foram implementados utilizando a API *scikit-learn* na versão $v0.24$. Para lidar com o grande número de características extraídas para cada *view*, aplicamos PCA com 100 componentes para reduzir a dimensionalidade, tanto na fase de treino quanto na de teste dos modelos de aprendizado de máquina. A avaliação dos classificadores foi baseada em suas taxas de TP e TN, onde TP refere-se à porcentagem de amostras de *malware* corretamente classificadas como *malware*, e TN à porcentagem de amostras de *goodware* corretamente classificadas como *goodware*.

4.3. Detecção de *Malware* para *Android* com *Multi-view*

Nossa primeira experiência tem como objetivo responder à *QP1* e avaliar a acurácia das técnicas tradicionais de detecção de *malware* para *Android* baseadas em aprendizado de máquina. Para isso, analisamos o desempenho dos classificadores previamente selecionados (ver Seção 4.2) em um cenário de classificação utilizando apenas uma única *view*.

Tabela 1. Resultados de acurácia dos modelos adotados de detecção de *malware* para *Android* baseados em aprendizado de máquina.

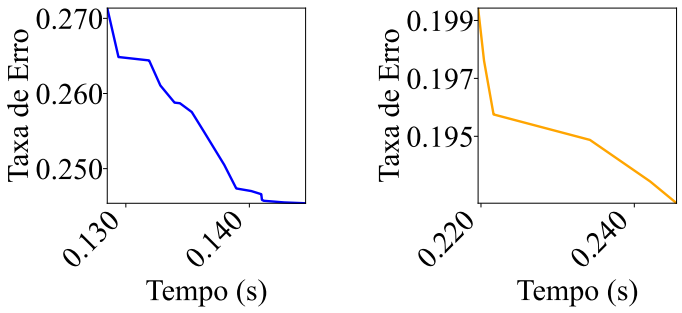
View	Classificador	Acurácia (%)		
		TP	***TN***	***F1***
API Calls	DT	70.91	71.02	0.7095
	RF	83.17	77.18	0.8084
Opcode	DT	70.34	70.91	0.7045
	RF	84.32	76.57	0.8116
Permissões	DT	68.43	68.82	0.6849
	RF	78.45	75.14	0.7741
Nossos Resultados	DT	75.86	75.35	0.7574
	RF	87.22	78.18	0.8324

A Tabela 1 apresenta a acurácia de classificação das técnicas adotadas ao considerar cada *view* isoladamente. Observa-se que, em média, os classificadores apresentaram uma precisão de classificação relativamente baixa. Por exemplo, o classificador RF atingiu taxas de TP de 83%, 84% e 78% ao utilizar as *views API Calls*, *Opcode* e permissões, respectivamente. Esses resultados indicam que a detecção de *malware* para *Android* baseada em aprendizado de máquina, quando realizada com uma única *view*, não oferece o nível necessário de confiança para uma implementação em ambiente de produção.

Nossa segunda experiência tem como objetivo responder à *QP2* e avaliar a precisão de classificação do modelo proposto. Para isso, iniciamos a análise do modelo utilizando o método de otimização multiobjetivo (ver Seção 3.2). Implementamos nossa abordagem como uma seleção de características baseada em *wrapper* utilizando o *Non-dominated Sorting Genetic Algorithm* (NSGA-III) [Deb et al. 2002], por meio da API *pymoo*. Nesse contexto, o NSGA-III foi configurado com uma população de tamanho 100, 100 gerações, taxa de cruzamento de $0,3$ e probabilidade de mutação de $0,1$. O objetivo da seleção de características multiobjetivo é reduzir tanto o tempo de inferência quanto a taxa de erro, conforme medido no conjunto de dados de validação. No entanto, nosso esquema foi avaliado sem variar a *view* utilizada pelo classificador subjacente (Figura 1, *Classificação*). Na prática, nosso método sempre emprega os classificadores DT e RF, alterando apenas as *views* utilizadas. Essa característica permite uma comparação direta do desempenho da nossa proposta em relação ao esquema tradicional avaliado anteriormente.

A Figura 2 ilustra a curva de Pareto do modelo proposto para cada classificador considerado. Observa-se uma correlação direta entre os custos de processamento de inferência e a taxa de erro do sistema. É crucial destacar que, na prática, o operador deve selecionar o ponto de operação que melhor se alinha às necessidades da sua aplicação específica. Para nossos testes, escolhemos o ponto de operação mais próximo de zero em ambos os objetivos.

Com base nos pontos de operação selecionados, iniciamos a avaliação do desempenho do modelo proposto em termos de eficácia. A Tabela 1 apresenta a acurácia de classificação do nosso esquema conforme o classificador utilizado. Nossa abordagem demonstrou uma melhoria significativa na precisão da classificação para todos os classi-



(a) DT *Multi-view* **(b) RF *Multi-view***

Figura 2. Curva de Pareto para o modelo proposto.

ficadores analisados. Por exemplo, ao utilizar as views *API Calls*, *Opcode* e permissões, respectivamente, com o classificador DT, nosso esquema foi capaz de aumentar o F1 em $0,05$, $0,05$ e $0,08$, em comparação com o uso de uma única *view*. Em média, nossa metodologia elevou os índices TP em $4,5$ pontos e os índices TN em $3,9$ pontos. Portanto, a estratégia de otimização multiobjetivo implementada no nosso esquema contribui para um aumento expressivo na precisão da classificação.

Por fim, para responder à *QP3*, investigamos os custos de processamento de inferência do nosso esquema em comparação com a abordagem tradicional. Para esse fim, avaliamos os custos de processamento de inferência sem a aplicação da otimização multiobjetivo (utilizando todas as características) em relação aos custos de processamento com as características selecionadas pelo nosso modelo (Tabela 1). Em média, nosso modelo exigiu apenas 65%, 78% e 66% dos custos de processamento correspondentes ao uso de todas as características com os classificadores DT e RF, respectivamente. Assim, o esquema proposto não apenas melhora a precisão geral do sistema, mas também reduz significativamente os custos de processamento de inferência associados.

5. Conclusão

A detecção de *malware* para *Android* utilizando técnicas baseadas em aprendizado de máquina tem sido extensivamente investigada na literatura nos últimos anos. No entanto, apesar dos resultados promissores, as abordagens atuais são raramente implementadas em ambientes de produção. Este artigo busca enfrentar esse desafio por meio de uma abordagem *multi-view* para a classificação de *malware* em *Android*, utilizando otimização multiobjetivo. O modelo proposto seleciona o subconjunto de características de cada *view*, maximizando a precisão da classificação quando combinadas através de um conjunto de classificadores. As experiências conduzidas em um novo *dataset* demonstram a viabilidade da nossa proposta, resultando em uma melhora significativa na precisão e uma redução nos custos computacionais de inferência. Como trabalho futuro, planejamos expandir nosso modelo para incorporar classificadores baseados em aprendizado profundo, combinados com a classificação *multi-view* e seleção de características.

Agradecimentos

Este trabalho foi parcialmente financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), termos 304990/2021-3 e 407879/2023-4, e

Referências

Allix, K., Bissyandé, T. F., Klein, J., and Traon, Y. L. (2016). Androzoo: Collecting millions of android apps for the research community. *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 468–471.

AndroidStats (2024). Android statistics. `https://www.businessofapps.com/data/android-statistics/`. [online: acessado em 02-junho-2024].

Azad, M. A., Riaz, F., Aftab, A., Rizvi, S. K. J., Arshad, J., and Atlam, H. F. (2022). Deepsel: A novel feature selection for early identification of malware in mobile applications. *Future Generation Computer Systems*, 129:54–63.

Darwaish, A. and Nait-Abdesselam, F. (2020). Rgb-based android malware detection and classification using convolutional neural network. In *IEEE Global Communications Conference*.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

dos Santos, R. R., Viegas, E. K., and Santin, A. O. (2021). A reminiscent intrusion detection model based on deep autoencoders and transfer learning. In *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE.

dos Santos, R. R., Viegas, E. K., Santin, A. O., and Tedeschi, P. (2023). Federated learning for reliable model updates in network-based intrusion detection. *Computers amp; Security*, 133:103413.

Geremias, J., Viegas, E. K., Santin, A. O., Britto, A., and Horchulhack, P. (2022). Towards multi-view android malware detection through image-based deep learning. In *2022 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE.

Geremias, J., Viegas, E. K., Santin, A. O., Britto, A., and Horchulhack, P. (2023). Towards a reliable hierarchical android malware detection through image-based cnn. In *2023 IEEE 20th Consumer Communications amp; Networking Conference (CCNC)*. IEEE.

Horchulhack, P., Viegas, E. K., Santin, A. O., and Simioni, J. A. (2024). Network-based intrusion detection through image-based cnn and transfer learning. In *2024 International Wireless Communications and Mobile Computing (IWCMC)*. IEEE.

Kaspersky (2023). Attacks on mobile devices significantly increase in 2023. `https://www.kaspersky.com/about/press-releases/2024_attacks-on-mobile-devices-significantly-increase-in-2023`. [online: acessado em 02-junho-2024].

Martín, A., Lara-Cabrera, R., and Camacho, D. (2019). Android malware detection through hybrid features fusion and ensemble classifiers: The andropytool framework and the omnidroid dataset. *Information Fusion*, 52:128–142.

Millar, S., McLaughlin, N., Martinez del Rincon, J., and Miller, P. (2021). Multi-view deep learning for zero-day android malware detection. *Journal of Information Security and Applications*, 58:102718.

Pektaş, A. and Acarman, T. (2020). Learning to detect android malware via opcode sequences. *Neurocomputing*, 396:599–608.

Qiu, J., Zhang, J., Luo, W., Pan, L., Nepal, S., and Xiang, Y. (2020). A survey of android malware detection with deep neural models. *ACM Computing Surveys*, 53(6):1–36.

Ravi, V., Alazab, M., Selvaganapathy, S., and Chaganti, R. (2022). A multi-view attention-based deep learning framework for malware detection in smart healthcare systems. *Computer Communications*, 195:73–81.

Santos, R. R. d., Viegas, E. K., Santin, A. O., and Cogo, V. V. (2023). Reinforcement learning for intrusion detection: More model longness and fewer updates. *IEEE Transactions on Network and Service Management*, 20(2):2040–2055.

Seraj, S., Khodambashi, S., Pavlidis, M., and Polatidis, N. (2022). Hamdroid: permission-based harmful android anti-malware detection using neural networks. *Neural Computing and Applications*, 34(18):15165–15174.

Smith, M. R., Johnson, N. T., Ingram, J. B., Carbajal, A. J., Haus, B. I., Domschot, E., Ramyaa, R., Lamb, C. C., Verzi, S. J., and Kegelmeyer, W. P. (2020). Mind the gap: On bridging the semantic gap between machine learning and malware analysis. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, CCS '20. ACM.

Virustotal (2024). Analyze suspicious files. `https://www.virustotal.com/`. [online: acessado em 02-junho-2024].

Wu, Y., Li, M., Zeng, Q., Yang, T., Wang, J., Fang, Z., and Cheng, L. (2023). Droidrl: Feature selection for android malware detection with reinforcement learning. *Computers amp; Security*, 128:103126.

Şahin, D. O., Kural, O. E., Akleylek, S., and Kılıç, E. (2021). A novel permission-based android malware detection system using feature selection based on linear regression. *Neural Computing and Applications*, 35(7):4903–4918.