Detecção de Malwares Android: reprodução da seleção de características do SigPID (versão estendida)*

Joner Assolin[1], Vanderson Rocha[3], Guilherme Silveira[1], Gustavo Rodrigues[1,2], Eduardo Feitosa[3], Karina Casola[1], Diego Kreutz[1]

[1]Laboratório de Estudos Avançados em Computação (LEA)
Programa de Pós-Graduação em Engenharia de Software (PPGES)
Universidade Federal do Pampa (Unipampa)

[2]Combate à Fraude

[3]Grupo de Pesquisa em Tecnologias Emergentes e Segurança de Sistemas (ETSS)
Instituto de Computação (IComp)
Universidade Federal do Amazonas (UFAM)

{NomeSobrenome}.aluno@unipampa.edu.br
gustavo.rodrigues@combateafraude.com
{vanderson,efeitosa}@icomp.ufam.edu.br, kreutz@unipampa.edu.br

Resumo. *Para atacar o desafio de escalabilidade na detecção de malwares em Android, há trabalhos que propõem a utilização de um número reduzido de permissões, como é o caso do SigPID. Neste trabalho, apresentamos a reprodução dos 3 níveis de seleção de permissões e avaliação dos principais métodos de aprendizagem do SigPID, utilizando um conjunto de dados publicamente disponível. Nosso estudo inicial indica que o número de permissões impacta o tempo de treinamento e execução, bem como a acurácia dos modelos. Entretanto, o tempo de execução pode não ser significativo a ponto de justificar um número menor de permissões para detecção de malwares.*

1. Introdução

Dentre os métodos para detecção de *malwares* em aplicativos Android, os que utilizam aprendizado de máquina vêm ganhando destaque [Wu et al., 2021]. Independente de focarem suas abordagens na análise estática, dinâmica ou híbrida, esses trabalhos utilizam as permissões do Android para o desenvolvimento de modelos de detecção de *malwares* [Alsoghyer and Almomani, 2020]. Entretanto, utilizar todas as 247 permissões das APIs de Android disponíveis, para o treinamento dos modelos de aprendizado de máquina, pode representar um desafio de escalabilidade [Yildiz and Doğru, 2019, Li et al., 2018] e impactar negativamente no tempo de execução das soluções.

Visando mitigar o problema da escalabilidade, há trabalhos (e.g., [Li et al., 2018, Yildiz and Doğru, 2019]) que investigaram o impacto da redução dos números de permissões utilizadas para o treino dos modelos. Como resultado, verificaram que, mesmo utilizando um número menor de permissões, o tempo de execução pode ser reduzido sem comprometer de forma significativa a acurácia do modelo (i.e., melhor escalabilidade sem comprometer o desempenho da classificação).

*Este artigo é uma versão estendida do paper [Assolin et al., 2021], de 6 páginas, originalmente publicado no WRSeg 2021 e convidado para publicação na ReABTIC.

Neste trabalho avaliamos e discutimos a reprodutibilidade e o desempenho do trabalho de [Li et al., 2018, Sun et al., 2016] (SigPID — *Significant Permission Identification for Android Malware Detection*), que pode ser considerado um dos mais relevantes e mais bem citados (mais de 320 citações *Google Scholar Citations* — GSC, em setembro de 2021) sobre escalabilidade de modelos de detecção de *malwares* Android. Como o conjunto de dados empregado no trabalho original não está disponível, utilizamos um conjunto de dados público contendo 113 permissões. Numa primeira etapa, reproduzimos a redução de características utilizada no SigPID, que consiste em três níveis seleção de permissões. A aplicação desses níveis de seleção, no conjunto de dados utilizado, resultou em 27 permissões mais significativas. Em seguida, avaliamos o desempenho do modelo em comparação com diferentes conjuntos de permissões, a saber: (a) 113 permissões (*baseline*) contidas no conjunto de dados utilizado; (b) 22 permissões identificadas no trabalho original do SigPID; (c) 32 permissões mais recorrentes em trabalhos de detecção de *malwares* Android; e (d) 22 permissões classificadas como perigosas pela Google[1].

Como contribuição do trabalho, podemos destacar: (a) a implementação da estratégia de seleção de características empregada pelo SigPID em um conjunto de dados público; (b) a identificação de um subconjunto essencial de permissões (permissões significativas) que pode ser utilizado para identificar efetivamente *malwares* no Android; (c) um comparativo com o trabalho original, que identifica 22 permissões como significativas, e também a comparação com outros conjuntos de dados com diferentes quantidades de permissões; e (d) a análise de aspectos de reprodutibilidade do trabalho original.

O trabalho está organizado como segue. Na Seção 2 apresentamos os requisitos para a reprodução do SigPID e detalhamos a metodologia de seleção das características. Nas Seções 3 e 4 apresentamos os resultados e as considerações finais.

2. Reprodução dos Experimentos

2.1. Detalhamento do Ambiente

Para o desenvolvimento e avaliação dos experimentos, utilizamos um notebook com processador Intel Celeron 1007U (1.5GHz, Dual Core, 2MB L2), 4GB DDR3 1.600MHz, disco rígido de 320GB (SATA - 5.400rpm), Windows 10 Home Single Language, compilação 19042.1110. Para a implementação e avaliação dos modelos, utilizamos as ferramentas Jupyter Notebook (IPython 7.12.0, Python 3.7.6 (default, jan. 8 2020) e o Google Chrome Versão 91.0.4472.124 (Versão oficial) 64 bits. Com exceção do algoritmo *Functional Tree*, versão 1.0.4, implementado com a ferramenta Weka versão 3.9.5, os demais foram implementados utilizando a versão 0.22.1 da biblioteca Scikit-learn.

Para análise e uso do conjunto de dados, utilizamos uma divisão estratificada pseudo-aleatória (*test_size*) de 70%/30% [James et al., 2013], a partir dos dados iniciais, sendo 70% utilizado para treinos e 30% para testes. As divisões estratificadas são desejáveis em casos de conjuntos de dados desbalanceados, como é o caso do escolhido. Para garantir a reprodutibilidade do experimento, definimos arbitrariamente a semente aleatória como 1 para `train_test_split`, de forma a controlar a seleção dos dados de treino e teste. Já os hiperparâmetros, variáveis que controlam o próprio processo de treinamento, foram seguidos conforme o padrão da biblioteca Scikit-learn.

[1]Ao total, a Google define 30 permissões como perigosas. Destas, 22 estão presentes no conjunto de dados analisado.

2.2. Conjunto de Dados

Como mencionado na introdução, o conjunto de dados original do trabalho está indisponível. Para a reprodução e comparação do SigPID com diferentes conjuntos de permissões, selecionamos o conjunto de dados Drebin_215, disponível publicamente no FigShare[2], um sub conjunto do *Drebin project* [Arp et al., 2014]. A escolha desse *dataset* se deve ao fato de ter de acesso público e possuir permissões do Android como características. O Drebin_215 possui 215 características extraídas de 15.036 aplicativos (5.560 malignos e 9.476 benignos), sendo que 113 características são permissões.

2.3. SigPID: Seleção das Permissões

Resumidamente, o trabalho do SigPID [Li et al., 2018] propõe o MLDP (*Multi-Level Data Pruning*), isto é, um método de seleção de permissões. O MLDP é composto por três níveis progressivos de seleção de características, cujo objetivo é identificar as permissões mais relevantes para uso na construção do modelo de detecção de *malwares* Android. A ideia por trás do método é diminuir o número de permissões (selecionando as mais significativas) e, consequentemente, o tempo de execução dos modelos. O MLDP assume, como parâmetro de seleção, uma taxa de detecção de *malware* acurácia e precisão de no mínimo 90%, sendo essa considerada uma taxa muito boa.

O método opera nos seguintes níveis de seleção: (1) classificação de permissão com taxa negativa (*Permission Ranking Negative Rate* ou PRNR), onde o objetivo é selecionar as permissões que geram as maiores métricas (*e.g.*, acurácia, recall, F1-score); (2) classificação de permissão baseada em suporte (*Support Based Permission Ranking* ou SPR), onde o objetivo é selecionar o menor número de permissões que alcançam uma taxa maior que 90% de acurácia; e (3) mineração de permissões com regras de associação (*Permission Mining with Association Rules* ou PMAR), para eliminar permissões que geram redundância ao modelo. Cada um dos três níveis é detalhado na versão estendida do trabalho, disponível em [Assolin et al., 2021]. Para reprodutibilidade, os conjuntos de dados e códigos estão disponíveis online no GitHub[3]. Ao final, chegamos a seleção de 108, 30 e 27 permissões nos níveis 1, 2 e 3 de seleção do MLDP, respectivamente.

2.4. Nível 1 (PRNR): classificação de permissão com taxa negativa

A PRNR opera em duas matrizes, uma de permissões utilizadas por amostras de *malwares* e outra utilizada por aplicativos benignos, onde cada linha corresponde a um aplicativo e cada coluna a uma permissão. O objetivo é remover as permissões que são frequentemente solicitadas tanto por aplicativos maliciosos quanto por benignos (*e.g.*, INTERNET).

Como o número de aplicativos benignos tende a ser maior que o de *malwares* em um conjunto de dados, o PRNR do SigPID propõe a Equação 1 para equilibrar às duas matrizes. A equação calcula o suporte de cada permissão no conjunto de dados maior e, em seguida, dimensiona proporcionalmente o suporte para corresponder ao conjunto de dados menor. O suporte é a frequência com que cada permissão aparece no conjunto de dados.

[2] `https://figshare.com/articles/dataset/Android_malware\_dataset_for_machine\_learning\_2/5854653`

[3] `https://github.com/Malware-Hunter/SigPID`

Tabela 1. Lista de permissões selecionadas após aplicação do MLDP

PRNR+SPR+PMAR		
ACCESS_NETWORK_STATE	CHANGE_WIFI_STATE	WRITE_EXTERNAL_STORAGE
WRITE_SETTINGS	READ_PHONE_STATE	CAMERA
WAKE_LOCK	CALL_PHONE	RECEIVE_BOOT_COMPLETED
WRITE_CONTACTS	VIBRATE	READ_EXTERNAL_STORAGE
GET_ACCOUNTS	USE_CREDENTIALS	ACCESS_FINE_LOCATION
READ_HISTORY_BOOKMARKS	ACCESS_COARSE_LOCATION	CHANGE_NETWORK_STATE
SEND_SMS	RECORD_AUDIO	READ_CONTACTS
READ_SYNC_SETTINGS	RECEIVE_SMS	RESTART_PACKAGES
READ_SMS	BLUETOOTH	GET_TASKS

$$S_B(P_j) = \frac{\sum_i B_{ij}}{Size(B_j)} * Size(M_j) \tag{1}$$

Na equação, M representa a matriz de permissões dos aplicativos maliciosos e B dos benignos. (P_j) denota permissão e $S_B(P_j)$ representa o suporte da permissão na matriz B. A listagem 1 apresenta o código da implementação da Equação 1 utilizando a linguagem de programação Python.

Listagem 1. Implementação da Equação 1

```
def S_B(j):
    sigmaBij = B.sum(axis = 0, skipna = True)[j]
    sizeBj = B.shape[0]
    sizeMj = M.shape[0]
    return (sigmaBij/sizeBj)*sizeMj
```

A implementação da PRNR é baseada na Equação 2, que determina uma classificação para cada permissão, variando no intervalo [-1, 1]. Na equação, se $R(P_j)$ = 1, significa que a permissão (P_j) é apenas usada no conjunto de dados malicioso e que é uma permissão de alto risco. Se $R(P_j)$ = -1, significa que a permissão (P_j) só é usada no conjunto de dados benigno e é uma permissão de baixo risco. Por outro lado, se $R(P_j)$ = 0, significa que (P_j) tem muito pouco impacto na detecção de *malware*, pois aparece em ambos os conjuntos de dados.

$$R(P_j) = \frac{\sum_i M_{ij} - S_B(P_j)}{\sum_i M_{ij} + S_B(P_j)} \tag{2}$$

O processamento da equação do primeiro nível de seleção, PRNR, é ilustrado na Figura 1. O processamento foi implementado utilizando o código da listagem 2.

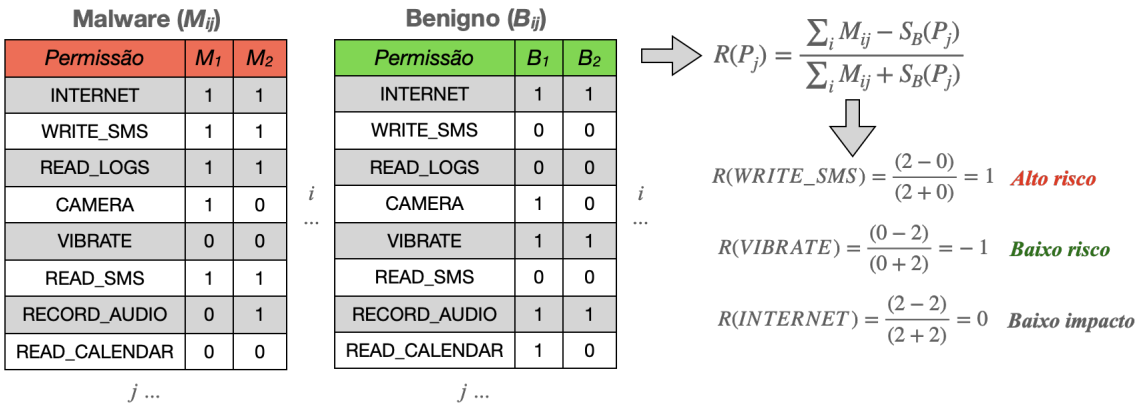

Figura 1. Exemplo do cálculo do ranking de cada permissão

Listagem 2. Implementação da Equação 2

```
def PRNR(j):
    sigmaMij = M.sum(axis = 0, skipna = True)[j]
    S_Bj = S_B(j)
    return (sigmaMij-S_Bj)/(sigmaMij+S_Bj)
```

O próximo passo é ordenar a lista dos valores obtidos pelo PRNR, associando-a pela ordem crescente aos aplicativos benignos e pela ordem decrescente aos *malwares*, conforme ilustrado na Figura 2. Dando continuidade ao processo de classificação, é utilizado o *Permission Incremental System* (PIS). As permissões são agrupadas 3 a 3, iniciando pelo topo de cada lista. A cada incremento de 6 permissões (3 benignas e 3 *malwares*), os grupos de permissões são submetidos ao algoritmo de aprendizado de máquina *Support Vector Machine* (SVM). A cada novo grupo, é avaliado o poder preditivo de detecção de *malware*, utilizando as métricas descritas na Tabela 2.

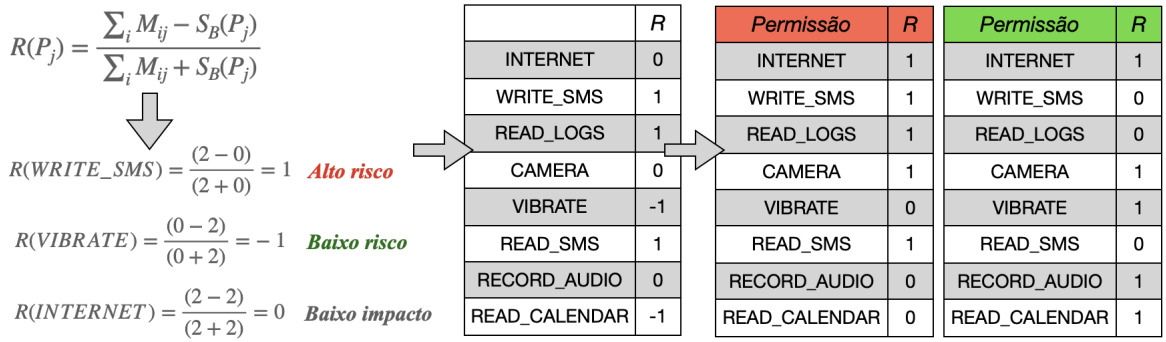

Figura 2. Ordenação pelo ranking

Tabela 2. Métricas

Acurácia = $\frac{TP+TN}{TP+TN+FP+FN}$	Precisão = $\frac{TP}{TP+FP}$
Recall = $\frac{TP}{TP+FN}$	F1_Score = $\frac{2*(Precisão*Recall)}{(Precisão+Recall)}$

O momento de seleção ocorre quando as métricas chegam ao seu valor máximo e, posteriormente, começam a decair. O objetivo do processo é encontrar o menor número de permissões que produza os melhores *scores* de detecção de *malware*. Como resultado do primeiro nível de seleção, chegamos a 108 permissões, de um total de 113 contidas no conjunto de dados. A Figura 3 ilustra o desempenho do PRNR para cada incremento (PIS).

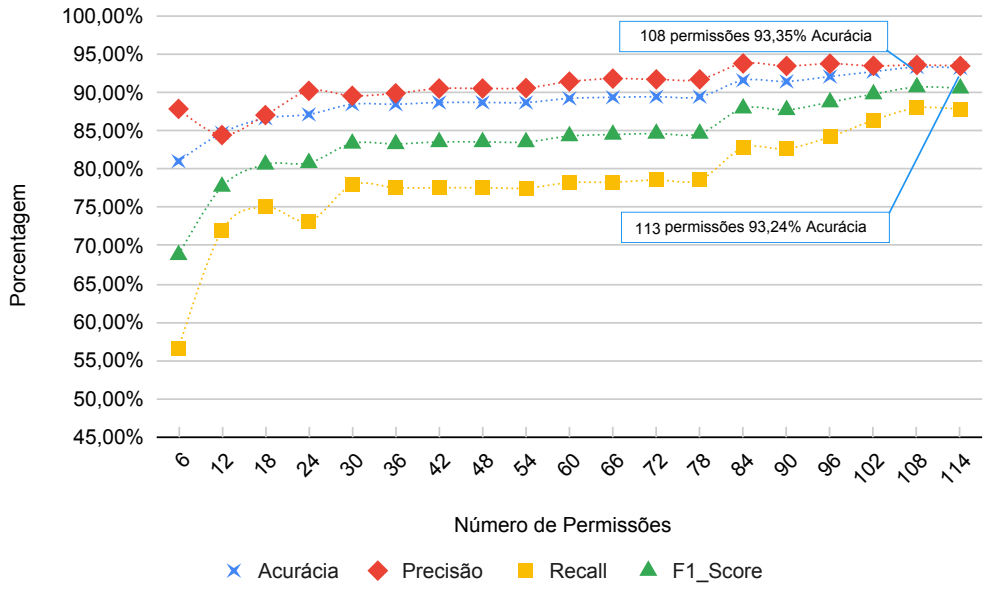

Figura 3. Desempenho do PRNR a cada incremento (PIS)

Podemos observar no gráfico da Figura 3 que ocorre uma pequena redução na acurácia, que passou de 93,35% (com 108 permissões) para 93,24% (com 113 permissões), ocorrendo algo similar com as demais métricas.

2.5. Nível 2 (SPR): classificação de permissão baseada em suporte

A classificação de permissão baseada em suporte busca avaliar a recorrência de uma permissão. Se ela possuir uma baixa frequência, seu impacto será mínimo no desempenho da detecção de *malware*. As permissões consideradas de baixo desempenho são excluídas. Assim, as 108 permissões selecionadas no passo anterior (PRNR) são ordenadas em ordem decrescente conforme o seu suporte.

O objetivo da SPR é encontrar o menor número de permissões de alto suporte capaz de produzir uma acurácia de detecção acima de 90%. Novamente é aplicado o incremento (PIS), porém, agora a cada 5 permissões. Quando o modelo atingir 90% de acurácia, selecionamos as permissões contidas no incremento. Com 30 permissões, foi possível satisfazer a condição e atingir 90,07% de acurácia, conforme podemos observar no gráfico da Figura 4.

2.6. Nível 3 (PMAR): mineração de permissões com regras de associação

A mineração de permissões com regras de associação inspeciona permissões que possuem maior probabilidade de estarem associadas (*e.g.*, WRITE_SMS e READ_SMS). Então, a

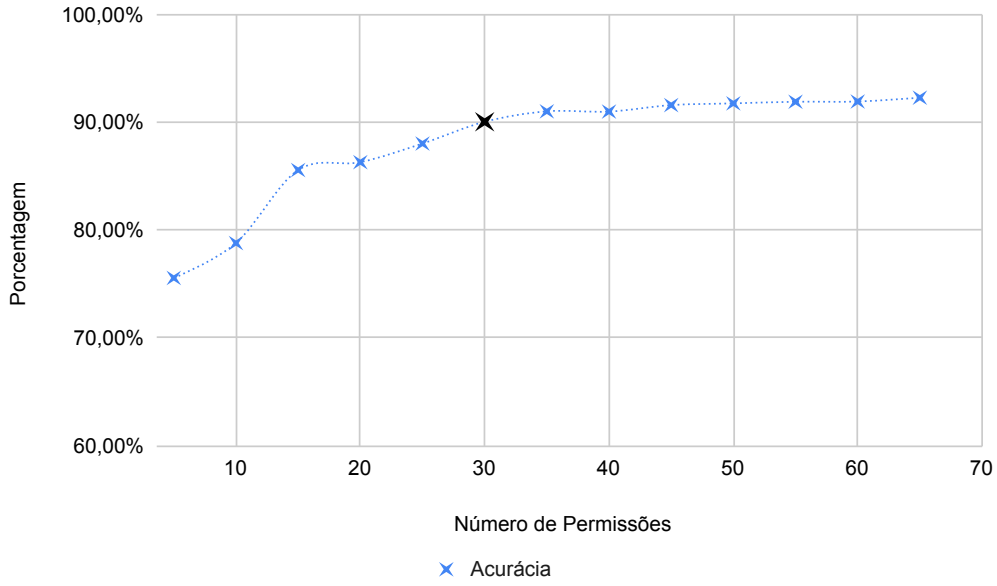

Figura 4. Desempenho da SPR para cada incremento (PIS)

permissão com o menor valor de PRNR (Equação 2) é eliminada.

Para identificar a relação de cada permissão, é aplicado o algoritmo Apriori [Agrawal et al., 1994] com os parâmetros de 96,5% de confiança mínima e 10% de suporte mínimo, os mesmos definidos no trabalho original do SigPID. O Apriori calcula a probabilidade de um item estar presente em um conjunto de itens frequentes, desde que outro item esteja presente.

A Tabela 3 representa a execução do Apriori sobre as 30 permissões selecionadas após as etapas PRNR e SPR. A confiança de uma regra indica a probabilidade do antecedente e o consequente aparecerem na mesma transação (i.e. a probabilidade condicional do consequente dado o antecedente). Por exemplo, no conjunto de dados de permissões, CHANGE_WIFI_STATE implica ACCESS_WIFI_STATE com 99,3% de confiança. Já o suporte de uma regra indica a frequência com que os itens na regra ocorrem juntos. Por exemplo, CHANGE_WIFI_STATE e ACCESS_WIFI_STATE podem aparecer juntos em 16,07% das transações. Nesse caso, as duas regras a seguir teriam, cada uma, um suporte mínimo de 16,07%. O *lift* nos diz que a probabilidade de WRITE_SMS e READ_SMS aparecerem juntos é 5,26 vezes maior do que a probabilidade de apenas READ_SMS e que são positivamente correlacionados.

Tabela 3. Saída do algoritmo Apriori

Antecedentes	Consequentes	Suporte	Confiança	Lift
CHANGE_WIFI_STATE	ACCESS_WIFI_STATE	0.160758	0.993016	2.28
MANAGE_ACCOUNTS	GET_ACCOUNTS	0.103359	0.992971	3.32
WRITE_SMS	READ_SMS	0.111407	0.984136	5.26

Apesar de WRITE_SMS e READ_SMS pertencerem à lista de permissões perigosas da Google, não é necessário considerar ambas as permissões, pois uma delas é suficiente para caracterizar comportamentos de aplicativos maliciosos. Algo similar ocorre com outras permissões. Por exemplo, pode-se remover, além de WRITE_SMS, as permissões MANAGE_ACCOUNTS e ACCESS_WIFI_STATE. Após remover as 3 permissões, identificadas pela regra de associação, chega-se a um conjunto de 27 permissões, apresentadas na Tabela 4.

Tabela 4. Lista de permissões selecionadas após aplicação do MLDP

PRNR+SPR+PMAR		
ACCESS_NETWORK_STATE	CHANGE_WIFI_STATE	WRITE_EXTERNAL_STORAGE
WRITE_SETTINGS	READ_PHONE_STATE	CAMERA
WAKE_LOCK	CALL_PHONE	RECEIVE_BOOT_COMPLETED
WRITE_CONTACTS	VIBRATE	READ_EXTERNAL_STORAGE
GET_ACCOUNTS	USE_CREDENTIALS	ACCESS_FINE_LOCATION
READ_HISTORY_BOOKMARKS	ACCESS_COARSE_LOCATION	CHANGE_NETWORK_STATE
SEND_SMS	RECORD_AUDIO	READ_CONTACTS
READ_SYNC_SETTINGS	RECEIVE_SMS	RESTART_PACKAGES
READ_SMS	BLUETOOTH	GET_TASKS

3. Resultados

Para a reprodução do SigPID, utilizamos o algoritmo *Support Vector Machine* (SVM), conforme proposto pelos autores no trabalho original [Li et al., 2018]. A Tabela 5 apresenta os resultados da execução do SVM para os diferentes conjuntos de dados, incluindo métricas para a avaliação do desempenho da detecção e o tempo de execução.

Tabela 5. Métricas de avaliação SVM

Conjunto de Dados	Quantidade de Permissões	Precisão	Recall	FPR	F1_Score	Acurácia	Tempo Execução (s)
Nível 1 (PRNR)	108	93,62	88,01	3,52	90,73	93,35	5,44
Nível 2 (SPR)	30	90,03	82,25	5,35	85,96	90,07	2,41
Nível 3 (PMAR)	27	90,13	82,07	5,28	85,91	90,05	2,26
Baseline	113	93,49	87,83	3,59	90,57	93,24	5,84
Perigosas Google	22	86,76	71,88	6,44	78,62	85,55	2,34
Recorrentes	32	88,54	81,53	6,19	84,89	89,27	3,17
SigPID	22	91,77	74,88	3,94	82,47	88,23	2,62

Como podemos observar, os conjuntos de dados que utilizam o mesmo número de permissões obtiveram resultados diferentes. As 22 permissões identificadas no SigPID se destacam em todas as métricas de avaliação quando comparadas com as 22 permissões

consideradas perigosas pela Google, ou seja, utilizar permissões perigosas não leva a um melhor resultado qualitativo. Isto indica que a seleção das permissões possui, de fato, um impacto no desempenho na detecção de *malwares*.

Quando reduzimos o número de permissões de 113 (*baseline*) para 108 com o nível 1 do MLDP, alcançamos taxas de precisão e acurácia mais altas, acima de 93%. Apesar de haver aumento no *recall*, o F1-Score e a taxa de falsos positivos (FPR) permaneceram mais baixa em relação ao *baseline*, assim como o tempo de execução que também diminuiu. Quando reduzimos o número de permissões de 113 para 30, com o nível 2 do MLDP, mantiveram-se a acurácia e precisão na faixa de 90%, porém com F1-Score abaixo de 90%. Além disso, obtivemos um ganho de mais de 2 segundos em relação ao tempo de execução.

Além do SVM, avaliamos também outros três algoritmos (*Random Forest*, *Decision Tree* e *Functional Trees* [Gama, 2004]) e comparamos os resultados com os conjuntos de dados discriminados na Tabela 6.

Tabela 6. Conjuntos de Dados

Nº de Permissões	Conjunto de dados	Observação
113	*Baseline*	Contidas no conjunto de dados Drebin_215
22	SigPID	Identificadas no trabalho original do SigPID
32	Recorrentes	Identificadas por meio da interseção de permissões identificadas em outros trabalhos
22	Perigosas	Permissões na lista de perigosas da Google e estavam contidas na *baseline* deste trabalho
27	MLDP	Identificadas após aplicação dos três níveis de seleção sobre a *baseline*

A Tabela 7 sintetiza os resultados dos algoritmos *Random Forest*, *Decision Tree* e *Functional Trees*. As métricas apresentadas são a acurácia (que indica o desempenho geral do modelo) e F1-Score (a média harmônica entre o *recall* e a precisão).

Tabela 7. Métricas de avaliação dos conjuntos de dados

	Decision tree		Random forest		Functional Trees	
Conjunto de dados	F1_Score	Acurácia	F1_Score	Acurácia	F1_Score	Acurácia
113 Baseline	92,28	94,32	94,44	95,94	97,30	97,27
22 SigPID	88,46	92,00	89,30	92,57	92,90	93,05
27 MLDP	91,09	93,50	92,62	94,64	95,40	95,43
32 Recorrentes	90,89	93,44	92,46	94,57	95,60	95,63
22 Perigosas Google	81,62	87,74	85,31	88,85	89,10	89,02

Quatro dos cinco conjuntos de dados obtiveram acurácia acima dos 90% (*Baseline*, SigPID, MLDP e Recorrentes). Destes, o *Baseline* apresentou melhor resultado que

os demais. Quando utilizamos o *Decision Tree*, atingimos 94,32% de acurácia e 92,28% de F1-Score. Com os algoritmos *Random Forest* e *Functional Trees*, atingimos 95,94% de acurácia e 94,44% de F1-Score e acurácia e F1-Score acima dos 97%, respectivamente.

Utilizando 22 permissões do SigPID, a acurácia se manteve na faixa de 92% com *Decision Tree* e *Random Forest*. Já F1-Score se manteve abaixo de 90%. O *Functional Trees*, com o melhor desempenho, atingiu 93,05% de acurácia e 92,90% de F1-Score. A Tabela 7 ainda mostra que os conjuntos de dados MLDP e Recorrentes obtiveram resultados próximos, ficando ambos com acurácia acima de 95% para o *Functional Trees*.

Na Figura 5 apresentamos os dados sobre o tempo de execução de cada algoritmo. Como podemos observar, o *baseline* tem o maior tempo de execução, chegando a 11,31 segundos com algoritmo *Functional Trees*. Entretanto, é interessante observar que o problema ocorre apenas para o *baseline*, já que o algoritmo executa em menos tempo que alguns dos demais para conjuntos de dados menores.

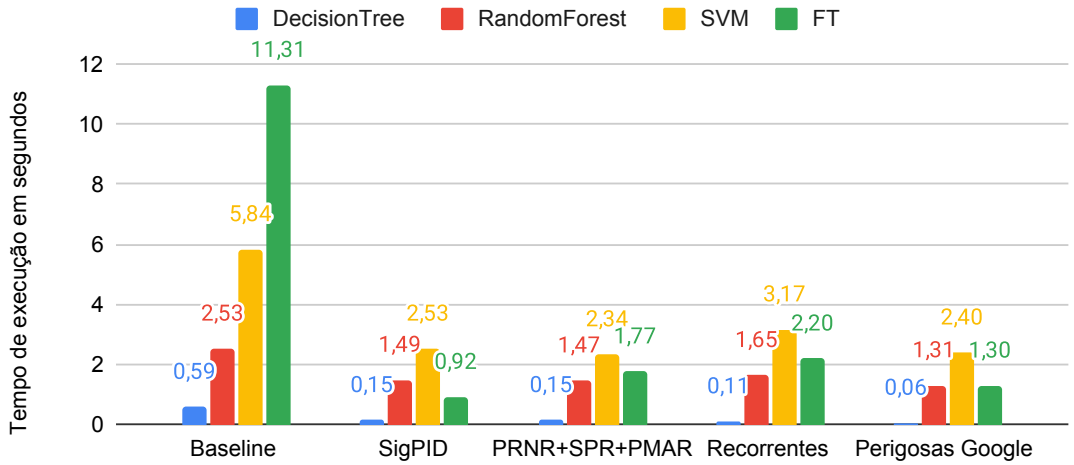

Figura 5. Tempo de execução para diferentes conjuntos de dados

O *Functional Trees* é um algoritmo baseado em árvores de classificação que podem ter funções de regressão logística nos nós internos ou folhas. Por conseguir lidar com variáveis binárias e multi-classe, atributos numéricos e valores ausentes, seu tempo de execução pode aumentar conforme o número de características de entrada [Gama, 2004]. Como a estrutura do algoritmo é uma generalização das *Multivariate Trees* [Gama, 2001], sua complexidade é $\mathcal{O}(n^2)$, o que explica o comportamento apresentado no gráfico da Figura 5. De fato, o *Functional Trees* reduziu significativamente o tempo de execução para conjuntos de dados menores (e.g., 1,77s para as 27 permissões do MLDP).

Ao analisarmos os conjuntos de dados que utilizam 27 e 32 permissões, observamos que o tempo de execução e as métricas estão muito próximas para a *Functional Trees*. Nesse caso, podemos dizer que os modelos são equivalentes, isto é, não há diferença em utilizar as 32 permissões recorrentes ou as 27 permissões do MLDP. Isto indica que a identificação das permissões recorrentes, em trabalhos existentes na literatura, leva a resultados tão bons quanto os resultados do método de múltiplos níveis de seleção do SigPID, o que confirma uma das nossas hipóteses, isto é, permissões recorrentes podem ter um impacto positivo sobre os modelos de detecção de *malwares*.

Discussão

O tempo de execução é o mais relevante? Como pode ser observado nos dados apresentados, o algoritmo *Decision Tree* foi o que executou no menor tempo em todos os conjuntos de dados, ficando abaixo de 1 segundo. Entretanto, o algoritmo ficou abaixo do *Decision Tree* e *Functional Tree* em relação às métricas de desempenho acurácia e F1-Score. Portanto, cabe ao usuário final realizar uma análise do *trade-off* entre tempo de execução e desempenho, isto é, latência do aprendizado $versus$ capacidade de detecção do modelo.

O tempo de execução pode tornar-se irrelevante?. Como executamos os experimentos em um ambiente com baixa capacidade computacional, acreditamos que o tempo de execução seja praticamente irrelevante quando aplicarmos os modelos em *smartphones* atuais. Enquanto os experimentos foram realizados utilizando uma CPU Intel Celeron 1007U de 1.5GHz, que produz apenas 96 Gigaflops, *smartphones* modernos disponibilizam CPUs como a Qualcomm Snapdragon 865, que opera em 2.84GHz e consegue produzir 1.228 Gigaflops, ou seja, um poder computacional mais de 12x maior. Outros fatores que irão impactar o tempo de execução são as instruções avançadas em CPUs modernas, voltadas para aprendizado de máquina, e a velocidade da memória RAM. Enquanto o experimento foi executado em um hardware com 4GB de RAM operando a 1.600MHz, um *smartphone* moderno fornece 6GB de RAM operando a 2.750MHz.

A quantidade de dados impacta o tempo de execução?. Conjuntos de dados maiores e mais atuais impactam o desempenho dos algoritmos de aprendizado de máquina. Por exemplo, o algoritmo SVM apresenta problemas de desempenho para conjuntos de dados maiores, aumentando substancialmente o tempo de treinamento [Cristianini et al., 2000]. Algo similar pode ser dito do algoritmo *Functional Trees*, que apresentou um tempo de computação substancialmente maior para 113 permissões (i.e., um conjunto com mais características), por exemplo.

Existem Desafios?. Encontramos diversos problemas de reprodutibilidade do SigPID, como a falta de informação sobre os hiper-parâmetros utilizados nos algoritmos, indisponibilidade de conjuntos de dados e falta de detalhamento de ferramentas e tecnologias utilizadas pelo mesmo.

4. Considerações Finais

Aplicando os 3 níveis de seleção do SigPID nas 113 permissões do conjunto de dados Drebin_215, conseguimos reduzir em 76% o número de permissões a serem analisadas para detecção de *malwares* Android, mantendo acurácia acima de 90% com SVM, 93,50% com *Decision Tree*, 94,64% com *Random Forest* e 95% com o *Functional Trees*. Além disso, conseguimos também reduzir o tempo de execução dos modelos, porém ao custo de um leve aumento na taxa de falsos positivos.

Além das 27 permissões resultantes dos 3 níveis de seleção, utilizamos também conjuntos de 113, 22 e 32 permissões. Percebemos que o conjunto de dados que utiliza todas as permissões (113) foi o que melhor performou (e.g., 97% de acurácia), porém ao preço de um tempo de execução significativamente maior. Um caso interessante ocorreu ao compararmos os conjuntos com 22 permissões, sendo um oriundo do trabalho original do SigPID e 22 permissões classificadas como perigosas pela Google. O SigPID chegou a 93% de acurácia enquanto as perigosas mantiveram acurácia abaixo de 90%, o que indica

que o fato da permissão ser classificada como perigosa não a torna necessariamente relevante para detecção de *malwares*. Outra observação interessante é o fato de os conjuntos de dados com 32 permissões mais recorrentes e das 27 identificadas atingirem resultados muito próximos ao aplicar os 3 níveis de seleção. Isto indica que escolher as permissões conforme a recorrência pode ser um caminho a ser investigado.

Como trabalhos futuros, podemos elencar: (a) testes com conjuntos de dados maiores; (b) testes com conjuntos de dados atuais; (c) avaliação dos níveis de seleção para outras características (e.g., *intents* e chamadas de API); (d) otimização de hiperparâmetros; (e) testar os modelos em *smartphones* modernos; e (f) mensurar o tempo de execução dos modelos em CPUs modernas projetadas para acelerar a computação de algoritmos de aprendizado de máquina.

Agradecimentos

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, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499. Citeseer.

Alsoghyer, S. and Almomani, I. (2020). On the effectiveness of application permissions for android ransomware detection. In *2020 6th Conference on Data Science and Machine Learning Applications (CDMA)*, pages 94–99.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., and Siemens, C. (2014). Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26.

Assolin, J., Rocha, V., Silveira, G., Rodrigues, G., Feitosa, E., Casola, K., and Kreutz, D. (2021). Detecção de Malwares Android: reprodução da seleção de características do SigPID. In *VI Workshop Regional de Segurança da Informação e de Sistemas Computacionais (WRSeg)*, Charqueadas-RS, Brasil.

Cristianini, N., Shawe-Taylor, J., et al. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press.

Gama, J. (2001). Functional trees for classification. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 147–154. IEEE.

Gama, J. (2004). Functional trees. *Machine learning*, 55(3):219–250.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., and Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225.

Sun, L., Li, Z., Yan, Q., Srisa-an, W., and Pan, Y. (2016). Sigpid: significant permission identification for android malware detection. In *2016 11th international conference on malicious and unwanted software (MALWARE)*, pages 1–8. IEEE.

Wu, Q., Zhu, X., and Liu, B. (2021). A survey of android malware static detection technology based on machine learning. *Mobile Information Systems*, 2021.

Yildiz, O. and Doğru, I. A. (2019). Permission-based android malware detection system using feature selection with genetic algorithm. *International Journal of Software Engineering and Knowledge Engineering*, 29(02):245–262.