Uma Análise de Métodos de Seleção de Características aplicados à Detecção de Malwares Android

Taina Soares[1], Diego Kreutz[1], Vanderson Rocha[2], Estevão Costa[2], Luiza Leão[2] Jonas Pontes[2], Joner Assolin[2], Gustavo Rodrigues[3], Eduardo Feitosa[2]

[1]Universidade Federal do Pampa (UNIPAMPA)
{TainaSoares.aluno,DiegoKreutz}@unipampa.edu.br

[2]Universidade Federal do Amazonas (UFAM)
vanderson@ufam.edu.br
{ecc,pontes,joner.assolin,lpml,efeitosa}@icomp.ufam.edu.br

[3]Combate à Fraude
gustavo.rodrigues@combateafraude.com

Abstract. *Android malware detection requires training machine learning models using datasets with a large number of samples (e.g., 100k, 1M) and features (e.g., 3k, 500k). However, those datasets require significant computation time and can impact the models' quality. To address such issues, researchers have proposed feature (e.g., permissions, API calls) selection methods. In this paper, we evaluate four of such feature selection methods (SigPID, SigAPI, RFG, ALR) using three datasets different from the ones used in their original evaluations. Our results indicate a strong correlation between datasets and feature selection methods, even for specific types of features (e.g., permissions).*

Resumo. *A detecção de malwares Android requer tipicamente o treinamento de modelos de aprendizado de máquina utilizando datasets que contém números expressivos de amostras (e.g., 100k, 1M) e características (e.g., 3k, 500k). Para reduzir a dimensionalidade dos datasets, pesquisadores vêm recorrentemente propondo diferentes métodos para a seleção de características (e.g., permissões, chamadas de API). Neste trabalho, avaliamos quatro métodos de seleção de características (SigPID, SigAPI, RFG, ALR) utilizando três datasets diferentes dos utilizados na avaliação original dos métodos. Os resultados indicam que há uma forte relação entre os datasets e os métodos de seleção, mesmo para tipos específicos de características (e.g., permissões).*

1. Introdução

A principal técnica utilizada na detecção de *malwares* Android são os modelos de aprendizado de máquina, que objetivam classificar aplicativos em malignos e benignos [Agrawal and Trivedi, 2021]. Os modelos são treinados e validados com conjuntos de dados, mais conhecidos como *datasets*, que contém amostras rotuladas, em benignas ou malignas, de aplicativos Android. As amostras vêm acompanhadas de conjuntos de características, como permissões, intenções, chamadas de API, chamadas de sistema, tráfego de rede, componentes de *hardware*, componentes do aplicativo, comandos de sistema, *bytecodes* e *strings* semânticas [Damodaran et al., 2017, Qiu et al., 2020].

Um dos principais desafios enfrentados na construção de modelos de detecção de *malwares* Android é a quantidade de dados que um *dataset* pode conter. Por exemplo,

há *datasets* que chegam a conter mais de 1 milhão de amostras e mais de 1 milhão de características [Roy et al., 2015]. Em termos computacionais, é demasiadamente custoso treinar e validar modelos com *datasets* dessa magnitude. Na prática, um *dataset* contendo apenas 6 mil amostras e 643 características pode consumir até 5,8 horas de computação em um computador moderno [Cai et al., 2021]. Portanto, trabalhar com *datasets* grandes torna-se inviável em termos de custo financeiro e tempo computacional. Além disso, o treinamento contínuo de modelos precisa ser eficiente para tornar viável a detecção de *malwares* em tempo real.

Reduzir a dimencionalidade do *dataset*, através da seleção das características mais significativas, é parte de um conjunto de otimizações possíveis e necessárias para viabilizar e dar qualidade ao treinamento e a validação dos modelos. Enquanto que na literatura há um conhecimento vasto sobre preparação dos dados, como a remoção de amostras duplicadas, adaptação de tipos de dados e a remoção de outros ruídos, e otimização dos parâmetros dos algoritmos de aprendizado de máquina, através de algoritmos como *grid search* e *random search*, há poucos estudos que avaliam e comparam o impacto e os desafios dos diferentes métodos de seleção das características, atividade importante para o desenvolvimento de modelos otimizados e viáveis na prática. A redução da dimensionalidade do *dataset*, através de métodos como SigPID [Sun et al., 2016], SigAPI [Galib and Hossain, 2020] e Robust Feature Generation (RFG) [Moutaz, 2020]), melhora o desempenho dos modelos, pois diminui o tempo de aprendizado, aumenta a capacidade de generalização da classificação e os tornam melhores em termos de escalabilidade, já que a representação e compreensão dos dados é superior [Venkatesh and Anuradha, 2019, Golrang et al., 2021].

Técnicas de seleção de características são frequentemente utilizadas para auxiliar os modelos de detecção de *malwares* Android [Lee et al., 2021, Mahindru and Sangal, 2021, Smmarwar et al., 2022]. Essas técnicas são implementadas através de métodos otimizados para encontrar o melhor e menor subconjunto de características no *dataset*, capaz de reduzir a complexidade do processamento e melhorar o desempenho do modelo. Cada um desses métodos pode incorporar diversas técnicas de seleção, como regressão linear e ganho de informação, para trabalhar com diferentes tipos de características. Por exemplo, o método SigAPI utiliza diferentes técnicas de seleção para reduzir entre 75% e 80% o número de caracterísitcas de *datasets* como o Drebin e o Android Malware Genome Project (Malgenome) [Galib and Hossain, 2020].

Na literatura podemos encontrar uma variedade de métodos de seleção de características [Sun et al., 2016, Moutaz, 2020, Salah et al., 2020, Şahin et al., 2021a, Wang et al., 2021, Mahindru and Sangal, 2021, Smmarwar et al., 2022]. Entretanto, apesar de métodos como SigPID [Sun et al., 2016] e RFG [Moutaz, 2020] implementarem e avaliarem diferentes técnicas e métricas de seleção de caraterísticas, esses métodos são utilizados de forma pontual, isolada e geralmente apenas comparados com técnicas clássicas de seleção de características, como ganho de informação e chi-quadrado. Por exemplo, enquanto o SigPID é utilizado para selecionar as permissões mais relevantes, o RFG é utilizado para selecionar as chamadas de API mais relevantes. Resumidamente, esses trabalhos demonstram o resultado positivo da seleção de características para um conjunto limitado de *datasets* (e.g., tipicamente um único conjunto de dados) e considerando apenas um sub-conjunto de técnicas clássicas de características na comparação.

Neste trabalhos procuramos dar um passo além e avaliar, de maneira empírica e exploratória, dois grupos de métodos, um voltado para permissões, composto pelos métodos SigPID [Sun et al., 2016] e ALR [Şahin et al., 2021b], e outro voltado para chamadas API, composto pelos métodos SigAPI [Galib and Hossain, 2020] e RFG [Moutaz, 2020]. É importante ressaltar também que permissões e chamdas de API são consideradas as principais e mais significantes características recorrentemente utilizadas para detecção de *malwares* Android [Wang et al., 2019, Liu et al., 2020, Kouliaridis and Kambourakis, 2021].

Para avaliar os métodos, que incorporam diversas técnicas de seleção, como *recursive feature elimination*, *mutual information gain*, ANOVA, Chi-Square e *SelectKBest with chi2*), utilizamos três *datasets* distintos, o Androcrawl, Drebin_215 e MD46K, sendo este último criado pelos autores. A partir de cada um desses três conjuntos de dados, disponíveis no repositório GitHub[1], juntamente com a implementação dos métodos e *scripts* de execução, derivamos dois sub-conjuntos de dados de acordo com as categorias de características de especialidade de cada método. Nosso objetivo é observar o comportamento dos métodos, a partir de dois sub-conjuntos (permissões e chamadas de API), para um grupo de três *datasets* do domínio de detecção de *malwares* Android.

As principais contribuições do trabalho podem ser resumidas em: (a) uma análise de quatro métodos de seleção de características utilizando três *datasets* distintos; e (b) a identificação de uma forte correlação entre os *datasets* e os métodos de seleção de características, o que diminui a generalidade e aplicabilidade dos métodos.

O restante do trabalho está organizado como segue. Nas Seções 2 e 3 apresentamos os trabalhos relacionados e uma descrição sucinta dos métodos. Em seguida, apresentamos a metodologia do trabalho na Seção 4, os resultados na Seção 5 e, na Seção 6, as considerações finais e trabalhos futuros.

2. Estado da Arte

No contexto de detecção de *malwares* Android, trabalhos como [Mas' ud et al., 2017, Nivaashini et al., 2018, Anggraeni et al., 2021] realizaram comparações entre diferentes técnicas de seleção de características, de forma indivualizada ou combinada, visando trazer resultados positivos para os modelos de classificação, como a redução de tempo de computação e o ganho de qualidade de detecção. Entretanto, esses trabalhos exploram os resultados na perspectiva de técnicas clássicas de seleção estatística, como chi-quadrado e ganho de informação, e técnicas de mineração de dados, como regras de associação.

Existem ainda trabalhos que propõe métodos mais elaborados e complexos de seleção de características no domínio de detecção de *malwares* Android, como SigPID [Sun et al., 2016], RFG [Moutaz, 2020], SigAPI [Galib and Hossain, 2020] e ALR [Şahin et al., 2021b]. Estes métodos atuam tipidamente sobre grupos específicos de categorias de características. Por exemplo, o SigPID avalia permissões enquanto que o SigAPI avalia chamadas de API. É importante ressaltar que esses métodos cobrem um repertório complexo, diverso e abrangente de técnicas, estratégias e recursos para a redução de características, incorporando, por exemplo, uma combinação de técnicas clássicas de seleção estatística, algoritmos de mineração de dados e modelos de aprendizado de máquina. O

[1] `https://github.com/Malware-Hunter/sbseg22-feature-selection`

objetivo dessa combinação de técnicas e recursos é tornar mais robusta a seleção de características, o que potencializa melhores resultados em termos de tempo de execução e maior efetividade dos modelos de aprendizado de máquina, posteriormente treinados e validados com o conjunto de características resultantes dos métodos de seleção.

É importante destacar também que os métodos SigPID [Sun et al., 2016], RFG [Moutaz, 2020], SigAPI [Galib and Hossain, 2020] e ALR [Şahin et al., 2021b] podem ser classificados como auto-contidos, ou seja, possuem um nível de detalhamento adequado para a sua implementação e reprodução, principal motivo da escolha dos métodos. Adicionalmente, vale ressaltar que nossas pesquisas exploratórias indicam que não existem análises comparativas desses tipos de métodos para conjuntos de *datasets* diversos, o que pode sucitar questões como: *O comportamento e os resultados dos métodos se mantêm para diferentes conjuntos de dados, isto é, para conjuntos de dados não avaliados pelos autores dos métodos?*

3. Métodos de Seleção de Características

Nesta seção apresentamos resumidamente as principais características dos quatro métodos escolhidos e implementados, isto é, SigAPI, SigPID, RFG e ALR. Como poderá ser observado, cada método possui suas particularidades e foi projetado e avaliado sobre um grupo específico de características e um conjunto limitado de *datasets*.

3.1. SigAPI

O SigAPI [Galib and Hossain, 2020] utiliza uma combinação de técnicas para seletivamente identificar as chamadas de API mais significativas para a detecção de *malwares* Android. O método funciona em duas etapas. Na primeira etapa, o SigAPI aplica e avalia diversas técnicas básicas de seleção, como *Mutual Information Gain*, *Based on Univariate ROC-AUC Score*, *Recursive Feature Elimination with Gradient Boosting Classifier*, *Recursive Feature Elimination with Random Forest Classifier*, *SelectKBest with chi2* e *SelectFromModel*. O melhor resultado entre as diversas técnicas é selecionado para a segunda etapa do método.

A segunda etapa do SigAPI é implementada utilizando o coeficiente de correlação de Pearson, que é aplicado para todas as combinações de pares de chamadas de API derivadas do melhor resultado da primeira etapa. Os pares que apresentam correlação maior que 0,85 são encaminhados para um sub-processo de eliminação de caracteristicas, que é baseado em um estimador implementado através do algoritmo *Random Forest*. O sub-processo procura determinar a importância relativa da característica para a classificação, eliminando aquela que for considerada a menos importante e reduzindo ainda mais a quantidade de chamadas de API do *dataset*. Como as duas características em cada par são fortemente correlacionadas, o pressuposto do método é que a remoção daquela de menor importância não afeta o desempenho da classificação dos modelos. Resultados experimentais indicam que método é capaz de reduzir em mais de 75% o número de chamadas de API, para *datasets* como o Android Malware Genome Project (Malgenome) [Zhou and Jiang, 2012], sem prejudicar o desempenho dos modelos de detecção de *malwares*.

3.2. SigPID

O SigPID [Sun et al., 2016] utiliza três níveis de poda para reduzir o número de permissões e opera com duas matrizes, sendo uma de permissões utilizadas por amostras de *malwares* e outra de características utilizadas por aplicativos benignos. No primeiro nível, o SigPID classifica permissões com taxa negativa através de um sistema incremental que utiliza o SVM. O objetivo na etapa é alcançar a melhor acurácia com um número reduzido de permissões, removendo as que geram ambiguidade ao modelo (e.g., INTERNET, CÂMERA). No trabalho orginal, o primeiro nível de corte possibilitou reduzir o *dataset* de 135 para 95 permissões.

No segundo nível ocorre a classificação de permissões com base em suporte, que busca avaliar a recorrência de uma permissão no intervalo entre 0 e 1. Uma recorrência muito baixa (e.g., 0) indica que seu impacto será baixo no desempenho do modelo, isto é, essas permissões podem ser excluídas do *dataset*. No segundo nível do SigPID foram eliminadas 70 permissões, restando apenas 25 para o terceiro nível de poda.

Finalmente, no terceiro nível ocorre a mineração de permissões com regras de associação, através do algoritmo apriori, com os parâmetros de 96,5% de confiança mínima e 10% de suporte mínimo. O objetivo é identificar as permissões que possuem maior probabilidade de estarem associadas (*e.g,* $WRITE\_SMS$ e $READ\_SMS$). A permissão com o menor suporte é eliminada. No exemplo, a permissão $WRITE\_SMS$ é considerada menos relevante que $READ\_SMS$. Ao final do terceiro nível, os autores chegaram a apenas 22 permissões, isto é, uma redução de 83,70% quando consideramos o *dataset* original, que contém 135 permissões.

3.3. RFG

O método *Robust Feature Generation* (RFG) [Moutaz, 2020] é composto de duas etapas. Na primeira etapa, o RFG utiliza o Chi-quadrado e o ANOVA (análise de variância) para selecionar as N melhores características, em que N pertence a um conjunto definido pelo usuário, que serão avaliadas na etapa seguinte. O Chi-quadrado e o ANOVA são funções que associam uma pontuação para cada característica. Como resultado, características altamente dependentes na classe da amostra (*malware* ou benigno) recebem uma pontuação alta, criando um *ranking* de importância das características. A segunda etapa do RFG consiste em uma avaliação das N melhores características com os algoritmos *Naive Bayes*, *k-Nearest Neighbors* (KNN), *Random Forest*, J48, *Sequential Minimal Optimization* (SMO), *Logistic Regression*, AdaBoost *decision-stump*, *Random Committee*, JRip e *Simple Logistics*. Cada um dos modelos é testado para cada valor de N em validação cruzada. O objetivo desta etapa é encontrar o menor valor de N cujo desempenho dos modelos ainda seja considerado aceitável, isto é, fique acima de 90% de acurácia.

Em uma avaliação empírica utilizando um *dataset* com 36.915 amostras (19000 benignas e 17915 malignas) e 9000 características e o conjunto $\{10, 25, 50, 100, 200, 300, 500, 1000, 3000, 5000, 7000, 9000\}$ de valores para N [Moutaz, 2020], os autores demonstraram que o modelo foi capaz de manter uma acurácia 98,1% com apenas 500 características, o que representa uma redução de 94,44% no tamanho original do *dataset*.

3.4. ALR

A método proposto por [Şahin et al., 2021], aqui denominado de ALR, é baseado em uma adaptação da técnica clássica de regressão linear, um técnica estatística utilizada para modelar a relação entre duas ou mais variáveis [Montgomery et al., 2021]. A regressão linear, no escopo de detecção de *malwares*, pode ser utilizada para remover características com baixa relevância nos *datasets* [Şahin et al., 2021]. Por exemplo, através da regressão linear, podemos identificar coeficientes iguais ou próximos de zero para as características de *datasets*, que podem ser caracterizados como uma matriz esparsa. Esses são aqueles em que há uma predominância de valores 0 para um grande número de características de diversas amostras.

Na regressão linear, o impacto das variáveis independentes na predição da variável dependente é determinado pelo cálculo dos coeficientes correspondentes a cada variável independente. Quando o modelo de regressão linear é criado sobre os dados obtidos, os coeficientes assumem tipicamente valores entre -1 e 1. Contudo, se o conjunto de dados processado é majoritariamente uma matriz esparsa, compostas por muitos valores zero, isto é, características não presentes nas amostras, isso faz com que os coeficientes das permissões sejam zero ou próximos de zero. No processo de seleção de características baseado em regressão linear eliminamos aquelas com coeficientes próximos de zero.

Para demonstrar o funcionamento do ALR, os autores utilizaram um conjunto de dados composto por 2000 aplicativos, sendo 1000 maliciosos e 1000 benignos, e contendo 102 permissões por amostra, que correspondem às variáveis independentes do método proposto. Removendo as permissões cujo coeficientes estivessem no intervalo -0.1 e 0.1, a utilização do método permitiu reduzir o número de caracteristicas para apenas 27 permissões, o que representa uma redução de mais de 73%.

3.5. Quadro Resumo

Na Tabela 1 apresentamos um quadro resumo sobre os métodos de seleção de características.

Tabela 1. Informações dos Artigos originários dos Métodos

Método	Caracters.	Datasets	Amostras	Técnicas de Seleção	Redução (%)
SigAPI	Chamadas de API	Próprio (*)	9470B+5560M, 2539B+1200M	Mutual Information Gain, Univariate ROC-AUC Score, Recursive Feature Elimination with Gradient Boosting Classifier, Recursive Feature Elimination with Random Forest Classifier, Select KBest with Chi2, SelectFromModel (Tree-based), Correlação	65,8 - 79,46, 63,77 - 78,28
SigPID	Permissões	Próprio (*)	5494B+1661M	Permission Ranking with Negative Rate, Support Based Permission Ranking, Permission Mining with Association Rules	83,70
RFG	Chamadas de API	Próprio (*)	19000B+17915M	Chi2, ANOVA e algoritmos de aprendizado de máquina	94,44
ALR	Permissões	Android Malware Dataset	1000B+1000M	Regressão Linear	73,53

Como podemos observar, três dos quatro trabalhos avaliam o método utilizando

um *dataset próprio* (*), geralmente criado a partir de outros conjuntos dados, como o Drebin e o Androcrawl, e outras fontes de aplicativos, como Google Play e APKPure.

Outro ponto interessante é que ao observarmos as técnicas estatísticas clássicas de seleção empregadas pelos métodos e os resultados de redução, podemos perceber uma divergência entre os métodos. Essas diferenças, aliadas ao fato de cada trabalho utilizar um *dataset* distinto para avaliar o método, tornam os resultados não comparáveis. Neste trabalho, nosso objetivo é estabelecer uma primeira base de comparação, utilizando os mesmos três conjuntos de dados para todos os quatro métodos.

4. Metodologia

Nesta seção apresentamos o detalhamento dos *datasets* utilizados no experimento, a configuração do experimento, as métricas e o ambiente de execução.

4.1. Datasets

Para avaliar e comparar os quatro métodos de seleção de características utilizamos três *datasets* publicamente disponíveis, o Drebin_215, o AndroCrawl e o MD46K, o que é essencial para a reprodutibilidade do trabalho. É importante ressaltar que o MD46K foi criado pelos autores. Para a construção do *dataset* MD46K, foram selecionados e baixados 46K aplicativos, datados a partir de 2018, do AndroZoo[2]. Cada APK foi rotulado utilizando a API do serviço online do VirusTotal[3], que disponibiliza mais de 60 *scanners* para analisar e rotular um aplicativo entre benigno ou maligno. Na etapa seguinte, foram extraídas as características estáticas de cada APK utilizando a ferramenta AndroGuard [Pontes et al., 2021]. Finalmente, os dados de rotulação e extração foram utilizados para a construção do *dataset*. Os *datasets*, incluindo detalhes adicionais sobre os aplicativos selecionados e as etapas de contrução do *dataset* MD46K, as implementações dos métodos e a configuração do experimento estão disponíveis publicamente no repositório desse estudo.

De cada um dos três *datasets*, selecionamos apenas as categorias de características consideradas nos trabalhos originais dos quatro métodos de seleção de características avaliados, ou seja, permissões e chamadas de API. Nosso objetivo foi eliminar características contidas nesses *datasets* que originalmente não foram consideradas pelos métodos de seleção, como intenções e *opcodes*. É importante destacar que as permissões e chamadas de API estão entre as características consideradas mais relevantes para detecção de *malwares* Android [Wang et al., 2019].

A Tabela 2 apresenta os detalhes dos *datasets* derivados e utilizados nos experimentos. Como pode ser observado, é apresentado o tipo e o número de características, bem como o número de amostras de cada *dataset*. Adicionalmente, a tabela apresenta também a proporção entre amostras benignas e malignas (**B:M**).

Além do agrupamento das características utilizadas nos experimentos, os *datasets* passaram também por uma limpeza, ou pré-processamento, uma etapa importante para a utilização do *dataset* em modelos de aprendizado de máquina. O processo de limpeza dos *datasets* incluiu [Han and Kamber, 2006]:

[2]`https://androzoo.uni.lu`
[3]`https://www.virustotal.com`

Tabela 2. Datasets

	Características			
Dataset	**Tipo**	**Número**	**Amostras**	**B:M**
md46k_permissions	Permissões	1.316	46.670	15,4:1
md46k_api_calls	Chamadas de API	1.524		
androcrawl_permissions	Permissões	49	96.732	8,5:1
androcrawl_api_calls	Chamadas de API	24		
drebin_215_permissions	Permissões	113	15.031	1,7:1
drebin_215_api_calls	Chamadas de API	73		

1. Remoção de amostras com um ou mais valores faltantes;
2. Remoção de amostras duplicadas, isto é, amostras contidas repetidas vezes no *dataset*;
3. Remoção de características que possuem o mesmo valor em todas as amostras (e.g., apenas 0s ou 1s);
4. Conversão dos tipos de dados para o tipo numérico inteiro.

4.2. Experimento

A Figura 1 ilustra o processo do experimento. Os *datasets*, apresentados na Seção 4.1, são utilizados como entradas para cada um dos quatro métodos. Cada método, para cada *dataset* de entrada, irá gerar um novo conjunto de dados contendo apenas as características selecionadas. Esses subconjuntos novos são então utilizados nos modelos de aprendizado de máquina para averiguação das métricas resultantes.

Na etapa final do experimento, todos os *datasets* derivados são utilizados em modelos de aprendizado de máquina baseados nos algoritmos *Random Forest* (RF) e *Support Vector Machine* (SVM). A escolha do RF e SVM é pelo fato de eles serem considerados os modelos mais frequentemente utilizados no domínio de detecção de *malwares* Android [Sharma and Rattan, 2021].

Com os resultados das métricas dos dois modelos, podemos então iniciar a análise para determinar a qualidade dos métodos de seleção de características com base em conjuntos de dados de entrada distintos. E, comparativamente, observar as diferenças das métricas resultantes em relação as dos conjuntos de dados originais dos *datasets*, isto é, sem redução de dimensionalidade.

4.3. Métricas

A discussão dos resultados dos modelos é construída em torno da curva ROC_AUC, que é uma métrica capaz de indicar o desempenho de um modelo ao considerar os aspectos de especificidade e sensibilidade, isto é, taxa de falso positivo e taxa de verdadeiro positivo.

4.4. Ambiente

A execução dos experimentos foi realizada em computadores com processador Intel Core i7-9700 CPU @ 3GHz (com 8 núcleos) e 16GB de memória RAM. O sistema operacional utilizado foi o Linux Debian GNU/Linux 11 Kernel 5.10.0-14-amd64.

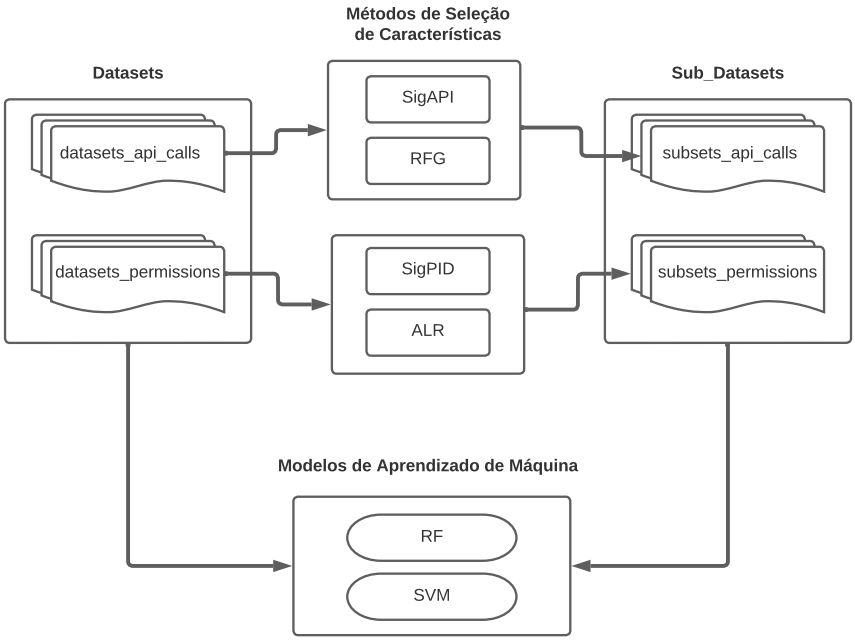

Figura 1. Etapas do Experimento

5. Resultados e Discussão

Nesta seção discutimos os resultados da aplicação dos métodos de seleção sobre os *datasets* de chamadas de API e permissões. Utilizamos os modelos classificadores *Random Forest* (RF) e *Support Vector Machine* (SVM) para verificar a qualidade dos resultados das reduções de características realizadas pelos métodos. Para simplificar a apresentação e discussão dos resultados, como o modelo RF resultou métricas (e.g., precisão, acurácia, *recall*, *f1_score* e ROC_AUC) iguais ou ligeiramente superiores ao SVM na maioria dos casos, iremos omitir as métricas do SVM por questão de simplicidade. Nos raros casos onde o SVM superou o RF, a diferença é insignificante, isto é, menos de 0,2%. Por exemplo, para o *androcrawl_permissions*, sem redução, o SVM atingiu 49,92% de ROC_AUC, enquanto que o RF atingiu 49,84%.

Nas Tabelas 3 e 4, as colunas de **redução** indicam (em porcentagem) o quanto o método foi capaz de reduzir o tamanho do *dataset* a partir da seleção de características. As colunas de **tempo** (em segundos) informam a média do tempo de execução do método de seleção. Finalmente, as colunas de **métrica** informam o resultado da métrica ROC_AUC para os modelos RF construídos a partir do *dataset* original e dos reduzidos (i.e., saída dos métodos de seleção).

5.1. Seleção de Chamadas de API

Na Tabela 3, sumarizamos os resultados dos métodos de seleção de características SigAPI e RFG. Como podemos observar, o SigAPI apresenta uma porcentagem de redução superior ao RFG para os três *datasets*, chegando a 99,15% de redução para o conjunto de dados do *md46k_api_calls*. Isso pode ser explicado pelo método agressivo de redução de características do SigAPI, que utiliza e avalia sete técnicas distintas, procurando identificar aquela que gera a maior redução. Para alguns *datasets*, como o *dre-*

bin_215_api_calls, que contém um conjunto reduzido de apenas 73 chamadas de API sensíveis e de maior relevância, o método funciona muito bem e apresenta uma boa métrica ROC_AUC (94,40%). Entretanto, para os outros dois *datasets*, contendo um número maior e mais desbalanceado de amostras, os resultados qualitativos foram inferiores (e.g., apenas 70,23% para o *md46k_api_calls*).

Tabela 3. Resultados do SigAPI e RFG

	Redução (%)		Tempo (s)		Métrica (ROC_AUC)		
Dataset	**SigAPI**	**RFG**	**SigAPI**	**RFG**	**SigAPI**	**RFG**	**Sem redução**
drebin_215_api_calls	79,45	71,23	540	203	94,40	91,71	97,55
md46k_api_calls	99,15	8,07	180803	4322	70,23	79,95	79,95
androcrawl_api_calls	95,83	12,50	314	391	81,56	90,94	91,18

Enquanto que para o *dataset md46k_api_calls* o resultado da ROC_AUC do SigAPI foi de 70,23%, para o *androcrawl_api_calls* foi melhor, ficando em 81,56%. Além de serem os dois *datasets* mais desbalanceados em termos de amostras, o *androcrawl_api_calls* contém um número reduzido de chamadas de API sensíveis, o que explica os resultados melhores. Entretanto, o *md46k_api_calls* contém um grande número de chamadas de API, sendo a maioria delas de pouca relevância para a detecção. Na prática, este é o tipo de *dataset* mais realista, ou seja, que apresenta todas as características do processo de extração para uma determinada categoria, sem etapas de filtragem prévias, pois é tarefa do cientísta de dados selecionar os dados mais qualitativos e representativos durante o *pipeline* de construção dos modelos de aprendizado de máquina, mais especificamente na etapa de engenharia de características. Em síntese, os dados indicam que o SigAPI apresenta um desempenho instável, que varia significativamente de *dataset* para *dataset* (e.g., vai de 70,23% a 94,40% para apenas três conjuntos de dados). Esses números nos sugerem que escolher o melhor método de seleção, levando com consideração os respectivos conjuntos de dados, é de fundamental importância.

Diferentemente do SigAPI, o RFG apresenta uma estabilidade maior, isto é, menor variância de métricas entre diferentes *datasets*, com resultados ROC_AUC superiores na maioria dos casos. Essa maior estabilidade tem a ver com etapas do método. Numa primeira etapa, o RFG ranqueia as características por ordem de importância. Num segundo estágio, o RFG utiliza modelos preditivos para identificar qualitativamente os melhores resultados para diferentes subconjuntos de caracterísiticas de maior relevância (e.g., 20, 40, 60 melhores). O método procura identificar o menor grupo de características mais importantes que potencializa os melhores resultados em modelos de aprendizado de máquina, isto é, aqueles que atingem as melhores métricas no que diz respeito à acurácia, *recall*, precisão e *f-measure*. Se os resultados das métricas ficarem abaixo de 90%, é selecionado o maior valor. Do contrário, é seleciono o valor inferior que ultrapassa a marca de 90%. Resumidamente, as técnicas implementadas pelo RFG conferem uma maior estabilidade em termos de qualidade dos resultados para *datasets* bem diversos. Portanto, podemos dizer que o RFG é um método mais recomendado para usuários com pouca experiência em ciência de dados, uma vez que irá produzir bons resultados para a maioria dos *datasets*.

Com relação ao tempo execução, podemos observar que o SigAPI consome mais recursos computacionais do que o RFG. O consumo de recursos do SigAPI é devido a quantidade e complexidade das sete técnicas implementadas. No SigAPI, o tempo de execução pode ser caracterizado como uma função exponencial da dimensionalidade (i.e., número de características) do *dataset*, o que pode ser observada na Tabela 3. Enquanto o método levou 540 segundos para o *drebin_215_api_calls* (15.031 amostras e 73 características), para o *md46k_api_calls*, que contém 46.670 amostras e 1.524 características, levou 180.803 segundos. Ou seja, no SigAPI, quanto mais características compõem um *dataset*, maior será o tempo da seleção. A quantidade e a complexidade das etapas do método, que buscam maximizar a capacidade de redução, poderá levar a penalidades computacionais.

Finalmente, é importante ressaltar que os melhores resultados foram para os *datasets* sem nenhuma redução, o que é o esperado, uma vez que o principal objetivo dos métodos de seleção de características é reduzir a dimensionalidade dos conjuntos de dados, o que pode ocasionar perdas. Em alguns casos, como o SigAPI para o *drebin_215_api_calls*, conseguimos ter uma redução substancial na dimensionalidade do *datasets* (praticamente 80%), com uma perda relativamente pequena (menos de 4%) na qualidade final das métricas do modelo. Esse é um *trade-off* que precisa ser considerado e analisado pelos desenvolvedores dos modelos de classificação.

5.2. Seleção de Permissões

Os resultados dos métodos de seleção de características SigPID e ALR são apresentados na Tabela 4. Como pode ser observado, o SigPID atinge uma redução de 71,68% para o *dataset drebin_215_permissions* e uma qualidade final do modelo RF de 94,05% ROC_AUC, muito próximo do modelo sem redução (95,86%), o que pode ser considerado um resultado excelente para um método de seleção de características. É importante destacarmos que o SigPID leva em consideração o balanceamento do *dataset* durante o processo de redução, pois ele trata as classes de amostras (i.e., aplicativos malignos e benignos) de maneira separada. Apesar de isto levar a bons resultados para *datasets* como o *drebin_215_permissions*, deixa a desejar para *dataset* desbalanceados.

Tabela 4. Resultados SigPID e ALR

	Redução (%)		**Tempo (s)**		**Métrica (ROC_AUC)**		
Dataset	**SigPID**	**ALR**	**SigPID**	**ALR**	**SigPID**	**ALR**	**Sem redução**
drebin_215_permissions	71,68	69,03	48	2	94,05	89,18	95,86
md46k_permissions	99,77	46,05	7.106	746	50,00	52,81	62,06
androcrawl_permissions	87,76	97,96	176	3	50,00	50,00	49,84

Nos *datasets md46k_permissions* e *androcrawl_permissions* observamos um comportamento diferente. Para o *md46k_permissions*, o SigPID apresenta uma redução agressiva (99,77%), porém sem capacidade preditiva (apenas 50%) e abaixo do ALR. Com o *dataset* completo, o modelo RF consegue atingir uma ROC_AUC de 62,06%, ou seja, superior aos resultados dos subconjuntos de características resultantes dos métodos de seleção. A explicação para os baixos desempenhos do modelo RF, tanto do SigPID quanto do ALR está relacionado ao alto desbalanceamento desses conjuntos de dados e o fato de

serem *datasets* com características dispostas como uma matriz esparsa de dados. Como consequência, ambos os métodos acabam realizando reduções pouco seletivas em dados desbalanceados e esparsos. Mais uma vez, podemos constatar que há uma relação não desprezível entre os *datasets* de entrada e os resultados dos métodos de seleção. O simples fato de escolher e aplicar um método com várias etapas bem definidas e complexas, mesmo que para um tipo específico de características, não significa que irá gerar bons resultados para os modelos de classificação.

Com relação ao método ALR, podemos observar uma relação direta entre a quantidade de amostras e a sua capacidade de redução. Quanto mais amostras no *dataset*, mais o ALR consegue entender as informações acerca das características e tende a realizar reduções maiores, mas não necessariamente mais qualitativas. Adicionalmente, quanto menos características um *dataset* tem, menores são os valores das correlações que o ALR consegue realizar, o que indica mais eliminações. Por exemplo, para o *androcrawl_permissions*, que contém o menos número de características entre os *datasets*, o ALR realizou uma alta redução. Entretanto, o resultado não gerou um bom modelo preditivo de qualidade.

Em termos de tempo de computação, o SigPID exige várias vezes mais recursos que o ALR. Isto é explicado pelo fato de o método implementar três níveis de corte, utilizando mineração com regras de associação e um modelo SVM. Isto torna a execução do método até cinquenta vezes mais lento que o ALR.

6. Conclusão

Neste trabalho apresentamos uma avaliação dos métodos de seleção de características SigPID, SigAPI, ALR e RFG, para características dos tipos permissões e chamadas de API, utilizando três *datasets* distintos. Os resultados sugerem que os métodos, mesmo tendo sido projetos e especializados para apenas um tipo de característica, não são gerais o suficiente a ponto de identificar padrões em diferentes *datasets*, isto é, conjuntos de dados com diferentes dimensões, densidade, balanceamento e diversidade de características. Enquanto métodos como o SigPID produzem resultados bons para *datasets* similares ao *drebin_215_permissions*, atingindo pontuação ROC_AUC acima de 90%, produzem resultados que deixam a desejar para conjuntos de dados similares ao *md46k_permissions*, onde o método consegue atingir uma ROC_AUC de apenas 50,00%.

Dentre os métodos avaliados, o RFG foi o único a apresentar resultados mais consistentes entre os diferentes *datasets*. O método consegue generalizar a seleção de chamadas de API para diferentes conjuntos de dados através da combinação de duas etapas robustas, uma de ranqueamento de importância das características e uma segunda, baseada em modelos de aprendizado de máquina, para identificar o subconjunto de características que leva aos melhores resultados de classificação. Entretanto, métodos similarmente elaborados e complexos, como o SigAPI, não conseguem generalizar a seleção para *datasets* diversos. Como lição, podemos tomar a necessidade de avaliarmos extensivamente os métodos de seleção de características para conjuntos de dados diversos, observando em profundidade o seu compartamento.

Como trabalhos futuros podemos mencionar: (a) avaliar os métodos de seleção de características com uma gama ainda maior e mais diversificada de *datasets*, em particular conjuntos de dados maiores e mais realistas, como aqueles sem qualquer pré-seleção de

características dos aplicativos; (b) avaliar extensivamente outros métodos de seleção de características; (c) identificar parâmetros de otimização que podem impactar os métodos de seleção; (d) avaliar resultados de detecção de *malwares* obtidos a partir da combinação de diferentes métodos de seleção.

Agradecimentos

Agradecemos ao Lucas Vilanova pela contribuição na construção do *dataset*. Esta pesquisa foi parcialmente financiada, conforme previsto nos Arts. 21 e 22 do decreto no. 10.521/2020, nos termos da Lei Federal no. 8.387/1991, através do convênio no. 003/2021, firmado entre ICOMP/UFAM, Flextronics da Amazônia Ltda e Motorola Mobility Comércio de Produtos Eletrônicos Ltda. O presente trabalho foi realizado também com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Referências

Agrawal, P. and Trivedi, B. (2021). Machine learning classifiers for android malware detection. In *Data Management, Analytics and Innovation*, pages 311–322. Springer.

Anggraeni, A., Mustofa, K., and Priyanta, S. (2021). Comparison of filter and wrapper based feature selection methods on spam comment classification. *IJCCS*, 15(3):245.

Cai, L., Li, Y., and Xiong, Z. (2021). JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. *Computers & Security*, 100:102086.

Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T., and Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *JCVHT*, 13.

Galib, A. H. and Hossain, B. M. M. (2020). Significant API calls in Android malware detection (using feature selection techniques and correlation based feature elimination). In García-Castro, R., editor, *The 32nd SEKE*, pages 566–571. KSI Research Inc.

Golrang, A., Yayilgan, S. Y., and Elezaj, O. (2021). The multi-objective feature selection in Android malware detection system. In *Intelligent Tech. and Applications*, page 311.

Han, J. and Kamber, M. (2006). Data mining: concepts and techniques, 2nd. *University of Illinois at Urbana Champaign: Morgan Kaufmann*.

Kouliaridis, V. and Kambourakis, G. (2021). A comprehensive survey on machine learning techniques for Android malware detection. *Information*, 12(5).

Lee, J., Jang, H., Ha, S., and Yoon, Y. (2021). Android malware detection using machine learning with feature selection based on the genetic algorithm. *Mathematics*, 9(21).

Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., and Liu, H. (2020). A review of Android malware detection approaches based on machine learning. *IEEE Access*, 8:124579.

Mahindru, A. and Sangal, A. L. (2021). SemiDroid: a behavioral malware detector based on unsupervised machine learning techniques using feature selection approaches. *International Journal of Machine Learning and Cybernetics*, 12:1411.

Mas' ud, M. Z., Sahib, S., Abdollah, M. F., Selamat, S. R., and Huoy, C. Y. (2017). A comparative study on feature selection method for N-gram mobile malware detection. *Int. J. Netw. Secur.*, 19(5):727–733.

Montgomery, D. C., Peck, E. A., and Vining, G. G. (2021). *Introduction to linear regression analysis*. John Wiley & Sons.

Moutaz, A. (2020). Automated malware detection in mobile app stores based on robust feature generation. *Electronics*, 9:435.

Nivaashini, M., Soundariya, R. S., Vidhya Shri, H., and Thangaraj, P. (2018). Comparative analysis of feature selection methods and machine learning algorithms in permission based Android malware detection. In *I2C2SW*, pages 72–77.

Pontes, J., Costa, E., Rocha, V., Neves, N., Feitosa, E., Assolin, J., and Kreutz, D. (2021). Ferramentas de extração de características para análise estática de aplicativos android. In *VI WRSeg*.

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 Comput. Surv.*, 53(6).

Roy, S., DeLoach, J., Li, Y., Herndon, N., Caragea, D., Ou, X., Ranganath, V. P., Li, H., and Guevara, N. (2015). Experimental study with real-world data for Android app security analysis using machine learning. In *31st ACSAC*, page 81–90. ACM.

Şahin, D. Ö., 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*, pages 1–16.

Salah, A., Shalabi, E., and Khedr, W. (2020). A lightweight Android malware classifier using novel feature selection methods. *Symmetry*, 12(5).

Sharma, T. and Rattan, D. (2021). Malicious application detection in android—a systematic literature review. *Computer Science Review*, 40:100373.

Smmarwar, S. K., Gupta, G. P., and Kumar, S. (2022). A hybrid feature selection approach-based Android malware detection framework using machine learning techniques. In *Cyber Security, Privacy and Networking*, pages 347–356. Springer.

Sun, L., Li, Z., Yan, Q., Srisa-an, W., and Pan, Y. (2016). SigPID: significant permission identification for android malware detection. In *11th MALWARE*, pages 1–8.

Venkatesh, B. and Anuradha, J. (2019). A review of feature selection and its methods. *Cybernetics and Information Technologies*, 19(1):3–26.

Wang, L., Gao, Y., Gao, S., and Yong, X. (2021). A new feature selection method based on a self-variant genetic algorithm applied to Android malware detection. *Sym.*, 13(7).

Wang, W., Zhao, M., Gao, Z., Xu, G., Xian, H., Li, Y., and Zhang, X. (2019). Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions. *IEEE Access*, 7:67602–67631.

Zhou, Y. and Jiang, X. (2012). Dissecting Android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy*, pages 95–109.

Şahin, D., Kural, O., Akleylek, S., and Kilic, E. (2021a). A novel Android malware detection system: adaption of filter-based feature selection methods. *Journal of Ambient Intelligence and Humanized Computing*.

Şahin, D., Kural, O., Akleylek, S., and Kilic, E. (2021b). A novel permission-based Android malware detection system using feature selection based on linear regression. *Neural Computing and Applications*, pages 1–16.