Universidade Federal do Amazonas
Instituto de Computação
Programa de Pós-Graduação em Informática

Vanderson da Silva Rocha

Avaliação de Métodos de Classificação Baseados em Regras de Associação para Detecção de *Malwares* Android

Manaus - Amazonas
2023

Vanderson da Silva Rocha

Avaliação de Métodos de Classificação Baseados em Regras de Associação para Detecção de *Malwares* Android

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Instituto de Computação da Universidade Federal do Amazonas como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Eduardo Luzeiro Feitosa

Manaus - Amazonas
2023

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

Rocha, Vanderson da Silva
R672a Avaliação de métodos de classificação baseados em regras de associação para detecção de malwares Android / Vanderson da Silva Rocha . 2023
83 f.: il. color; 31 cm.

Orientador: Eduardo Luzeiro Feitosa
Coorientador: Diego Luis Kreutz
Dissertação (Mestrado em Informática) - Universidade Federal do Amazonas.

1. Android. 2. Detecção de Malwares. 3. Regras de Associação. 4. Qualidade de Regras. 5. Eqar. I. Feitosa, Eduardo Luzeiro. II. Universidade Federal do Amazonas III. Título

Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Programa de Pós-Graduação em Informática

FOLHA DE APROVAÇÃO

"AVALIAÇÃO DE MÉTODOS DE CLASSIFICAÇÃO BASEADOS EM REGRAS DE ASSOCIAÇÃO PARA DETECÇÃO DE MALWARES ANDROID"

VANDERSON DA SILVA ROCHA

Dissertação de Mestrado defendida e aprovada pela banca examinadora constituída pelos Professores:

Prof. Dr. Eduardo Luzeiro Feitosa - PRESIDENTE

Prof. Dr. Horácio Antônio Braga Fernandes de Oliveira - MEMBRO INTERNO

Prof. Dr. André Ricardo Abed Grégio - MEMBRO EXTERNO

Manaus, 19 de Abril de 2023

Documento assinado eletronicamente por **Eduardo Luzeiro Feitosa**, **Professor do Magistério Superior**, em 25/04/2023, às 12:36, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **André Ricardo Abed Grégio**, **Usuário Externo**, em 26/04/2023, às 16:30, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

Documento assinado eletronicamente por **Horácio Antônio Braga Fernandes de Oliveira**, **Professor do Magistério Superior**, em 27/04/2023, às 14:55, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufam.edu.br/sei/controlador_externo.php?

Avenida General Rodrigo Octávio, 6200 - Bairro Coroado I Campus Universitário Senador Arthur Virgílio Filho, Setor Norte - Telefone: (92) 3305-1181 / Ramal 1193
CEP 69080-900, Manaus/AM, coordenadorppgi@icomp.ufam.edu.br

Referência: Processo nº 23105.013826/2023-61 SEI nº 1467405

Dedico esta dissertação aos meus pais,
Mara Nilza Calvalcente da Silva (in memoriam) e Valdenor dos Santos Rocha,
que nunca mediram esforços para me dar uma boa educação
e me ensinaram, desde cedo, a importância dos estudos.

Entrega o teu caminho ao Senhor;
confia nEle,
e Ele tudo fará.

Salmos 37,5

Agradecimentos

Primeiramente agradeço a Deus, por ter me dado força e saúde para enfrentar as tribulações que surgiram em minha vida antes e durante o curso, mas não me impediram de alcançar este objetivo.

Quero agradecer à minha família, meu porto seguro, que sempre me apoiou nos estudos e nunca me deixou esmorecer perante os desafios.

À minha amada Nayara Medeiros, por motivar-me e por ser tão compreensível nos momentos em que eu estive ausente.

Ao meu orientador Prof. Dr. Eduardo Luzeiro Feitosa e coorientador Prof. Dr. Diego Luis Kreutz pelo conhecimento, auxílio e paciência que foram de suma importância durante o desenvolvimento deste trabalho.

Aos professores, Dr. Edson N. da Silva Jr. e Dr. Raimundo S. Barreto, pela confiança em mim depositada ao redigirem as suas cartas de recomendação.

Aos colegas do CTIC-UFAM (Centro de Tecnologia de Informação e Comunicação) pelo clima agradável de descontração e pelo café.

Por fim, agradeço a todos aqueles que a seu modo contribuíram e me apoiaram para a conclusão deste trabalho.

Muito obrigado!

Resumo

Esta pesquisa tem por objetivo investigar o desempenho e a viabilidade de diferentes modelos de regras de associação no contexto de classificação de *malwares* Android. Para tanto, desenvolvemos um novo modelo de classificação baseado em regras de associação e qualidade de regras. Para fins de comparação dos modelos, utilizamos *datasets* conhecidos e frequentemente usados para o treino de modelos de detecção de *malwares* Android. Os resultados demonstram que nosso modelo possui desempenho equivalente a outros modelos baseados em regras de associação, obtendo valores de acurácia acima de 85%, e em alguns casos sobressaindo-se a modelos de aprendizagem de máquina.

Palavras-chave: Android, Detecção de *Malwares*, Regras de Associação, Mineração de Dados, Apriori, FP-Growth, ECLAT, Qualidade de Regras, EQAR

Abstract

This research aims to investigate the performance and viability of different models of association rules in the context of classifying Android malwares. For that, we developed a new classification model based on association rules and rule quality. For model comparison purposes, we used known and frequently used datasets for training Android malware detection models. The results demonstrate that our model performs equivalently to other models based on association rules, obtaining accuracy values above 85%, and in some cases surpassing machine learning models.

Keywords: Android, Malware Detection, Association Rules, Data Mining, Apriori, FP-Growth, ECLAT, Rules Quality, EQAR

Lista de Figuras

Lista de Tabelas

Nomenclatura

AOT Compilação Antecipada *(Ahead of Time)*

API Interfaces de Programação de Aplicativos (*Application Program Interface*)

APK *Android Package Kit*

ARCID *Association Rule-Based Classification for Imbalanced Datasets*

ART Android Runtime

CBA *Classification Based on Associations*

CFG Grafo de Fluxo de Controle (*Control Flow Graph*)

CMAR *Classification Based on Multiple Association Rules*

CPAR *Classication Based on Predictive Association Rules*

CPU Unidade Central de Processamento (*Central Process Unit*)

DFG Grafo de Fluxo de Dados (*Data Flow Graph*)

EQAR *ECLAT and Qualified Association Rules*

GPS Sistema de Posicionamento Global (*Global Positioning System*)

HAL Camada de Abstração de Hardware (*Hardware Abstraction Layer*)

IFME Intervalo de Frequência Mais Evidente

KDD Descoberta de Conhecimento em Bancos de Dados (*Knowledge Discovery in Databases*)

KNN K-ésimo Vizinho Mais Próximo (*k-Nearest Neighbors Algorithm*)

MCAR *Multi-Class Classification Based on Association Rules*

NLP Processamento de Linguagem Natural (*Natural Language Processing*)

PRM *Predictive Rule Mining*

PRNR Classificação de Permissão com Taxa Negativa (*Permission Ranking with Negative Rate*)

PSO *Particle Swarm Optimization*

RF Florestas Aleatórias (*Random Forest*)

SDK Kit de Desenvolvimento de Software (*Software Development Kit*)

SMS Serviço de Mensagens Curtas (*Short Message Service*)

SVM Máquina de Vetores de Suporte (*Support Vector Machine*)

UI Interface de Usuário (*User Interface*)

Sumário

Capítulo 1

Introdução

Este capítulo contém uma visão geral da pesquisa realizada nesta dissertação. Em primeiro lugar, expomos os principais motivos pelos quais escolhemos a **detecção de *malwares* em dispositivos Android** como escopo principal do trabalho. Em segundo lugar, apresentamos o histórico da pesquisa, enfocando brevemente as principais questões de seu domínio de conhecimento e esclarecendo porque essas questões merecem atenção. Em seguida, relacionamos o enunciado do problema de pesquisa com os objetivos desta dissertação, bem como a metodologia de pesquisa utilizada. Finalmente, demonstramos como a dissertação está estruturada.

1.1 Problema e Motivação

Os dispositivos móveis, em especial os *smartphones*, tornaram-se indispensáveis em grande parte das atividades humanas por permitir acesso às informações a qualquer momento. Neste cenário, o sistema operacional Android, desenvolvido pela Google, é o mais embarcado e utilizado em *smartphones* em todo o mundo. De acordo com a StatCounter, site especializado em análise de tráfego web, *smartphones* executando o Android representavam, em fevereiro de 2023, 72,26% do mercado global (StatCounter, 2023). Somado-se a isso, a Google Play Store possuía cerca de 2,6 milhões de aplicativos disponíveis para o sistema operacional Android em março de 2023 (Statista, 2023).

Por ser o sistema operacional mais utilizado, o Android também é o mais visado para ataques e fraudes. O último *Mobile Security Report* da G DATA CyberDefense (GDATA, 2022) mostra que mais de 700 mil aplicativos maliciosos para *smartphones* com sistema operacional Android estavam em circulação nos primeiros seis meses do ano de 2022, indicando que um novo aplicativo com *malware* aparece a cada 23 segundos.

Para combater a presença de aplicativos maliciosos no mercado, vêm sendo criadas diversas soluções de análise e detecção de *malwares*, grande parte delas concentradas nas características dos aplicativos para realizar a detecção (Wang et al., 2019). Essas características podem ser estáticas ou dinâmicas e são utilizadas para representar o comportamento dos aplicativos, permitindo identificar possíveis ameaças. De acordo com Muttoo e Badhani (2017), as soluções para detecção de *malwares* Android utilizam técnicas como:

- **Aprendizagem de máquina**: que utiliza algoritmos de aprendizagem para analisar as características dos aplicativos e identificar padrões que possam indicar a presença de *malwares*.
- **Detecção de anomalias**: que identifica comportamentos anômalos em relação aos padrões de uso normal do dispositivo.

Dentre esses tipos de soluções, pode-se afirmar que as baseadas em aprendizagem de máquina são as mais pesquisadas. Prova disto, é a grande quantidade de artigos e *surveys* sobre o assunto. Entretanto, se o processo de detecção é feito ao se avaliar as características dos aplicativos, porque não empregar outros tipos de soluções? Uma alternativa, pouco explorada na detecção de aplicativos maliciosos, é a mineração de regras de associação, que pode ser usada para identificar padrões frequentes nos dados. Na detecção de *malwares* Android, essa técnica pode ser útil para identificar as combinações de características de aplicativos que possam representar riscos de segurança.

Inicialmente proposta para resolver o problema da cesta de mercado em conjuntos de dados transacionais (Agrawal et al., 1993), é um procedimento que visa observar padrões, correlações ou associações que ocorrem com frequência em conjuntos de dados encontrados em bancos de dados ou outras formas de repositórios. Genericamente, a mineração de regras de associação pode ser encarada como simples instruções `Se ... Então` que ajudam a descobrir relacionamentos entre os dados e tem sido utilizada em problemas do mundo real, como gestão de relacionamento com o cliente (Kaur e Kang, 2016; Kim e Yum, 2011), diagnóstico médico (Ali et al., 2019; Gu et al., 2016) e na área de segurança da informação (e.g., no combate à fraudes) (Jeeva e Rajsingh, 2016; Sadgali et al., 2021).

Na área de mineração de regras de associação, existem três algoritmos clássicos (Apriori, FP-Growth e ECLAT) recorrentemente referenciados e utilizados na literatura (Li e Sheu, 2021; Zhang e He, 2010). A partir destes algoritmos base, surgiram diversos métodos especializados em classificação baseada em regras de associação, como CBA

(Liu et al., 1998), CMAR (Li et al., 2001), CPAR (Yin e Han, 2003), MCAR (Thabtah et al., 2005) e ARCID (Abdellatif et al., 2018). Esses métodos incorporam e/ou evoluem os algoritmos clássicos, como por exemplo, o CMAR, que incrementa o FP-Growth para minerar grandes conjuntos de dados e assim melhorar a precisão e a eficiência da classificação.

Com base nesse contexto, esta pesquisa, parte integrante do projeto "Malware Hunter" (financiado pela Flex e Motorola), é motivada pela necessidade de elaborar novas formas de detecção de *malwares* Android. Para tanto, propomos duas questões de pesquisa:

1. *Os algoritmos de mineração de regra de associação são mais eficientes do que os algoritmos de aprendizagem de máquina na detecção de malwares Android?*

2. *Qual melhor algoritmo de mineração de regra de associação para detecção de malwares Android?*

1.2 Objetivo

O objetivo desta dissertação é realizar uma análise exploratória de diferentes algoritmos de classificação baseados na mineração de regras de associação para o contexto de classificação de aplicativos maliciosos Android, utilizando um conjunto de *datasets* conhecidos e comparando-os com algoritmos clássicos de aprendizagem de máquina.

Para atingir esse objetivo, pretende-se realizar os seguintes objetivos específicos:

- Avaliar os algoritmos de mineração de regras de associação existentes na literatura com a finalidade de identificar aquele que melhor se enquadra a geração de regras de associação utilizando características de aplicativos Android;

- Desenvolver um método utilizando métricas de qualidade de regra a fim de obter regras mais significativas e representativas, melhorando o processo de detecção realizado pelo método proposto.

- Estimar, para o método proposto, os parâmetros iniciais que permitam a geração de regras e obtenha os melhores resultados;

1.3 Organização da Dissertação

O restante desta dissertação está organizado da seguinte forma:

- O Capítulo 2 aborda os conceitos teóricos utilizados na pesquisa, bem como os fundamentos que norteiam o desenvolvimento de métodos para detecção de *malwares* em dispositivos Android.
- O Capítulo 3 apresenta diversos estudos anteriores que trabalharam na detecção de aplicativos maliciosos ou que fazem uso de mineração de regras de associação, incluindo uma discussão desses trabalhos frente ao proposta nesta dissertação.
- O Capítulo 4 descreve a concepção e algoritmos gerados para a criação do método proposto, assim como sua comparação a outros métodos de classificação baseados em regras.
- O Capítulo 5 apresenta as conjuntos de dados e métricas avaliados, os parâmetros utilizados, como e em quais condições ocorreram as avaliações.
- O Capítulo 6 compreende a avaliação e comparação do método proposto, apresentando e discorrendo a respeitos dos resultados obtidos, incluindo métodos de seleção de características.
- O Capítulo 7 traz as conclusões gerais deste trabalho, incluindo seus desafios e trabalhos futuros.

Capítulo 2

Conceitos Básicos

Neste capítulo, apresentamos as informações necessárias para entender o contexto deste trabalho. É de suma importância ter uma visão geral do modelo de segurança do sistema operacional Android, para ser capaz de avaliar seus pontos fortes e vulnerabilidades e, portanto, ter uma ideia das deficiências, aspectos e/ou recursos que precisam ser melhorados e que são explorados pelos desenvolvedores de *malwares*.

2.1 Sistema Android

O Android é um sistema operacional baseado em Linux projetado principalmente para dispositivos móveis com tela de toque, como *smartphones*, relógios inteligentes, *tablets* e leitores eletrônicos.

Os aplicativos Android são escritos em Java e compilados, junto com arquivos de dados e recursos, em um único arquivo. O formato de arquivo usado para distribuir e instalar aplicativos móveis é o *Android Package Kit* (APK). Um arquivo APK contém todo o código do binário (arquivos .dex compilados) e diversos outros recursos do aplicativo empregados na detecção de *malwares*. A Figura 2.1 ilustra a estrutura de um arquivo APK, que consiste em diversos componentes e diretórios, incluindo:

- Assinaturas (META-INF), que contém os certificados e as assinaturas.
- *Asset*, diretório contendo ativos de aplicativos que podem ser recuperados pelo AssetManager.
- *Lib* (bibliotecas nativas), diretório que contém o código compilado de bibliotecas, que é dependente da plataforma. O diretório código compilado para plataformas como:

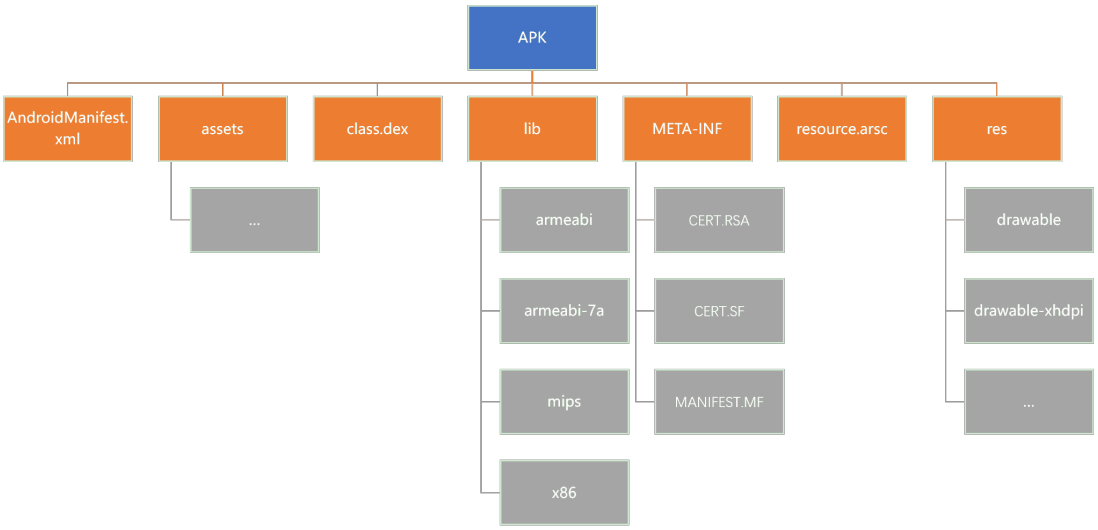


Figura 2.1: Estrutura do Arquivo APK (Al-Fawa'reh et al., 2020)

  - `armeabi-v7a`: código compilado para processadores baseados em ARMv7 e versões superiores;
  - `arm64-v8a`: código compilado para processadores baseados em ARMv8 e versões superiores;
  - `x86`: código compilado para processadores x86;
  - `x86_64`: código compilado apenas para processadores x86 64 bits.

- *Res* (recursos), diretório que contém recursos pré-compilados, como XML binário no arquivo `resources.arsc`;
- *Classes.dex*, arquivos no formato DEX utilizados para inicializar e executar aplicativos desenvolvidos para o Android. Os dados armazenados nesses arquivos DEX incluem o código compilado que localiza e inicializa outros arquivos de programas, necessários para executar o aplicativo.
- *AndroidManifest.xml*, arquivo de manifesto do Android que descreve o nome, a versão, os direitos de acesso (permissões) e os arquivos de biblioteca referenciados no aplicativo. Este arquivo pode estar em XML binário, que pode ser convertido em XML de texto simples, isto é, legível por humanos.

Dentre estes, o `AndroidManifest.xml` (arquivo de manifesto do aplicativo) é essencial para extração de grande parte das características estáticas. Todo aplicativo Android

precisa ter um arquivo `AndroidManifest.xml`, uma vez que ele descreve informações essenciais sobre o aplicativo para as ferramentas de compilação do Android, para o sistema operacional e para a loja de aplicativos.

Um arquivo de manifesto precisa declarar os seguintes itens:

- **Nome do pacote do aplicativo**;
- **Componentes do aplicativo**, que incluem todos os serviços, *broadcast receivers*, provedores de conteúdo e atividades. Cada componente precisa definir propriedades básicas, além de poder declarar recursos como quais configurações de dispositivo podem ser processadas e os filtros de *intents* que descrevem a forma de inicialização do componente;
- **Permissões** que o aplicativo precisa ter para acessar partes protegidas do sistema ou de outros aplicativos. Também declara todas as permissões que outros aplicativos precisam ter para acessar conteúdo desse aplicativo;
- **Recursos de hardware e software** exigidos pelo aplicativo, que afetam quais dispositivos podem instalar o aplicativo a partir da loja.

A Figura 2.2 ilustra um exemplo do conteúdo de um **AndroidManifest.xml**.

2.2 Características de Aplicativos Android

Na detecção de *malwares* em Android, a característica é definida como uma informação de algum aspecto da aplicação (por exemplo, permissão), que permite determinar se tal aplicação possui indícios de comportamento malicioso. Os trabalhos na literatura categorizam, tipicamente, as características extraídas de aplicativos Android de duas formas: estáticas e dinâmicas.

Por definição, características estáticas são informações sobre o aplicativo como nome, tamanho, permissões requeridas, código-fonte e padrão de programação, obtidas sem a necessidade de instalá-lo e executá-lo. Já as características dinâmicas são aquelas que refletem os comportamentos do aplicativo na interação com o sistema operacional ou na conectividade de rede, e para serem obtidas requerem a instalação e execução do APK e a coleta de processos de execução do aplicativo, seja através de um emulador ou dispositivo físico.

Um dos processos de detecção de *malwares* Android (Figura 2.3), adaptado de Wang et al. (2019), aponta que a partir dos APKs pode-se extrair diversos tipos de

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.toggletest"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.toggletest.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

</manifest>
```

Figura 2.2: Exemplo de `AndroidManifest.xml`

características, transformá-las em uma matriz e então submeter esta ao aprendizado de algum modelo de classificação, como redes neurais, redes bayesianas, clusterização, aprendizado profundo, cadeias de Markov, entre outros (Buczak e Guven, 2016).

A extração de características pode ser demorada devido ao tamanho crescente e aos comportamentos complexos de um APK, resultando na detecção ineficaz. Por exemplo, os aplicativos mais comuns agora aproveitam a localização de um usuário para fornecer recursos adicionais, como destacar pontos de interesse ou até mesmo outros usuários que possam estar próximos. Os aplicativos de realidade aumentada vão um passo além, aproveitando não apenas a localização do usuário, mas também a câmera e os sensores do telefone para fornecer uma experiência imersiva ao usuário. Os identificadores de

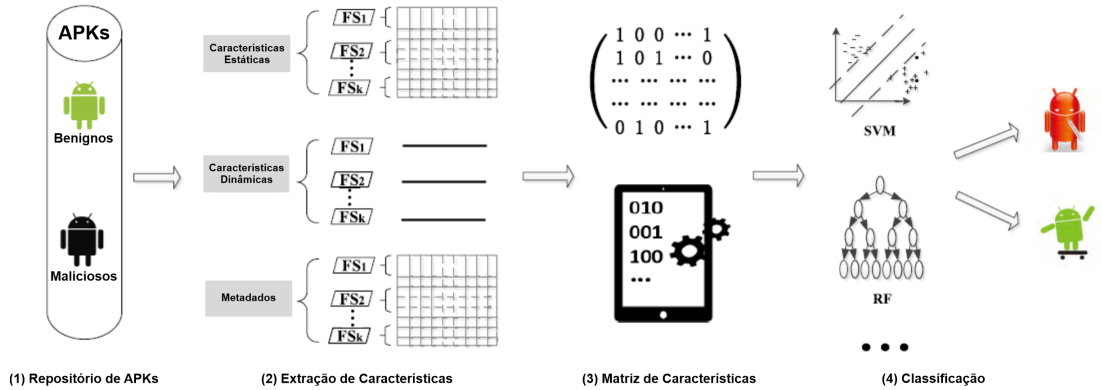


Figura 2.3: Detecção de *Malwares* Android Utilizando Características de APK. Adaptado de Wang et al. (2019)

telefone agora são comumente usados para identificar usuários exclusivamente por aplicativos que adaptam seu comportamento às necessidades do usuário. Isso significa que os aplicativos benignos agora usam as mesmas informações que os aplicativos mal-intencionados coletam. Como resultado direto, muitos dos comportamentos existem em aplicativos maliciosos e benignos são semelhantes (Shen et al., 2017).

As seções 2.2.1 e 2.2.2 apresentam, em detalhes, as características efetivamente empregadas na detecção de *malware* Android, com base nos resultados do estudo original de Wang et al. (2019), estendido até o ano de 2021 dentro do projeto Malware Hunter. Neste trabalho, os autores fornecem uma visão clara e abrangente do estado da arte dos trabalhos que detectam *malwares*, demonstrando comportamentos de aplicativos utilizando várias características, destacando a questão de explorar características eficazes.

2.2.1 Características Estáticas

A principal vantagem da utilização de características estáticas é o tempo e o processo de extração, que são menos custosos quando comparados com a extração de características dinâmicas. Por outro lado, a utilização de características estáticas pode apresentar problemas causados por fragilidades, como ofuscação de código, re-empacotamento da aplicação ou *malwares* que manifestam intenções maliciosas apenas em tempo de execução (*i.e.* somente podem ser detectados por análise de características dinâmicas).

Com base nos resultados do trabalho de Wang et al. (2019) e dos obtidos na extensão de sua revisão, a Figura 2.4 apresenta as características estáticas mais empregadas, bem como o número de artigos que as utilizam.

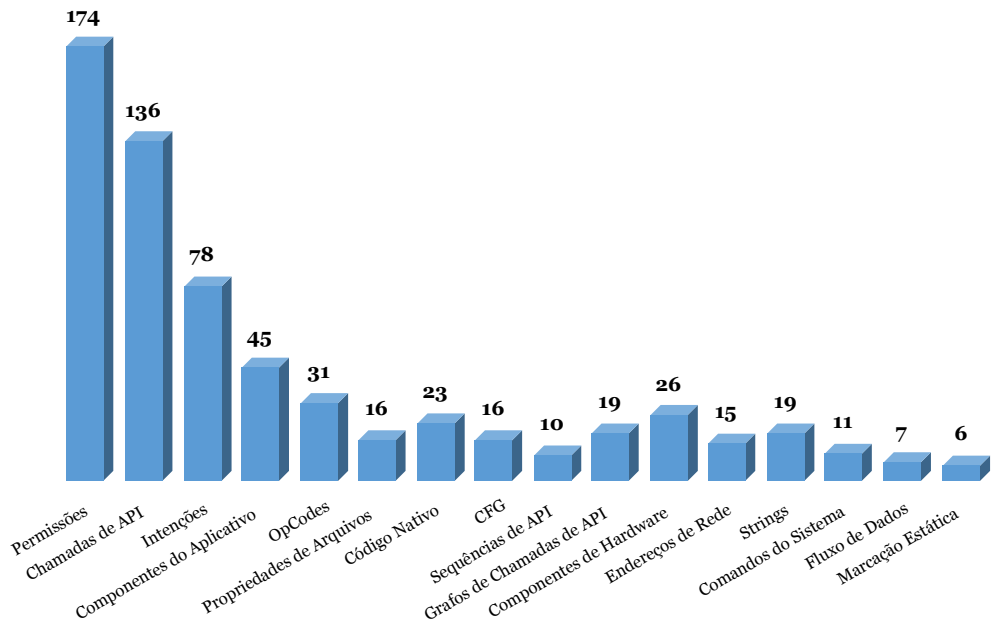


Figura 2.4: Trabalhos Utilizando Características Estáticas

Permissões

São privilégios especiais que os aplicativos devem pedir e ter aprovação para usar funcionalidades e recursos do sistema operacional e do dispositivo. Por definição, todas as permissões que um aplicativo precisa ter para acessar partes protegidas do sistema ou de outros aplicativos, ou as permissões que outros aplicativos precisam ter para acessar conteúdo desse aplicativo, são declaradas no arquivo de manifesto. A Figura 2.2, nas linhas 10, 11, 12, 29 e 30, ilustra um exemplo da declaração de permissões no arquivo de manifesto.

O Android possui três tipos de permissões (de instalação, de execução e especiais), a fim de refletir o escopo de dados restritos que o aplicativo pode acessar e o escopo de ações restritas que o aplicativo pode realizar.

As permissões são as características mais utilizadas em soluções de prevenção e detecção de *malwares* Android. Isto porque elas refletem o processo inicial de um aplicativo malicioso, que é solicitar acesso a recursos e informações cruciais para seus objetivos. Além disso, a existência de combinações entre várias permissões pode refletir alguns comportamentos prejudiciais, como o envio de SMS e/ou ligações para serviços *premium* e o funcionamento inadequado do dispositivo (Vaishanav et al., 2017).

Chamadas de API - *API Calls*

Pode ser entendida como uma forma de comunicação entre sistemas, em que um deles fornece informações e serviços que podem ser utilizados pelo outro, sem a necessidade do conhecimento de detalhes da implementação.

No Android, as chamadas de APIs representam a interação do aplicativo com a estrutura do sistema operacional, uma vez que precisam interagir com o dispositivo. Elas são categorizadas de acordo com o tipo de recursos solicitados e suas funcionalidades (Sharma e Dash, 2014).

Intenções - *Intents*

São mensagens assíncronas que permitem aos componentes de um aplicativo solicitar funcionalidades de outros componentes do Android (do mesmo aplicativo ou não), enviando objetos de intenção. É considerado um mecanismo de segurança para impedir que aplicativos tenham acesso direto a outros aplicativos. Por isso, os aplicativos devem ter permissões específicas para usar *Intents*. A Figura 2.2, das linhas 19 a 27, ilustra a declaração de uma atividade com um filtro de *intents* para receber um *intent* `MAIN` .

Existem dois tipos de *intents*: explícitos e implícitos. Os ***intents* explícitos** especificam qual aplicativo atenderá ao *intent*, enquanto os ***intents* implícitos** não nomeiam nenhum componente específico, mas declaram uma ação geral a realizar. No que tange a detecção, as *intents* precisam ser extraídas do arquivo de manifesto.

Componentes do Aplicativo - *App Components*

São os blocos de construção básicos de um aplicativo Android. Eles são os pontos de entrada para o sistema Android acessar aplicativos. Cada componente existe como uma entidade distinta e desempenha uma função específica. Esses componentes são vinculados pelo arquivo de manifesto do aplicativo.

Existem quatro componentes principais em um aplicativo Android:

- *Activity*, que controla a Interface do Usuário (IU) e lida com a interação do usuário com a tela do dispositivo.
- *Service*, que gerencia o processo em segundo plano do aplicativo para realizar operações de longa duração.
- *Broadcast Receiver*, que trata da comunicação entre o sistema operacional e os aplicativos.

- *Content Provider*, que resolve problemas de gerenciamento de dados e banco de dados. Os dados podem ser armazenados no sistema de arquivos, banco de dados ou em outro lugar.

Códigos de Operação - *OpCodes*

É uma instrução única (atômica) que pode ser executada pela CPU. No caso do Android, *OpCodes* são as instruções da máquina virtual para execução do aplicativo.

No que tange a detecção de *malware*, a frequência ou sequência de *OpCodes* dos aplicativos pode ser vista como característica relevante, uma vez que estão intimamente relacionados ao código fonte do aplicativo.

A Figura 2.5 ilustra o processo de extração desta característica..

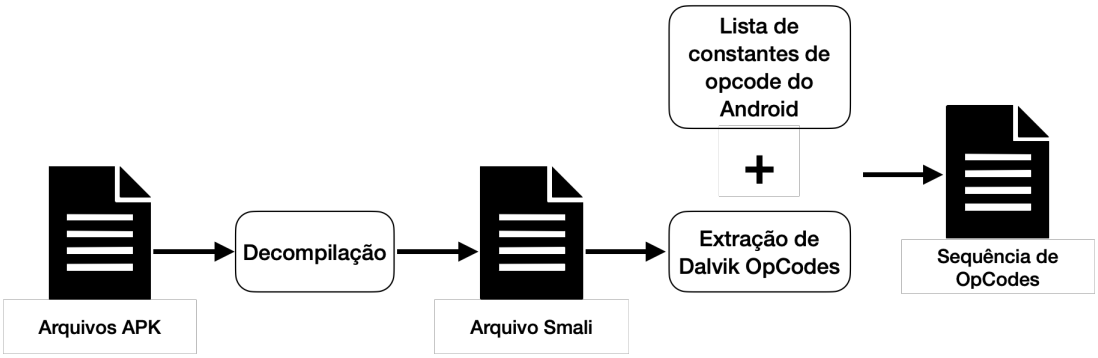


Figura 2.5: Exemplo de Obtenção de OpCodes

Endereços de Rede

Certos *malwares* precisam se comunicar com seus criadores, seja para obter instruções, seja para enviar as informações coletadas, os endereços de rede utilizados e acessados pelos aplicativos servem como indicadores de atividade maliciosa. Após a extração dos respectivos endereços de rede pode-se, por exemplo, proceder uma análise para detecção de padrões de *botnet* ou consultar uma *blacklist* de endereços de rede maliciosos.

Componentes de Hardware

Os aplicativos devem solicitar todo o hardware de que precisam para funcionar, como o uso da câmera (Figura 2.6). Certas combinações de hardware solicitado podem, portanto, implicar em comportamentos maliciosos. Por exemplo, não há necessidade aparente de um aplicativo de calculadora exigir acesso à 3G e GPS.

```
<uses-permission android:name="android.permission.CAMERA"/>

    <uses-feature android:name="android.hardware.camera"/>
    <uses-feature android:name="android.hardware.camera.autofocus"/>
    <uses-feature android:name="android.hardware.camera.front"/>
    <uses-feature android:name="android.hardware.camera.front.autofocus"/>
```

Figura 2.6: Permissão Para Uso da Câmera no `AndroidManifest.xml`

Grafo de Fluxo de Controle - *Control Flow Graph* (CFG)

O gráfico de fluxo de controle (CFG) é uma ferramenta essencial na análise estática de programas, pois permite visualizar, de forma gráfica, todos os caminhos que um programa pode seguir durante sua execução. Essa técnica é muito útil para a detecção de ofuscação de código, pois, embora seja possível alterar a sequência de chamadas de API ou renomear chamadas para evitar a detecção, o fluxo do código em si não é alterado.

O CFG pode ser construído a partir do código-fonte ou do código compilado e representa todas as possíveis instruções que podem ser executadas, bem como as condições que levam à sua execução. Como exemplo, ao construir o CFG, é possível identificar pontos no código em que a entrada e/ou dados do usuário é utilizada sem a devida validação ou sanitização, ou a injeção de código malicioso.

Marcação Estática - *Static Taint*

Pode ser vista como uma forma de análise do fluxo de informações. A ideia é que os fluxos de dados de fontes não confiáveis podem causar vulnerabilidades de segurança nos aplicativos. Portanto, a análise de marcação estática pode decidir se um vazamento realmente constitui uma violação de política e apresenta fluxos de dados potencialmente maliciosos para analistas humanos ou para ferramentas automatizadas de detecção de *malwares*.

A análise de marcação estática é realizada rastreando-se os fluxos de informações, examinando o código-fonte (ou seja, realizado um teste de caixa branca) (Wang et al., 2019).

Fluxo de Dados

A análise do fluxo de dados rastreia os dados dos aplicativos, mostrando as dependências de dados entre funções. Com base no fluxo de dados, as decisões relevantes para a

segurança podem ser tomadas automaticamente. Um grafo de fluxo de dados, ou *Data Flow Graph* (DFG), é uma representação gráfica do fluxo de dados através de um sistema de informação, apresentando que tipo de informação entrará e sairá do sistema, onde os dados serão armazenados e como eles são manipulados.

Ao examinar o DFG de um sistema de informação, é possível identificar possíveis pontos de vulnerabilidade relacionados à segurança dos dados. Por exemplo, pode-se detectar fluxos de dados não seguros, como informações sendo transmitidas sem criptografia ou autenticação adequada, ou dados sensíveis sendo armazenados em locais não seguros, como em arquivos de texto simples ou bancos de dados sem proteção adequada.

Propriedades de Arquivos

Os aplicativos Android são distribuídos como arquivos compactados, no formato APK, com o intuito de reduzir o número de arquivos baixados durante a instalação de um aplicativo. No entanto, como os arquivos compactados não contêm restrições de tipo de dados, às vezes são utilizados para transportar cargas maliciosas.

Comandos do Sistema - *System Commands*

É uma diretiva de um programa para executar uma tarefa específica e podem ser extraídos dos arquivos `.smali` dos APKs. Como os comandos do sistema podem realizar algumas tarefas especiais, os atacantes utilizam-os em aplicativos maliciosos. Por exemplo, comandos de sistema como `chmod`, `chown`, `mount` e *strings* específicas como `/sys/bin/sh` podem permitir que o *malware* obtenha os privilégios de usuário `root` (super usuário) do dispositivo ou execute *scripts de shell* maliciosos em tempo de execução. Outros tipos de comandos como `pm install` podem ser utilizados para instalação furtiva de pacotes maliciosos adicionais (Yerima et al., 2015).

Código Nativo

Refere-se ao código de programação configurado para ser executado em um processador específico, e que geralmente não é executado em um emulador, a menos que seja permitido. Possibilita ao invasores ocultar partes da funcionalidade de seus aplicativos.

2.2.2 Características Dinâmicas

A extração de características dinâmicas requer a instalação e execução do APK, a coleta de processos de execução do aplicativo, seja em um emulador ou dispositivo físico e alguma interação humana ou automatizada, já que comportamentos maliciosos às vezes são acionados somente após a ocorrência de certos eventos.

A desvantagem é que para se obter as características dinâmicas, de forma significante, leva-se um tempo considerado grande quando se trata de detecção de *malwares*.

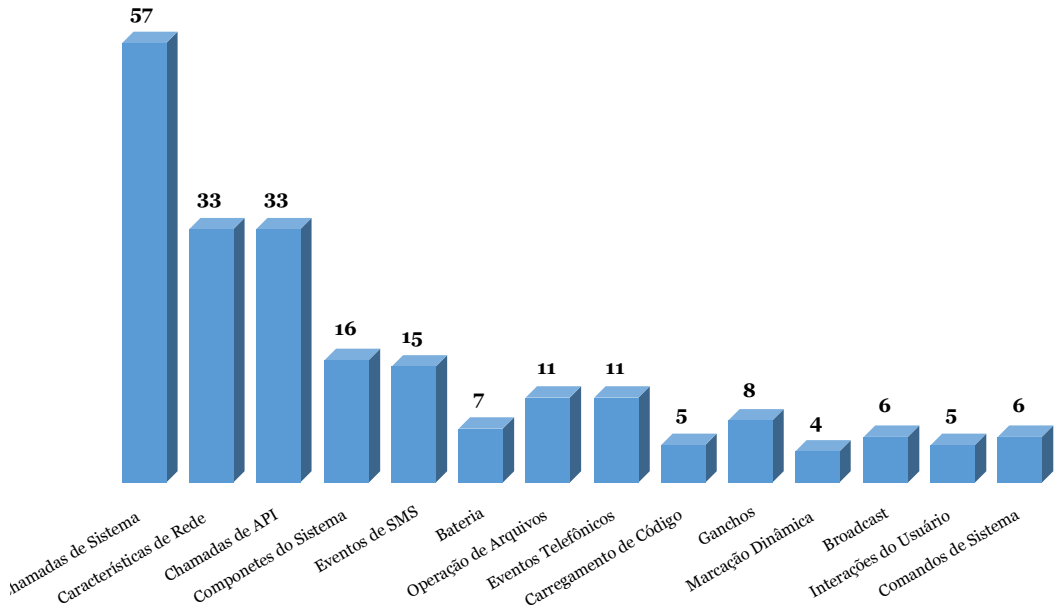


Figura 2.7: Trabalhos Utilizando Características Dinâmicas.

Chamadas de Sistema - *System Calls*

São gerados pela interação de um usuário com o sistema Android por meio de um aplicativo. Existem mais de 250 chamadas de sistema disponíveis no Android. Fornecem funções úteis para aplicativos, como rede, arquivo e outras operações relacionadas. Portanto, ao analisar as chamadas do sistema pode-se obter informações precisas sobre o comportamento dos aplicativos.

Como mostrado na Figura 2.7, as chamadas de sistema são as características mais utilizadas entre as características dinâmicas. Isso ocorre devido ao *Android RunTime* (ART), disponível no sistema operacional Android a partir da versão 5.0, que utiliza um

processo de *sandboxing* para restringir a interação direta dos aplicativos com o `kernel` do sistema operacional Android.

Substituindo o Dalvik, máquina virtual de processos originalmente utilizada pelo Android, o ART traduz o *bytecode* do aplicativo em instruções nativas que são posteriormente executadas pelo ambiente de execução do dispositivo. No ART, aplicativos inteiros são compilados em código de máquina nativo antes da instalação usando compilação antecipada (AOT), que elimina a interpretação do Dalvik e a compilação JIT (*Just in Time*) baseada em rastreamento. A compilação JIT não apenas irá subtrair recursos do programa em execução, mas também precisará de energia da bateria, e precisará disso toda vez que o programa for executado, enquanto um compilador AOT precisará utilizar esse recurso apenas uma vez, quando o aplicativo for instalado. Além disso, o ART fornece execução mais rápida de aplicativos, alocação de memória aprimorada e mecanismos de coleta de lixo, novos recursos de depuração de aplicativos e criação de perfil de alto nível mais precisa (Android Open Source Project, 2023).

Características de Rede

A maioria dos aplicativos precisa se conectar à rede para enviar e receber dados, receber atualizações, entre outros, possibilitando o envio de dados pessoais dos usuários para atacantes através da rede.

No entanto, para utilizar características de rede é preciso lidar com um grande número de registros. Além disso, a análise dos registros de rede coletados requer um entendimento profundo da arquitetura da rede.

Para interpretar corretamente os registros de rede, é necessário compreender a estrutura e o funcionamento da rede em que os dados foram coletados. Por exemplo, é importante entender os diferentes protocolos de rede utilizados e seus respectivos formatos de pacotes, bem como os endereços IP e portas utilizados para o envio e recebimento de dados.

Componentes do Sistema

Os dispositivos móveis têm componentes semelhantes aos dos computadores pessoais, como CPU, memória e armazenamento. Juntos, eles fazem o sistema operacional e o dispositivo móvel funcionarem de maneira correta e eficiente. Podem fornecer informações precisas sobre o comportamento dos aplicativos. Por exemplo, alguns *malwares* ocupam muito tempo de CPU.

Bateria

As características da bateria pode refletir alguns comportamentos dos aplicativos, como fazem os componentes do sistema. Avaliação da tensão, temperatura e estado de descarga (nível da bateria) pode levar a identificação de aplicativos maliciosos.

Eventos Telefônicos

Alguns *malwares* executam ações, como chamadas telefônicas, sem o conhecimento do usuário. Portanto, alguns trabalhos consideram os eventos telefônicos como um tipo de característica dinâmicas.

Eventos de SMS

Existe uma espécie de *malware* que envia ou recebe mensagens SMS. Algumas mensagens SMS contêm links de vírus que podem prejudicar os usuários ou a segurança das informações.

Interações do Usuário

Como os usuários são vítimas potenciais de *malwares*, as atividades dos usuários fazem parte dos comportamentos do aplicativo, como tocar na tela, arrastar e manter pressionado e assim por diante. E uma das soluções possíveis para detecção de *malwares* é analisar a interação dos usuários com os aplicativos.

Operação de Arquivos

Da mesma forma que os computadores pessoais, os dispositivos móveis têm operações de arquivo, como abrir, ler e gravar. As operações de alguns arquivos importantes podem refletir os comportamentos dos aplicativos. Portanto, monitorar as operações de arquivos dos aplicativos é uma maneira eficaz de detectar comportamentos maliciosos.

Diversos indicadores podem ser utilizados como, contagem de arquivos criados, excluídos e modificados, contagem de arquivos criados em caminhos predefinidos como `%APPDATA%, %TEMP% e %PROGRAMFILES%`, tipo de arquivo e assim por diante.

Broadcast

A transmissão (ou *broadcast*) é um mecanismo amplamente usado para transferir informações entre aplicativos. O receptor de transmissão é um componente que filtra e recebe transmissões, podendo ser monitorado para a detecção de *malwares* em Android.

Carregamento de Código

Alguns códigos maliciosos são baixados dinamicamente da Internet durante a execução dos aplicativos. Existem duas situações: as cargas estáticas incluem arquivos `.dex` ou `.jar`, e as cargas dinâmicas contêm arquivos `.elf` ou `.so`. Logo, alguns trabalhos inspecionam os aplicativos Android e adquirem o código de carregamento dinâmico quando o aplicativo está sendo executado em um ambiente virtual ou em um dispositivo real, e então aplicam técnicas de análise estática nesse código.

Ganchos - *Hooks*

Baseado em instrumentação, esse método incorpora pontos de monitoramento (ganchos) ao código do aplicativo para registrar suas atividades durante a execução. Esses ganchos podem ser usados coletando informações, rastreando as instruções executadas, recuperando a sequência de eventos ou monitorando o fluxo de dados armazenados. Pode ser realizada a nível de aplicação ou a nível de sistema operacional.

Marcação Dinâmica - *Dynamic Taint*

Rastreia os fluxos de dados desde as fontes, durante a execução de um aplicativo. Permite a detecção e, consequentemente, a prevenção de vulnerabilidades baseadas em fluxo, como vazamentos de dados ou ataques de injeção. É uma técnica de controle de informação.

2.3 Mineração de Regras de Associação

De acordo com Han et al. (2012), a mineração de dados é um processo para o descobrir conhecimento a partir de uma grande quantidade de dados (*Knowledge Discovery in Databases* - KDD). Para sair dos dados e alcançar o conhecimento é necessário uma divisão em fases pelo fato do processo ser muito extenso. Conforme ilustrado na Figura 2.8 as fases de KDD são:

- **Limpeza dos dados**: remoção de impurezas e dados inconsistentes.
- **Integração dos dados**: é a fase onde diversas bases de dados podem se combinar.
- **Seleção dos dados**: busca de dados relevantes na(s) base(s) de dados.
- **Transformação de dados**: preparação dos dados para a fase de mineração utilizando sumarização, agregação ou técnicas semelhantes.

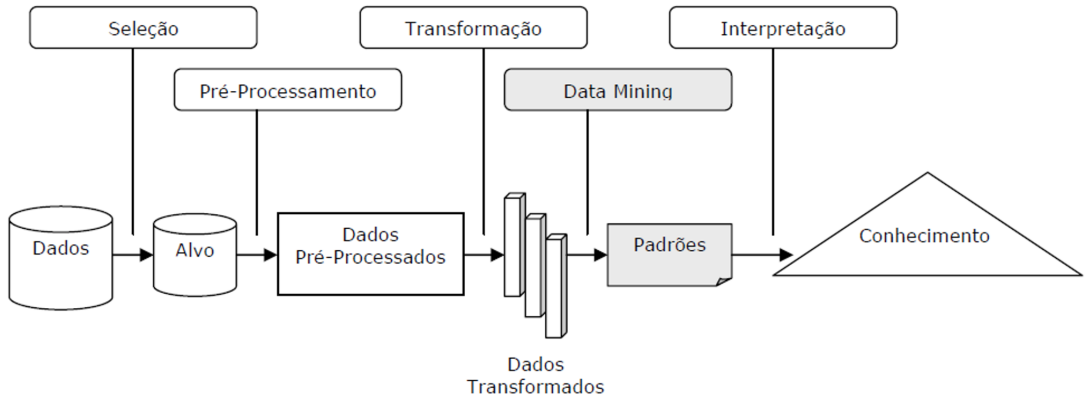


Figura 2.8: Fases do Processo de KDD

- **Mineração dos dados**: aplicação de métodos inteligentes que tem o objetivo de extrair padrões dos dados.
- **Avaliação de padrões**: identificar os padrões que melhor se adaptam ao objetivo baseado em algumas métricas.
- **Apresentação do conhecimento**: utilização de técnicas de visualização para apresentar os resultados obtidos ao usuário.

Neste contexto, a mineração de regras de associação atua na etapa de mineração de dados enquanto a qualificação das regras compreende a avaliação de padrões.

2.3.1 Definição

Para entender melhor a mineração de regras de associação é preciso falar do algoritmo Apriori (Agrawal e Srikant, 1994), o método de mineração de dados bastante eficiente, conhecido e utilizado. Proposto com o objetivo específico de descobrir associações existentes em bases de dados, o algoritmo se baseia no princípio de que há uma grande chance de existir uma associação entre itens presentes entre os dados, se estes aparecem frequentemente juntos. De modo geral, um algoritmo de busca de associações nada mais é que uma engenharia de contagem de correlações existentes.

Já a mineração de regras de associação é uma técnica para identificar relações subjacentes entre diferentes itens. Uma regra de associação baseia-se no princípio de que há uma grande chance de existir uma relação entre itens de um conjunto de dados quando eles aparecem frequentemente juntos. Na prática, essas regras são implementadas por

algoritmos de mineração de regras de associação, ou seja, procedimentos que quantificam as correlações existentes entre os itens desses conjuntos.

2.3.2 Formulação

Segundo Agrawal et al. (1993), podemos formalizar a descoberta de regras de associação como se segue. Seja $I = \{I_1, I_2, I_3, \cdot\cdot, I_m\}$ um conjunto de m atributos distintos, denominado **itens**, t uma transação que contém um conjunto de itens ($t \subseteq I$) e D uma base de dados com diferentes transações t, representadas por um vetor binário, onde $t[k] = 1$ indica a presença do item k em t, e $t[k] = 0$ sua ausência.

Uma **regra de associação** é uma implicação da forma $X \implies Y$, onde X (antecedente) e Y (consequente) são subconjuntos de itens em I ($X \subset I$ e $Y \subset I$), onde X e Y não possuem itens em comum ($X \cap Y = \varnothing$) (Agrawal e Srikant, 1994).

2.3.3 Suporte e Confiança

Para avaliar a importância de determinada regra de associação, diferentes métricas foram desenvolvidas. Assim, dado um conjunto de transações, o problema de mineração por regras de associação está em gerar todas as regras cujas as métricas possuam valores iguais ou superiores aos valores mínimos determinados.

As principais métricas para avaliar a importância de determinada regra de associação são o suporte e a confiança (Agrawal et al., 1993). O **suporte** é usado para medir a frequência, muitas vezes interpretada como significância ou importância, de um conjunto de itens em uma base de dados. Se o suporte de um conjunto de itens for maior do que um limite de suporte mínimo especificado, este será referido como um **conjunto de itens frequentes**.

Matematicamente, dada uma regra $X \implies Y$, o **suporte** é definido como a fração de transações t em D que satisfaz a união dos itens no antecedente e consequente da regra, como pode ser visto na Equação 2.1.

$$Suporte(X \implies Y) = \frac{Frequência(X \cup Y)}{Total\ de\ transações\ t\ em\ D}, [0, 1] \tag{2.1}$$

Como representado na Equação 2.2, a **confiança** de uma regra $X \implies Y$ é a probabilidade do consequente Y estar presente em uma transação, dado que ela também contém o antecedente X.

$$Confiança(X \implies Y) = \frac{Suporte(X \implies Y)}{Suporte(X)}, [0, 1] \tag{2.2}$$

É importante ressaltar que a métrica não é simétrica ou direcionada. Por exemplo, a confiança para $X \implies Y$ é diferente da confiança para $Y \implies X$. A confiança é 1 (máxima) para uma regra $X \implies Y$ se o consequente e o antecedente ocorrerem juntos.

O objetivo da **mineração de regras de associação** é descobrir regras de associação que satisfaçam os valores mínimos preestabelecidos de suporte e confiança. O problema de se descobrir todas as regras de associação pode ser decomposto em duas partes (Agrawal et al., 1993):

- Encontrar os conjuntos de itens que têm **suporte** maior ou igual a um limite mínimo, ou seja encontrar os conjuntos de itens frequentes;
- Gerar as regras de associação a partir dos conjuntos de itens frequentes considerando o limite mínimo para a **confiança**.

2.3.4 Outras Métricas

Existem outras métricas utilizadas para a geração de regras de associação. O **lift**, chamado inicialmente de **interesse** (do inglês *interest*), é utilizado para avaliar o grau de dependência do consequente em relação ao antecedente de uma regra. Indica quão mais frequente torna-se Y quando X ocorre (Brin et al., 1997), correspondendo a:

$$Lift(X \implies Y) = \frac{Confiança(X \implies Y)}{Suporte(Y)}, [0, \infty] \tag{2.3}$$

Se $lift(X \implies Y) = 1$, a ocorrência dos itens do consequente é estatisticamente independente da ocorrência dos itens do antecedente, e vice-versa, e então não há correlação entre eles. Se $lift(X \implies Y) > 1$, então os itens do antecedente e consequente são positivamente dependentes, o que significa que a ocorrência de um provavelmente leva a ocorrência do outro. Se $lift(X \implies Y) < 1$, os itens do antecedente e consequente são negativamente dependentes, ou seja, a ocorrência de um provavelmente leva a ausência do outro.

Outra métrica, definida por Brin et al. (1997), e a **convicção** (Equação 2.4). Um valor de convicção alto significa que o consequente é altamente dependente do antecedente. Por exemplo, no caso de uma pontuação de confiança perfeita, o denominador torna-se 0, para o qual a pontuação de convicção é definida como **inf**. Semelhante ao lift, se os itens forem independentes, a convicção é 1.

$$Convicção(X \implies Y) = \frac{1 - Suporte(Y)}{1 - Confiança(X \implies Y)}, [0, \infty] \tag{2.4}$$

A métrica de alavancagem (*leverage*, em inglês) calcula a diferença entre o suporte observado de X e Y aparecendo juntos e o suporte que seria esperado se X e Y fossem independentes. Um valor de alavancagem de 0 indica independência (Piatetsky-Shapiro, 1991). Ou seja:

$$\begin{aligned} Alavancagem(X \implies Y) = Suporte(X \implies Y) \\ - (Suporte(X) \times Suporte(Y)), [-1, 1] \end{aligned} \tag{2.5}$$

2.4 Algoritmos de Mineração de Regras de Associação

Os algoritmos clássicos para mineração de regras de associação, como Apriori, FP-Growth e ECLAT, são recorrentemente referenciados e utilizados na literatura (Li e Sheu, 2021; Zhang e He, 2010). A partir destes algoritmos base, surgiram diversos métodos especializados em classificação baseada em regras de associação, como CBA (Liu et al., 1998), CMAR (Li et al., 2001), CPAR (Yin e Han, 2003), MCAR (Thabtah et al., 2005) e ARCID (Abdellatif et al., 2018). Esses métodos incorporam e/ou evoluem os algoritmos clássicos, como por exemplo, o CMAR, que incrementa o FP-Growth para minerar grandes conjuntos de dados e assim melhorar a precisão e a eficiência da classificação.

2.4.1 Apriori

O algoritmo Apriori (Agrawal e Srikant, 1994) é um dos algoritmos mais conhecidos para mineração de regras de associação. O algoritmo emprega busca em profundidade e gera conjuntos de itens candidatos (padrões) de k elementos a partir de conjuntos de itens de $k-1$ elementos. Os padrões não frequentes são eliminados. Toda a base de dados é rastreada e os conjuntos de itens frequentes são obtidos a partir dos conjuntos de itens candidatos.

O Algoritmo 1 mostra o algoritmo Apriori, baseado nos seguintes teoremas:

1. Se um conjunto de itens é frequente, então todos os seus **subconjuntos** também são frequentes.

2. Se um conjunto de itens não é frequente, então todos os seus **superconjuntos** também são não frequentes

A primeira passagem do algoritmo simplesmente conta as ocorrências de itens para determinar os conjuntos de itens frequentes de tamanho 1. As passagens subsequente

Algoritmo 1: Apriori

Entrada:
D: banco de dados de transações
$minSup$: suporte mínimo

```
1 L_1: conjunto de itens frequentes de tamanho 1
2 para k ← 2; L_{k-1} ≠ ∅; k++ faça
3     C_k ← apriori-gen(L_{k-1});                    // novos candidatos
4     para cada transação t ∈ D faça
5         C_t ← subset(C_k, t);                      // candidatos contidos em t
6         para cada candidato c ∈ C_t faça
7             c.suporte++;                           // incrementa o suporte de c
8     L_k ←{c ∈ C_t∧ c.suporte ≥ minSup}
9 retorna ⋃_k L_k;                                   // união dos conjuntos de itens frequentes
```

consistem em duas fases. Primeiro, os conjuntos de itens frequentes L_{k-1} encontrados na $(k-1)$-ésima passagem são usados para gerar os conjuntos de itens candidatos C_k, usando a função **apriori-gen** descrita na Algoritmo 2. Em seguida, o banco de dados é verificado e o suporte dos candidatos em C_k é contado. Para uma contagem rápida, é preciso determinar com eficiência os candidatos em C_k que estão contidos em uma determinada transação t. A função **subset** é usada para este propósito.

Algoritmo 2: apriori-gen

Entrada:
L_{k-1}: conjunto de todos os $(k-1)$-itemset frequentes

```
1  C_k ← ∅
2  para cada conjunto de item p ∈ L_{k-1} faça
3      para cada conjunto de item q ∈ L_{k-1} faça
4          se p[1] = q[1] ∧ ... ∧ p[k-2] = q[k-2] ∧ p[k-1] < q[k-1] então
5              c ← p ∪ q ;                          // junção: geração de candidato
6              adicione c a C_k
7  para cada conjunto de item c ∈ C_k faça
8      para cada subconjunto de (k-1)-itens s ⊂ c faça
9          se s ∉ L_{k-1} então
10             delete c de C_k;                     // poda: remove candidatos inutéis
11 retorna C_k
```

A função **apriori-gen** tem como argumento L_{k-1}, o conjunto de todos os $(k-1)$-

itemset frequentes. Retorna um conjunto de todos os k-itemset frequentes. Primeiro, na etapa de *junção*, obtemos o produto cartesiano C_k de $L_{k=1}$ com $L_{k=1}$. A condição $p[k-1] < q[k-1]$ garante que nenhum conjunto de itens duplicados seja gerado. Em seguida, na etapa de *poda*, excluímos todos os conjuntos de itens $c \in C_k$ se algum subconjunto $k-1$ de c não estiver em L_{k-1}.

Exemplo: Seja $L_3 = \{\{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{1, 3, 5\}, \{2, 3, 4\}\}$. Após a etapa de *junção*, C_4 será $\{\{1, 2, 3, 4\}, \{1, 3, 4, 5\}\}$. A etapa de *poda* excluirá o conjunto de itens $\{1, 3, 4, 5\}$ porque o subconjunto de itens $\{1, 4, 5\}$ não está em L_3. Então, ficaremos apenas com $\{1, 2, 3, 4\}$ em C_4.

O algoritmo Apriori foi o primeiro algoritmo de mineração de regras de associação eficiente. Ele incorpora várias técnicas para acelerar o processo, bem como para reduzir o uso de memória. Por exemplo, a etapa de *junção* usado no processo de geração de candidatos pode reduzir o número de candidatos gerados e o processo de poda pode reduzir significativamente o número de candidatos possíveis em cada nível.

Um dos mecanismos mais importantes no algoritmo Apriori é o uso da estrutura de dados em árvore *hash*. Ele usa essa estrutura de dados na fase de contagem de suporte candidato para reduzir a complexidade de tempo.

A principal vantagem do algoritmo Apriori vem de seu uso de memória, porque apenas os $k-1$ conjuntos de itens frequentes, L_{k-1}, e os candidatos no nível k, C_k, precisam ser armazenados na memória. Ele gera o número mínimo de candidatos com base na etapas de *junção* e *poda*, e os armazena na estrutura de arvore *hash* compacta.

O Apriori pode extrair as regras dos bancos de dados com eficiência, mas custa muito tempo e sua eficiência não é muito alta. Alcança um bom desempenho ao reduzir o tamanho dos conjuntos candidatos. No entanto, é demasiadamente custoso lidar com um grande número de conjuntos de candidatos que possuam padrões frequentes (Zhang e He, 2010).

O processo do Apriori, com suporte em 30%, é apresentado na Tabela 2.1

2.4.2 FP-Growth

O algoritmo *Frequent Pattern Growth* (Han et al., 2000) usa uma estrutura semelhante a uma árvore (chamada de árvore de padrão frequente ou FP-Tree) para encontrar os conjuntos de itens frequentes. O método de geração de candidatos encontra os candidatos dos conjuntos de itens frequentes antes de reduzi-los aos conjuntos de itens frequentes reais por meio da contagem de suporte.

O algoritmo primeiro verifica um conjunto de dados e encontra os conjuntos de itens

Tabela 2.1: Apriori - Processo de Mineração (Zhao e Bhowmick, 2003)

TID	**Transações**
T01	I_1, I_2, I_5
T02	I_2, I_4
T03	I_2, I_3
T04	I_1, I_2, I_4
T05	I_1, I_3
T06	I_2, I_3
T07	I_1, I_3
T08	I_1, I_2, I_3, I_5
T09	I_1, I_2, I_3
T10	I_1, I_2, I_5, I_6

(a) Banco de Dados Original

Itens	**Suporte**
I_1	7
I_2	8
I_3	6
I_4	2
I_5	3
I_6	1

(b) C_1

Itens Frequentes ($k = 1$)
I_1
I_2
I_3
I_5

(c) L_1

Itens	**Suporte**
I_1, I_2	5
I_1, I_3	4
I_1, I_5	3
I_2, I_3	4
I_2, I_5	3
I_3, I_5	1

(d) C_2

Itens Frequentes ($k = 2$)
I_1, I_2
I_1, I_3
I_1, I_5
I_2, I_3
I_2, I_5

(e) L_2

Itens	**Suporte**
I_1, I_2, I_3	2
I_1, I_2, I_5	3

(f) C_3

Itens Frequentes ($k = 3$)
I_1, I_2, I_5

(g) L_3

mais frequentes. Em seguida, uma árvore de padrão frequente é construída examinando o conjunto de dados novamente. Os itens são adicionados à árvore na ordem de seu suporte. Uma vez que a árvore é completada, ela é percorrida de baixo para cima e uma FP-Tree condicional é gerada. Finalmente, o algoritmo gera os conjuntos de itens

frequentes.

O algoritmo FP-Growth é mais escalável do que o algoritmo Apriori na maioria dos casos, uma vez que faz menos passagens e não requer geração de candidatos. No entanto, ele sofre de limitações de memória, uma vez que a FP-Tree é bastante complexa e pode não caber na memória. Percorrer a FP-Tree também pode ser demorado se esta não for compacta o suficiente (Li e Sheu, 2021).

2.4.3 ECLAT

Diferente do algoritmo Apriori e do algoritmo FP-Growth que funcionam em conjuntos de dados horizontais (por exemplo, T01: $\{I_1, I_2, I_3\}$, T02: $\{I_2, I_4\}$), o algoritmo ECLAT (*Equivalence Class Clustering and bottom-up Lattice Traversal*) (Zaki, 2000) usa um conjunto de dados vertical (por exemplo, I_1: {T01, T02}, I_2: {T01, T02}, I_3: {T01}, I_4: {T02}). O algoritmo ECLAT verifica o conjunto de dados apenas uma vez, utilizando uma estratégia que combina a busca em profundidade com interseções entre conjuntos.

A ideia é manter em memória apenas a lista dos identificadores de transações (*TIDlist*) dos itens frequentes em análise, sem precisar dividir o conjunto de dados. Quando uma intersecção entre duas *TIDlist* é realizada, a *TIDList* resultante é composta pelos itens frequentes de maior suporte. Além disso, quando o suporte mínimo não é atingido, o processo de intersecção é interrompido.

No entanto, quando o conjunto de dados é grande e o suporte mínimo é definido com um valor baixo, a *TIDList* associada a cada item frequente pode se tornar muito longa e não caber na memória.

2.4.4 Outros Algoritmos de Mineração de Regras de Associação

Existem três categorias de algoritmos de mineração de regras de associação/mineração de conjuntos de itens frequentes (Chee et al., 2019): Algoritmos baseados em Apriori, algoritmos baseados em árvore e algoritmos de crescimento de padrão. Os algoritmos Apriori, ECLAT e o FP-Growth são os algoritmos mais populares para as três categorias, respectivamente.

Na categoria de algoritmo baseado em Apriori, o algoritmo AprioriTID (Agrawal e Srikant, 1994) é semelhante ao Apriori. O AprioriTID também usa a função **apriori-gen** para determinar os conjuntos de itens candidatos, mas a diferença é que a base de dados não é usada para a contagem de suporte após a primeira passagem. Em vez disso, o conjunto de conjuntos de itens candidatos C_k é usado para esse propósito.

O algoritmo Apriori Hybrid (Agrawal e Srikant, 1994) é uma combinação do algoritmo Apriori e do algoritmo AprioriTID. O algoritmo DHP (*Direct Hashing and Pruning*) (Park et al., 1995) usa uma função *hash* para distribuir os conjuntos de itens. Os algoritmos MR-Apriori (Lin, 2014) e HP-Apriori (Nadimi-Shahraki e Mansouri, 2017) são versões distribuídas do algoritmo Apriori. O MR-Apriori usa o modelo MapReduce na plataforma Hadoop. Eles permitem a execução paralela do algoritmo Apriori.

Os algoritmos baseados em árvore, representados pelo algoritmo ECLAT, encontram o conjunto de itens frequente construindo uma árvore lexicográfica. O algoritmo AIS (Agrawal et al., 1993) e o algoritmo SETM (Houtsma e Swami, 1995) são os dois primeiros algoritmos de mineração de regras de associação nesta categoria. O algoritmo Tree-Projection (Agarwal et al., 2001) conta os suportes dos conjuntos de itens frequentes e usa os nós de uma árvore lexicográfica como a representação desses números de suporte. O algoritmo TM (Song e Rajasekaran, 2006) mapeia o identificador de cada transação para intervalos de transação antes de realizar interseções entre esses intervalos.

Por último, os algoritmos na categoria de crescimento de padrão se concentram em padrões frequentes. O algoritmo P-Mine (Baralis et al., 2013) é um algoritmo de computação paralela que utiliza a estrutura de dados VLDBMine para armazenar o conjunto de dados e acelerar a distribuição dos dados, enquanto o algoritmo LP-Growth (Pyun et al., 2014) faz uso de uma árvore de prefixo linear baseada em *array* para melhorar a eficiência da memória. O algoritmo Can-Mining (Hoseini et al., 2015) encontra os conjuntos de itens frequentes de uma árvore de ordem canônica, o que acelera o processo de travessia da árvore quando o número de conjuntos de itens frequentes é baixo. Finalmente, o algoritmo EXTRACT (Feddaoui et al., 2016) usa a teoria da rede de Galois para derivar regras de associação.

2.5 Qualidade de Regras

O objetivo da qualidade de regras é encontrar um conjunto simples de regras que explique os dados do conjunto, inclusive os dados invisíveis. Ela pode ser alcançada através da otimização de dois critérios simultaneamente (Janssen e Fürnkranz, 2010):

- **Cobertura**: o número de amostras positivas que são abrangidas pela regra deve ser maximizado.

- **Consistência**: o número de amostras negativas que são cobertos pela regra deve ser minimizado.

Vale destacar que no escopo da detecção de *malwares*, a amostra positiva é maliciosa, enquanto que a amostra negativa é benigna.

Muitas métricas de qualidade de regras são derivadas da análise da relação entre uma regra R e uma classe C. Esta relação pode ser representada por uma **tabela de contingência** (Arkin e Colton, 1970; Bruha e Kockova, 1993) que consiste em uma tabulação cruzada de categorias de observações, conforme mostrado na Tabela 2.2, onde p é o número de amostras cobertas pela regra R e pertencentes a classe C; n é o número de amostras cobertas pela regra R e não pertencentes a classe C.

Tabela 2.2: Tabela de Contingência

Amostras	Pertencentes a Classe C	Não Pertencentes a Classe C
Cobertas Pela Regra R	p	n
Não Cobertas Pela Regra R	$\overline{p}$	$\overline{n}$
Total de Amostras	P	N

Assim, cada regra pode ser caracterizada por:

- p e n: as amostras abrangidas pela regra.
- P e N: o número total de amostras no conjunto de dados.

Consequentemente, a maioria das métricas de qualidade de regras dependem de p, n, P e N, combinando esses valores de maneiras diferentes.

Como exemplos de métricas de qualidade de regras podemos citar aquelas encontradas em Lenca et al. (2007), Janssen e Fürnkranz (2010), Wróbel et al. (2016) e suas referências.

Nossa escolha de métricas de qualidade de regra é apresentada na Tabela 2.3.

- *Kappa (Kap)*: concordância entre as classes reais e as classes previstas por um classificador.
- *Acurácia (Acc)*: porcentagem das amostras classificadas corretamente entre todas as amostras do conjunto de dados.
- *Cobertura (Cov)*: fração de amostras cobertas pela regra. O valor máximo é alcançado quando abrange todas as amostras avaliadas.

Tabela 2.3: Métricas de Qualidade de Regras

Métrica	Formulação	Referência
Kap	$2 \times \frac{(p \times N) - (P \times n)}{((P+N) \times (p+n+P)) - (2 \times (p+n) \times P)}$	Lenca et al. (2007)
Acc	$\frac{p + (N-n)}{P+N}$	Janssen e Fürnkranz (2010)
Cov	$\frac{p+n}{P+N}$	
Prec	$\frac{p}{p+n}$	
BC	$\frac{p}{p+N} - \frac{P-p}{P-p+N-n}$	Wróbel et al. (2016)
C1	$Coleman \times \frac{2+Cohen}{3}$	
C2	$Coleman \times 0{,}5 \times (1 + \frac{p}{P})$	
Corr	$\frac{(p \times N) - (P \times n)}{\sqrt{P \times N \times (p+n) \times (P-p+N-n)}}$	

- *Precisão (Prec)*: fração das amostras classificadas corretamente (p) entre todos as amostras cobertas ($p+n$) pela regra.
- *Confirmação Bayesiana (BC)*: proposta por Christensen (1999) e Joyce (1999) como uma medida de confirmação. O primeiro componente da medida avalia a precisão da regra e o segundo é responsável pela diminuição da qualidade das regras que abrangem um pequeno número de amostras positivas.
- *C1* e *C2*: combinam duas métricas de qualidade conhecidas como métrica de *Coleman* e *Cohen* (Tabela 2.4). A métrica de *Coleman* prioriza regras com alta precisão e baixa cobertura, enquanto a medida de *Cohen* prioriza regras com alta cobertura.
- *Correlação (Corr)*: coeficiente de correlação entre as classes previstas e as classes

reais. É aplicado a algoritmos de indução de regra de classificação, bem como à descoberta de subgrupo e avaliação de regras de associação (Geng e Hamilton, 2006; Janssen e Fürnkranz, 2010).

Tabela 2.4: Métricas de Coleman e Cohen

Métrica	Formulação
Coleman	$\frac{(P+N) \times (\frac{p}{p+n}) - P}{N}$
Cohen	$\frac{(P+N) \times (\frac{p}{p+n}) - P}{(\frac{P+N}{2}) \times (\frac{p+n+P}{p+n}) - P}$

Capítulo 3

Trabalhos Relacionados

Neste capítulo apresentamos trabalhos que propuseram a detecção de aplicativos maliciosos ou que usam regras de associação. Primeiramente, descrevemos trabalhos cujas soluções englobam diferentes ideias, conforme apresentado no trabalho de (Wang et al., 2019). Em seguida, apresentamos trabalhos alguns trabalhos que fazem uso da mineração de regras de associação, mas no contexto diferente de *malware* Android. Vale destacar que este trabalho é o primeiro empregar a mineração de regras de associação no contexto de detecção *malware* Android. Entretanto, Adebayo e Abdul Aziz (2019) e Sun et al. (2016) utilizaram a mineração de regras de associação em seus trabalhos, este para encontrar permissões que ocorrem juntas, removendo a de menor suporte; e aquele associada ao PSO (*Particle Swarm Optimization*), para extrair assinaturas das amostras coletadas, removendo aquelas características consideradas desnecessárias.

3.1 Detecção de Aplicativos Maliciosos

Estratégias de detecção baseadas na avaliação de características são bastante comuns e usuais na literatura. Os trabalhos de Wang et al. (2017), Li et al. (2018), Yildiz e Doğru (2019) e Alsoghyer e Almomani (2020) introduziram técnicas que utilizam as permissões solicitadas pelos aplicativos para detecção. A ideia é aproveitar o fato de que o *malware* muitas vezes tende a solicitar permissões perigosas ou desnecessárias. No entanto, essa abordagem pode estar sujeita a falsos positivos, pois aplicativos benignos também podem solicitar permissões perigosas (Jiang et al., 2020). Além disso, desde o Android 6.0, o modelo de permissão permite que os usuários concedam permissões em tempo de execução quando são solicitadas pela primeira vez. Portanto, algumas permissões perigosas podem nunca ser realmente concedidas.

Outros trabalhos se baseiam no uso ou na sequência de chamadas de API para detecção de *malware*. Scalas et al. (2019) classifica os aplicativos baseando-se nas chamadas de API utilizadas para realizar suas ações, permitindo a distinção entre *malware* genérico, *ransomware* e aplicativos benignos. TriFlow (Mirzaei et al., 2017) classifica os aplicativos com base nos riscos potenciais usando os fluxos, possíveis e observados, de informação nos aplicativos. Zou et al. (2021) apresenta o IntDroid, um método para realizar a detecção de *malware* Android, analisando as relações entre chamadas API sensíveis e nós centrais dentro de gráficos de chamadas de função, enquanto o SigAPI (Galib e Hossain, 2020) propõe e avalia uma abordagem de redução de características para identificar chamadas API significativas na detecção de *malware* do Android de forma eficaz. Cabe destacar que, devido às mudanças na API do Android, bem como à evolução dos *malwares*, os métodos baseados no uso de chamadas de API específicas requerem retreinamento frequente conforme novas versões da API são lançadas e novos tipos de *malware* são desenvolvidos. A suspensão de uso e/ou adição de chamadas de API é bastante comum, e isso pode demandar alterações nos aplicativos por parte dos desenvolvedores.

Os trabalhos apresentados até aqui funcionam aprendendo os padrões de comportamento dos *malwares* de modo *offline* ou estática, onde tipicamente um modelo[1] treinado com este tipo de comportamento é utilizado para identificar *malwares*. Tipicamente, as características utilizadas são extraídas do manifesto do aplicativo (permissões, *intents*, componentes de hardware, componentes do aplicativo) e/ou do *byte code* do aplicativo (chamadas de API restritas e suspeitas, endereços de rede, permissões, entre outros).

De forma geral, o comportamento malicioso é refletido em padrões e combinações das características extraídas. Por exemplo, a existência da permissão `SEND_SMS` e do componente `android.hardware.telephony` em um aplicativo pode indicar uma tentativa de enviar mensagens SMS para serviços pagos, e essa combinação pode eventualmente constituir um padrão de detecção. Infelizmente, técnicas baseadas em código descompilado podem ser evitadas usando carregamento dinâmico de código, reflexão e o uso de código nativo.

No outro lado do processo de detecção, estão os trabalhos que realizam análise dinâmica Guerra-Manzanares et al. (2021); Malik e Khatter (2016); Martín et al. (2018). A ideia deste tipo de detecção é a existência de vários pontos de entrada durante a execução de um aplicativo Android (por exemplo, por meio de um *activity*, *services* ou

[1] A palavra modelo tem um sentido mais amplo do que um modelo elaborado por um classificador de aprendizagem de máquina.

broadcast).

A análise dinâmica permite examinar o comportamento de um aplicativo em tempo de execução. Essa técnica é essencial para identificar vulnerabilidades e comportamentos maliciosos que não podem ser facilmente observados por meio da análise estática.

Durante a análise dinâmica, um aplicativo é executado em um ambiente onde é possível observar suas interações com o sistema operacional e outros aplicativos. É possível monitorar as atividades do aplicativo, como chamadas a APIs, criação e manipulação de arquivos, acesso à rede, interação com o usuário e muito mais, possibilitando a identificação de atividades maliciosas.

Existem várias ferramentas disponíveis para a análise dinâmica de aplicativos Android, como o emulador Android, o dispositivo virtual Genymotion[2], e ferramentas de automação de teste, como Monkey[3] e AppCrawler[4].

O Monkey é uma ferramenta desenvolvida pelo Google que gera eventos aleatórios do sistema, como toques na tela, gestos, pressões de botões e rolagem, para testar a estabilidade do aplicativo. Já o AppCrawler é uma ferramenta que simula a interação do usuário com o aplicativo. A ferramenta automatiza o processo de testes de usabilidade e gera relatórios de *feedback* que ajudam a identificar problemas de navegação, fluxo de trabalho, legibilidade, entre outros aspectos. Ambas são úteis para testar aplicativos Android em diferentes cenários.

Por fim, existem os soluções que combinam análise estática e dinâmica, por exemplo, usando a primeira para analisar um APK e o último para determinar quais caminhos de execução percorrer, ou combinando as características extraídas. Lindorfer et al. (2014) apresentam o **Andrubis**, que é um *sandbox* de análise de *malwares* que extrai permissões, *services*, *broadcast receivers*, *activities*, nome do pacote e versão do SDK do manifesto de um aplicativo. Em seguida, ele cria um perfil comportamental utilizando diversas características.

Em Saracino et al. (2018) são extraídos cinco grupos de características (chamadas do sistema, chamadas API sensíveis, SMS, atividade do usuário e metadados do aplicativo) em quatro camadas diferentes - kernel, aplicativo, usuário e pacote - que são usados para construir um modelo comportamental para aplicativos e utiliza dois classificadores paralelos para detectar *malwares*.

Alzaylaee et al. (2017) implementou um sistema híbrido integrando a ferramenta

[2] https://www.genymotion.com/
[3] https://developer.android.com/studio/test/other-testing-tools/monkey
[4] https://developer.android.com/studio/test/other-testing-tools/app-crawler

Monkey com a ferramenta DroidBot[5] para melhorar a cobertura de código e descobrir comportamentos maliciosos em potencial. De acordo como (Xu et al., 2019), as famílias de *malwares* compartilham grafos de fluxo de controle (CFG) e grafos de fluxo de dados (DFG) semelhantes. Geralmente, um CFG reflete o que um programa pretende fazer (por exemplo, opcodes) e como ele se comporta (por exemplo, possíveis caminhos de execução). Por outro lado, um DFG representa as dependências de dados entre várias operações e, portanto, pode ajudar na detecção de *malware* envolvendo dados confidenciais ou de rede.

Há trabalhos que utilizam a presença ou ausência de arquivos como características. Varsha et al. (2015) escolheu o número de arquivos `SMALI` como característica, enquanto Lindorfer et al. (2015) verificou presença de arquivos suspeitos, como bibliotecas nativas (compartilhadas), executáveis e *scripts shell* embutidos nos APKs. Surendran et al. (2021) propõe um mecanismo detecção de *malwares*, onde as chamadas de sistema geradas por cada aplicativo são capturadas em um arquivo de log usando uma ferramenta chamada *STrace*.

3.2 Uso de Regras de Associação

Em Tightiz et al. (2020) é apresentado um método inteligente para diagnóstico e classificação de falhas em transformadores de potência com base nos atributos instrutivos do Método de Análise de Gás Dissolvido (DGAM) e algoritmos de aprendizagem de máquina. No método proposto, 14 atributos obtidos através do DGAM são utilizados como entradas iniciais e não processadas do *Adaptive Neuro-Fuzzy Inference System* (ANFIS). Neste método de detecção e classificação de falhas, os atributos mais significativos são selecionados por regras de associação. O uso de atributos de iluminação eficientes e a eliminação de atributos tautológicos levam a uma maior precisão e operação superior. Além disso, o treino do ANFIS, por atributos adequados, tem efeito significativo em sua precisão e robustez. Os resultados mostram que o sistema de diagnóstico proposto possui alta precisão, desempenho robusto e curto tempo de execução e que a seleção dos atributos mais relevantes do DGAM, melhoraram a robustez do ANFIS e aumentam a precisão da classificação de falhas em transformadores de potência.

Ed-Daoudy e Maalmi (2020) demonstra a aplicação de regras de associação no conjunto de dados de câncer de mama *Wisconsin Breast Cancer Diagnostic* (WBCD) do repositório de aprendizagem de máquina Irvine da Universidade da Califórnia, eliminado as características insignificantes. A taxa de classificação correta obtida com o modelo

[5] https://github.com/honeynet/droidbot

utilizando o SVM com regras de associação apresenta precisão de até 98%. Os resultados mostram que a abordagem proposta pode ser usada para reduzir o espaço de atributos e economizar tempo durante a fase de treino dos modelos, levando a uma melhor precisão e sistemas de classificação mais rápidos.

Liu et al. (2003) propõe uma técnica eficaz e eficiente para pontuar os dados usando regras de associação. Chamada de *Scoring Based on Associations* (SBA), a técnica utiliza valores de suporte e confiança mínimos diferentes para cada classe, onde o valor de suporte mínimo é estabelecido de acordo com a distribuição das classes no conjunto de dados, permitindo atribuir uma estimativa de probabilidade de classe mais precisa.

Em Yagin et al. (2021), um estudo comparativo de desempenho de métodos de classificação para previsão de câncer do colo do útero foi conduzido, usando o conjunto de dados de acesso aberto chamado "*Cervical Cancer Behavioral Risk Data Set*". Foi alcançada acurácia de 98,6% ao utilizar o método de classificação baseado em regras de associação, superando os resultados dos demais métodos avaliados.

3.3 Aprendizagem de Máquina

Aprendizagem de máquina é um subconjunto de técnicas de inteligência artificial, que aplica algoritmos para extrair padrões usando matemática, estatística, otimização e métodos de descoberta de conhecimento. Consiste em três categorias principais de aprendizagem: (1) supervisionada, (2) não supervisionado e (3) por reforço (Bishop e Nasrabadi, 2006).

A aprendizagem supervisionada, que é a categoria de aprendizagem de máquina mais conhecida, tenta encontrar relações entre um conjunto de entradas e saídas fornecidas para treinar o modelo (Verbraeken et al., 2020). Podem ser divididos em duas técnicas principais: a classificação, que é utilizada para modelar problemas discretos, e a regressão, que é utilizada para modelar problemas contínuos. Ambas as técnicas têm como objetivo realizar predições, mas diferem nas variáveis que são consideradas como resposta. Na classificação, a variável resposta é composta por categorias ou rótulos de classe, como em problemas de classificação binária ou multiclasse. Já na regressão, a variável resposta é contínua (Telikani et al., 2021).

No escopo da detecção de *malwares* Android, os modelos de aprendizagem de máquina mais utilizados são *Random Forest* (RF) e *Support Vector Machine* (SVM) (Kouliaridis e Kambourakis, 2021; Liu et al., 2020).

O RF usa uma abordagem aleatória para construir árvores de decisão onde cada árvore é treinada com amostras e atributos selecionados aleatoriamente. Então, com

base no resultado obtido para cada árvore, pode-se obter uma predição. O resultado pode ser formado de várias maneiras, como por meio de uma votação majoritária rápida. Usando essa abordagem aleatória, é possível reduzir a dispersão das predições do modelo (Stepanov et al., 2020).

O SVM baseia-se na criação de um hiperplano ótimo, também conhecido como limite de decisão ou limite ideal, que maximiza a distância entre as amostras mais próximas ao plano, conhecidos como vetores de suporte, e efetivamente separa as classes. O modelo busca encontrar o hiperplano ótimo de separação entre classes focando nos casos de treino que ocorrem na borda das distribuições de classe (Adugna et al., 2022).

3.4 Discussão

A detecção de aplicativos maliciosos em dispositivos Android é um processo crítico para garantir a segurança dos usuários. Existem diversas técnicas que podem ser utilizadas para detectar aplicativos maliciosos, incluindo análise de código, análise comportamental, verificação de assinaturas digitais e uso de regras de associação.

Por outro lado, a mineração de as regras de associação é uma técnica que identifica padrões frequentes. Esses padrões podem muito bem ser utilizados para identificar comportamentos suspeitos em aplicativos Android. Por exemplo, é possível identificar padrões de permissões excessivas ou inesperadas solicitadas pelos aplicativos, como acesso à câmera, microfone, mensagens de texto ou localização, sem uma razão clara. Essas permissões podem ser usadas por *malwares* para acessar dados sensíveis do usuário ou para espionar o dispositivo.

Além disso, as regras de associação também podem ser usadas para identificar padrões de comunicação suspeitos entre aplicativos Android e servidores remotos. Por exemplo, é possível identificar quando um aplicativo está enviando informações do dispositivo para um servidor remoto sem a permissão do usuário. Esses padrões de comunicação podem indicar que o aplicativo está coletando dados do usuário ou executando outras atividades maliciosas.

É importante ressaltar que as regras de associação podem não ser uma solução completa para a detecção de *malwares*. Além do mais, é possível utilizá-la em conjunto com outras técnicas, para garantir uma detecção mais completa e precisa de aplicativos maliciosos.

Em resumo, as regras de associação podem ser uma técnica útil para detectar aplicativos maliciosos em dispositivos Android, carecendo de análises cuidadosas para garantir uma detecção mais precisa e redução de falsos positivos.

Capítulo 4

Método Proposto

A detecção de *malware* pode ser encarada como um problema de classificação, cujo objetivo é categorizar os rótulos de uma classe. De acordo com Han et al. (2012), a classificação é uma forma de análise de dados que extrai modelos que descrevem classes importantes de dados. A classificação possui inúmeras aplicações, incluindo detecção de fraude, marketing alvo, previsão de desempenho, diagnóstico médico, entre outros.

Para o método proposto, utilizando-se dos conceitos apresentados na Seção 2.3.2, o conjunto de itens $I = \{I_1, I_2, I_3, ..., I_m\}$ seria o conjunto de todas As características extraídas dos aplicativos, com cada transação $t \in D$ representando as características de um único aplicativo, onde $t[k] = 1$ indica que o aplicativo possui a característica I_k e $t[k] = 0$, caso não a possua (Figura 4.1).

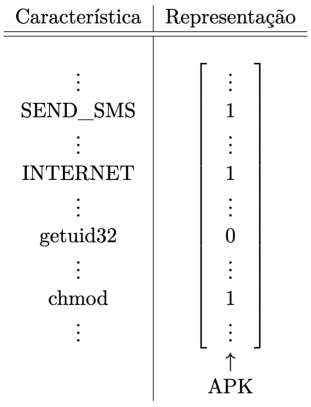


Figura 4.1: Representação do APK: Vetor Binário de Características

4.1 EQAR (*ECLAT and Qualified Association Rules*)

Nesta pesquisa, desenvolvemos um método, denominado EQAR, que incorpora o algoritmo clássico ECLAT e funções de qualidade de regras para melhorar a eficácia e eficiência em problemas de classificação.

O EQAR, disponível no GitHub[1], cujas etapas são ilustradas na Figura 4.2, tem como principal objetivo gerar conjuntos de regras utilizando o ECLAT. Através dos valores mínimos de **suporte** e **confiança**, o método gera os relacionamentos mais significativos entres as características que atendam aos valores apresentados. Posteriormente, esses conjuntos de regras são qualificados para se obter aqueles que melhor caracterizam os aplicativos utilizados na aprendizagem do método.

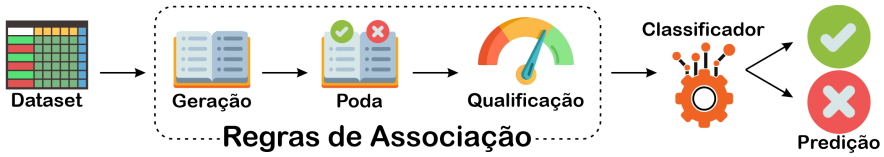


Figura 4.2: Etapas do Método EQAR

Vale destacar que o EQAR tem suporte aos algoritmos Apriori, FP-Growth e ECLAT, mas empregamos este último por ser o mais rápido em termos de tempo de execução (Gayathri, 2017; Heaton, 2016; Sinha e Ghosh, 2014; Vani, 2015). O algoritmo Apriori é o menos eficiente, porque precisa percorrer o conjunto de dados muitas vezes. Já o FP-Growth também tem um desempenho ruim em dados de transações longas, pois a profundidade de árvore (FP-Tree) é diretamente proporcional ao tamanho das transações. Cabe ressaltar que todos os algoritmos (Apriori, FP-Growth e ECLAT) geram as mesmas regras, com a biblioteca utilizada, quando executados com os parâmetros iguais.

A seguir serão detalhadas as etapas do método EQAR.

4.1.1 Geração e Poda de Regras de Associação

O Algoritmo 3 resume o processo de geração e poda das regras de associação do EQAR. Inicialmente, o conjunto de dados utilizado para treino é dividido entre amostras maliciosas e benignas (linha 2). As regras de associação são então geradas separadamente para obter as combinações de características que demonstrem o comportamento das amostras em cada classe (linha 3). Para isso, a função `gerarRegras` utiliza o ECLAT

[1] https://github.com/vanderson-rocha/eqar

com os parâmetros informados, selecionando os antecedentes das regras nas quais o consequente seja a classe em análise.

Algoritmo 3: Geração e Poda de Regras de Associação

Entrada:
D: conjunto de dados
minSup: suporte mínimo
minConf: confiança mínima

1 **para cada** *classe* $C \in$ *{benigno, malicioso}* **faça**
2 datasetClasse $\leftarrow t \in D \wedge t[classe] = C$
3 regras[C] $\leftarrow$ `gerarRegras(`*datasetClasse, minSup, minConf*`)`
4 regrasInterseccao $\leftarrow$ regras[benigno] $\cap$ regras[malicioso]
5 **para cada** *classe* $C \in$ *{benigno, malicioso}* **faça**
6 regras[C] $\leftarrow$ regras[C] $\setminus$ regrasInterseccao
7 regras[C] $\leftarrow X \in$ regras[C] $\wedge\ X \not\supset Y, \forall\ Y \in$ regras[C]
8 **retorna** *regras*

A partir da intersecção entre o conjunto de regras geradas (linha 4), removemos, dos conjuntos de regras de cada classe, as regras presentes nesta intersecção (linha 6). O intuito é remover as regras que se encontram tanto nos conjuntos de dados de amostras maliciosas quanto nas benignas.

Nesse ponto, utilizando os teoremas do algoritmo Apriori (Seção 2.4.1), inferimos que "Se um conjunto de características pode classificar uma amostra, logo seus subconjuntos também levarão à mesma classificação". Ou seja, consideremos a seguinte regra de associação:

$$\{C_1, C_2, C_3, C_4, C_5\} \implies \textit{Malicioso}$$

Logo, qualquer subconjunto dessas características implicará na mesma classificação. Por exemplo:

$$\begin{matrix} \{C_1, C_2, C_4, C_5\} \\ \{C_2, C_5\} \\ \{C_1, C_3, C_4\} \end{matrix} \implies \textit{Malicioso}$$

Portanto, faz-se necessário remover, do conjunto de regras de cada classe, aquelas regras que são superconjuntos de alguma outra regra existente neste conjunto, podando as regras que possuam um número maior de itens (características) e que levariam a mesma classificação (linha 7).

4.1.2 Qualificação das Regras de Associação

Para melhorar a eficacia e eficiência das regras de associação, o método proposto faz uso das métricas de qualidade de regras de apresentadas na Seção 2.5.

O Algoritmo 4 resume o processo de qualificação das regras de associação do EQAR. Após a obtenção das regras, é encontrada a cobertura e calculado o valor da métrica de qualificação selecionada (linhas 6 e 7, respectivamente) para cada regra, dispondo-as em ordem decrescente, baseando-se no valor encontrado (linha 9). Em seguida, as melhores regras são selecionadas de acordo com a porcentagem definida (no mínimo 1, caso haja regras geradas) (linhas 10 e 11) e encaminhadas para a avaliação dos aplicativos.

Algoritmo 4: Qualificação de Regras de Associação

Entrada:
D: conjunto de dados
regras: conjunto de regras (Algoritmo 3)
metricaQualidade: métrica de qualidade a ser utilizada
threshold: porcentagem de regras a serem selecionadas

1 **para cada** *classe* $C \in$ *{benigno, malicioso}* **faça**
2 P $\leftarrow$ `tamanhoDe`$(t \in D \wedge t[classe] = C)$
3 N $\leftarrow$ `tamanhoDe`$(t \in D \wedge t[classe] \neq C)$
4 listaRegraQualidade $\leftarrow \varnothing$; `// lista com os pares (regra, qualidade)`
5 **para cada** *regra* $R \in$ *regras[C]* **faça**
6 p, n $\leftarrow$ `encontrarCobertura`(R, C)
7 qualidade $\leftarrow$ `calcularQualidade`$(metricaQualidade, p, n, P, N)$
8 listaRegraQualidade $\leftarrow$ listaRegraQualidade $\cup$ (R, qualidade)
9 regrasOrdenadas $\leftarrow$ `ordenar`$(listaRegraQualidade)$
10 k $\leftarrow$ `tamanhoDe`$(regrasOrdenadas) \times threshold + 1$
11 regrasQualificadas[C] $\leftarrow$ regrasOrdenadas[1:k]
12 **retorna** *regrasQualificadas*

4.1.3 Avaliação das Amostras

No processo de avaliação das amostras com o uso de regras de associação qualificadas (Algoritmo 5), cada amostra é avaliada individualmente e o número de regras que se enquadram como subconjunto das características presentes na amostra é contabilizado para cada classe (linhas 7 a 9).

Tendo como base a distribuição das regras entre as classes, após a contabilização

Algoritmo 5: Avaliação dos Aplicativos

Entrada:
D: conjunto de dados de teste
regrasQualificadas: conjunto de regras qualificadas (Algoritmo 4)

```
1  para cada classe C ∈ {benigno, malicioso} faça
2      regrasCont[C] ← 0
3  predProb ← ∅;                  // lista com as predições probabilísticas
4  predBin ← ∅;                   // lista com as predições binárias
5  para cada transação t ∈ D faça
6      para cada classe C ∈ {benigno, malicioso} faça
7          para cada regra R ∈ regrasQualificadas[C] faça
8              se R ⊂ t então
9                  regrasCont[C]++
10         regrasCont[C] = normalizar(regrasCont[C])
11     para cada classe C ∈ {benigno, malicioso} faça
12         porcentagem[C] ← regrasCont[C] / soma(regrasCont)
13     predProb ← predProb ∪ porcentagem
14     se porcentagem[malicioso] ≥ porcentagem[benigno] então
15         pred ← malicioso;                          // amostra maliciosa
16     senão
17         pred ← benigno;                            // amostra benigna
18     predBin ← predBin ∪ pred
19 retorna predProb, predBin
```

da cobertura, é feita a normalização dos contadores (linha 10) e então calculado esta cobertura em valores percentuais (linha 12), gerando a predição probabilística, selecionando a classe com maior valor como resultado para a predição binária. Em caso de valores iguais a amostra será considerada maliciosa.

4.2 Comparação com Outros Métodos

Como forma de avaliar o EQAR em relação a outros métodos de classificação, que empregam mineração de regras de associação, decidimos compará-lo com o CBA, o CMAR e o CPAR. Esses métodos apresentam particularidades em relação ao EQAR e

possuem implementações disponíveis (motivo de sua escolha).

O método CBA (Liu et al., 1998) emprega o algoritmo Apriori modificado para gerar as regras de associação, podando as regras usando um método baseado em taxa de erro pessimista (Quinlan, 2014). Se a taxa de erro pessimista da regra R for maior que a taxa de erro pessimista da regra R' (obtida pela exclusão de um dos itens de R), então a regra R é podada. Quando o antecedente de uma regra está presente em mais de uma classe, enquanto o EQAR remove esta regra da avaliação de todas as classes o CBA a mantém na classe na qual a regras possui maior valor de confiança.

O CMAR (Li et al., 2001) emprega múltiplas regras de associação para predição. Utilizando como base o FP-Growth para minerar grandes conjuntos de dados, o CMAR adiciona uma nova estrutura de dados, denominada *CR-Tree*, para melhorar a precisão e a eficiência. A função principal dessa estrutura é armazenar e recuperar um grande número de regras de forma compacta e eficiente.

O método de classificação CPAR (Yin e Han, 2003) incorpora um algoritmo guloso para gerar regras diretamente do conjunto de dados de treino, evitando a geração de regras grandes candidatas a partir de conjuntos de itens frequentes, como ocorre nos outros métodos de classificação. Adicionalmente, o CPAR gera e testa uma quantidade maior de regras do que classificadores tradicionais, evitando assim a perda de regras importantes. O CPAR utiliza a acurácia de Laplace para avaliar cada regra e selecionar as melhores regras para a predição.

A Tabela 4.1 apresenta um resumo das principais características dos quatro métodos (CBA, CMAR, CPAR e EQAR).

Como pode ser observado, no critério de classificação de regras, CBA e CMAR utilizam essencialmente confiança e suporte. O CBA altera o Apriori para calcular tanto o suporte da regra quanto o suporte apenas do antecedente da regra. Já o CMAR usa como base o FP-Growth, modificando-o para encontrar os itens frequentes e gerar as regras em apenas um passo. Em ambos, quando duas regras têm suporte e confiança idênticos, a regra escolhida é a de menor cardinalidade,isto é, a quantidade de itens que compõem a regra.

Para evitar o *overfitting*, o CPAR usa a acurácia de Laplace (Clark e Boswell, 1991) para avaliar cada regra. A acurácia de Laplace é dada por $(n_c + 1)/(n_t + k)$, onde k é o número de classes, n_t é o número total de amostras satisfeitas pela regra, dentre as quais n_c amostras pertencem a classe c, que representa a classe de predição da regra.

Com relação à seleção/poda, no CMAR as regras geradas além de terem alta confiança e estarem positivamente correlacionadas a classe indicada nas regras, devem

Tabela 4.1: Métodos de Classificação Baseados em Regras de Associação

Método	**Geração de Regras**	**Classificação das Regras**	**Seleção/Poda**	**Predição**
CBA	Apriori	Confiança, Suporte, Cardinalidade	Taxa de Erro Pessimista, Cobertura	Menor Número de Erros
CMAR	FP-Growth	Confiança, Suporte, Cardinalidade	Confiança, Correlação Positiva, Cobertura	Maior χ^2 Ponderado
CPAR	PRM	Acurácia de Laplace	K Melhores	Maior Precisão Esperada
EQAR	ECLAT	Métrica de Qualidade	Regras Redundantes, Melhores Regras	Maior Probabilidade

cobrir um limiar δ de amostras do conjunto de treino. Para o CBA, além de considerar a **taxa de erro pessimista**, a cobertura de uma amostra é suficiente. Enquanto o CPAR utiliza a seleção das regras de acordo com os valores da acurácia de Laplace mais altos, o EQAR utiliza a remoção de regras redundantes entres as classes, selecionando as melhores regras utilizando as métricas de qualidade.

Na predição, tanto o CMAR quanto o CPAR utilizam múltiplas regras. Contudo, enquanto o CMAR utiliza o maior qui-quadrado (χ^2) ponderado, o CPAR usa a maior precisão esperada ao considerar a predição. Especificamente, o CMAR seleciona múltiplas regras aplicáveis à uma amostra de teste e avalia a correlação entre as regras pertencentes a mesma classe, medida através de um qui-quadrado ponderado. No caso do CPAR, o método utiliza as melhores regras de cada classe para predição, escolhendo a classe com a maior precisão esperada como classe predita.

O CBA, adotando a ordenação das regras baseando no menor número de erros, prediz a amostra de acordo com a primeira regra que a satisfaça. Nos casos em que não há regra aplicável para a amostra, esta é associada à classe padrão, isto é, a classe dominante no conjunto de dados de treino (Thabtah, 2007).

Finalmente, o EQAR seleciona, para o classificador, as regras, tanto de maliciosos

como de benignos, e associa a amostras de teste a classe que apresenta a maior probabilidade de ocorrer, considerando ambos os conjuntos de regras. Usamos a probabilidade na previsão porque não se pode esperar que uma única regra seja capaz de predizer perfeitamente a classe de cada amostra que esta satisfaça, o que segundo Li et al. (2001) pode levar a uma predição tendenciosa ou super ajustada. Devemos também considerar que amostras benignas podem apresentar padrões de comportamentos que sugerem atividade maliciosa.

Além disso, usamos o limiar das melhores regras, baseando-se nas métricas de qualidade em vez de todas as regras, porque há um número diferente de regras para classes diferentes e não queremos usar regras de classificação com baixa qualidade quando já existem regras suficientes para fazer a predição.

Os resultados da experimentação e a comparação entres os métodos será mostrado no Capitulo 6, Seção 6.2.

Capítulo 5

Metodologia da Experimentação

Este capítulo apresenta a metodologia aplicada para testar e validar o método proposto, bem como compará-lo aos outros métodos e classificadores. Para tanto, descrevemos os conjuntos de dados utilizados e apresentamos uma análise de sua estruturação (Seção 5.1). Também definimos toda a configuração para experimentação (Seção 5.2), incluindo as métricas empregadas na avaliação (Seção 5.2.1), os parâmetros empregados na configuração dos algoritmos (Seção 5.2.2) e as técnicas aplicadas na validação do método (Seção 5.2.3). Por fim, o ambiente de execução (Seção 5.3) é apresentado.

5.1 Conjuntos de Dados (*Datasets*)

Para avaliar o método proposto e compará-lo com outros métodos de classificação, esta pesquisa utilizou *datasets* empregados para treino de modelos de detecção de *malwares* Android em outras pesquisas acadêmicas e disponibilizados publicamente. São eles:

- **AndroCrawl**[1] (Sisto, 2013), que inclui informações sobre aplicativos encontrados em 8 lojas alternativas, com intuito de caracterizar e avaliar sua segurança.
- **ADROIT**[2] (Martín et al., 2016), que utiliza metadados de aplicativos disponíveis no Aptoide[3], uma loja alternativa de aplicativos, adotando processamento de linguagem natural (NLP) para eliminar as informações irrelevantes.
- **DefenseDroid**[4] (Colaco et al., 2021), utilizado para detectar código malicioso e

[1] https://github.com/phretor/ransom.mobi/blob/gh-pages/f/filter.7z
[2] https://www.kaggle.com/datasets/saurabhshahane/android-malware-dataset
[3] https://en.aptoide.com/
[4] https://github.com/DefenseDroid/DefenseDroid

suas variantes em tempo de execução e estender dinamicamente as informações de características dos *malwares*.

- **DREBIN-215**[5], que é um subconjunto do *dataset* DREBIN (Arp et al., 2014). O DREBIN é um conjunto de dados e uma ferramenta de detecção de *malwares* Android. O conjunto de dados contém caracteristicas de mais de 120 mil aplicativos, enquanto a ferramenta DREBIN usa aprendizado de máquina para detectar as amostras maliciosas.

- **KronoDroid**[6] (Guerra-Manzanares et al., 2021), que possui, além de diversos metadados, características estáticas e dinâmicas de aplicativos Android, extraídas de emulador e de dispositivo real em dois conjuntos de dados distintos.

Vale destacar que para o KronoDroid, escolhemos como características as **permissões** e as **chamadas de sistema**, uma vez que são as características comumente empregadas os trabalhos que utilizam características estáticas e dinâmicas para detecção de *malwares* Android (Wang et al., 2019).

A Tabela 5.1 apresenta a composição dos *datasets* utilizados nas avaliações.

Tendo em vista os desafios de predição que os diversos métodos encontram com as mudanças de características nos *datasets*, foi realizada uma análise exploratória em cada conjunto de dados, buscando entender o que pode afetar a predição em todos os métodos (Tabela 5.2).

Cada um dos valores apresentados na Tabela 5.2 pode fornecer informações valiosas sobre as características dos diferentes conjuntos de dados. Primeiramente, o número de **amostras duplicadas** é importante para avaliar a qualidade dos dados, fornecendo informações sobre a natureza dos dados e a probabilidade de amostras repetidas ou semelhantes. A porcentagem de amostras duplicadas nos *datasets* é alta na maioria dos conjuntos de dados, variando de 27,33% a 85,15%. Com exceção do *DefenseDroid*, todos os outros *datasets* apresentam mais da metade de suas amostras duplicadas. Isso sugere que podem haver problemas com a coleta ou pré-processamento de dados nesses conjuntos de dados, afetando a precisão de quaisquer modelos treinados neles. Vale destacar que, teoricamente, uma porcentagem maior de amostras duplicadas favorece a composição de regras de associação, mas em uma avaliação usando aprendizagem de máquina gerará um modelo enviesado.

A **esparsidade** mostra o quão dispersas as características estão nos dados e se há características que não são relevantes, o que pode gerar regras de associação pouco

[5] https://doi.org/10.6084/m9.figshare.5854653.v1
[6] https://github.com/aleguma/kronodroid

Tabela 5.1: *Datasets*

Dataset	**Características**		**Amostras**		
	Qtde.	**Tipos**	**Maliciosas**	**Benignas**	**Total**
AndroCrawl	81	Chamadas de API (24) Intenções (8) Permissões (49)	10170	86562	96732
ADROIT	166	Permissões	3418	8058	11476
DefenseDroid	2938	Permissões (1490) Intenções (1448)	6000	5975	11975
DREBIN-215	215	Chamadas de API (73) Permissões (113) Comandos do Sistema (6) Intenções (23)	5560	9476	15036
KronoDroid Disp. Real	246	Permissões (146) Chamadas de Sistema (100)	41382	36755	78137
KronoDroid Emulador	268	Permissões (145) Chamadas de Sistema (123)	28745	35246	63991

Tabela 5.2: Análise dos *Datasets*

Propriedade	AndroCrawl	ADROIT	DefenseDroid	DREBIN-215	KronoDroid Disp. Real	KronoDroid Emulador
Amostras Duplicadas (%)	75,46	85,15	27,33	51,72	56,08	56,81
Esparsidade (%)	94,27	95,31	99,23	84,48	85,31	83,80
Frequência Máxima (%)	51,05	99,07	97,85	87,20	99,09	99,65
Características no IFME (%)	87,72	88,55	98,47	63,26	70,73	71,64
IG Máximo	0,16502	0,26490	0,22690	0,18430	0,24281	0,21353
Características com IG Zero (%)	33,33	43,98	44,18	18,60	23,58	27,99
RFI Máximo	0,2496	0,12991	0,06317	0,06263	0,10498	0,11107
Características com RFI Zero (%)	0,00	21,69	20,22	1,40	0,41	1,13
PI Máximo	0,48120	0,02489	0,03050	0,02003	0,04181	0,07104
Características com PI Zero (%)	20,99	58,43	90,30	59,07	43,50	46,64
Pares com Alta Concordância (%) $\lvert\tau\rvert > 0.8$	0,19	4,93	0,13	0,13	0,23	0,33
Pares com Alta Discordância (%) $\lvert\tau\rvert < 0.2$	97,41	78,93	99,25	89,84	92,48	95,26

significativas para a classificação. Nos *datasets* analisados, a esparsidade variou de 83,80% a 99,23%. De forma geral, quanto mais esparso (perto de 100%) for um conjunto de dados mais difícil pode ser treinar modelos precisos, pois pode não haver informações suficientes disponíveis para fazer predições confiáveis. Para a mineração de regras de associação, uma esparsidade maior pode gerar uma quantidade menor de regras e/ou regras pouco representativas.

A **frequência máxima** indica a maior proporção de amostras cobertas por alguma das características e pode ajudar a identificar as características mais comuns nos diferentes conjuntos de dados. Por exemplo, para o ADROIT pelo menos uma característica está presente em 99,07% das amostras, enquanto no AndroCrawl a cobertura máxima das características se limita a 51,05% das amostras.

Para determinar melhor a cobertura das características, particionamos a frequências destas em intervalos de 10%. Enquanto o IFME (Intervalo de Frequência Mais Evidente) representa o intervalo com o maior quantitativo de características, as **características no IFME** representam a proporção de características que estão neste intervalo. Tal propriedade pode interferir nos parâmetros de avaliação, como suporte mínimo.

Para todos os conjuntos de dados avaliados, o IFME ficou no intervalo de 0 a 10%, o que significa dizer que enquanto no DefenseDroid 98,47% das características possuem frequência de até 10% das amostras, para o DREBIN-215 esse percentual é de 63,26%.

O **ganho de informação (IG)**, a **importância baseada em árvores aleatórias (RFI)** e **permutação (PI)** podem ajudar a identificar as características mais importantes para diferentes modelos, assim como a porcentagem de características com IG, RFI ou PI iguais a zero podem ajudar a identificar aquelas que são irrelevantes.

O ganho de informação (IG) máximo varia amplamente entre os conjuntos de dados, com valores entre 0,16502 (AndroCrawl) a 0,26490 (ADROIT). Da mesma forma, a importância das características baseadas em árvores aleatórias (RFI) e em permutação (PI) também variam amplamente. A porcentagem de características com valores de IG, RFI ou PI iguais a zero é relativamente alta na maioria dos *datasets*, variando de 18,60% (DREBIN-215) a 44,18% (DefenseDroid) para o IG, de 0% (AndroCrawl) a 21,69% (ADROIT) para o RFI, e de 20,99% (AndroCrawl) a 90,30% (DefenseDroid) para o PI. Isso sugere que muitas das características utilizadas por esses conjuntos de dados podem não agregar valor informativo suficiente para predizer a classe da amostra.

Essas diferenças demonstram que diferentes características podem ser mais ou menos importantes em diferentes *datasets*, e que a seleção ou engenharia de características podem ser eficientes para melhorar a precisão de modelos treinados por esses conjuntos de dados.

Por fim, a porcentagem de **pares de características com alta concordância** e a porcentagem de **pares de características com baixa concordância** podem fornecer informações sobre a relação entre diferentes características no *dataset*, o que é útil para determinar quais características podem ser redundantes ou interdependentes.

Tal concordância é medida pela correlação de Kendall. Se compararmos dois valores dessas características (indexadas por i e j), então qualquer par de observações (x_i,y_i) e (x_j,y_j) são consideradas concordantes se as classificações de ambos os elementos concordarem: ou seja, se ambas ($x_i > x_j$ e $y_i > y_j$) ou se ambos ($x_i < x_j$ e $y_i < y_j$). Eles são ditos discordantes, se ($x_i > x_j$ e $y_i < y_j$) ou se ($x_i < x_j$ e $y_i > y_j$). Produz um valor $-1 \leq \tau \leq 1$. Especificamente, quanto maior o valor absoluto de τ, mais forte a associação entre as duas características. Valores positivos sugerem que as características estão diretamente associadas, enquanto valores negativos sugerem que as características estão inversamente associadas (Puth et al., 2015).

A porcentagem de pares de características com alta concordância é geralmente baixa, variando de 0,13% (DefenseDroid) a 4,93% (ADROIT), sugerindo que os métodos baseados em regras de associação treinados com esses conjuntos de dados podem não fazer predições altamente confiáveis.

No geral, essas informações sugerem que os diferentes *datasets* têm características diferentes e podem exigir diferentes abordagens para análise e modelagem.

5.2 Configuração da Experimentação

5.2.1 Métricas de Avaliação

Tradicionalmente, métodos de classificação são avaliados com base em: **Verdadeiros Positivos (TP)**, amostras positivas reais que são corretamente previstas como positivas; **Falsos Positivos (FP)**, amostras negativas reais que são incorretamente previstas como positivas; **Verdadeiros Negativos (TN)**, amostras negativas reais que são corretamente previstas como negativas; e **Falsos Negativos (FN)**, amostras positivas reais que são incorretamente previstas como negativas.

Os valores de TP, FP, TN e FN são usados em uma matriz de confusão, ferramenta para analisar o quão bem um método de classificação pode reconhecer as amostras de diferentes classes. Os valores TP e TN nos indicam quando um método está classificando corretamente, enquanto os valores FP e FN quando está classificando erroneamente.

É com base nos valores TP, FP, TN e FN que as métricas de avaliação podem ser calculados, a partir da matriz de confusão, da seguinte forma:

- **Acurácia (ACC)**: proporção de amostras que são corretamente classificadas. Quanto maior o valor de ACC, melhor será o efeito da classificação.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

- **Precisão (PREC)**: proporção das amostras classificadas como *malwares* e que de fato são maliciosas.

$$PREC = \frac{TP}{TP + FP} \tag{5.2}$$

- **Recall (REC)**: proporção de amostras maliciosas que são corretamente classificadas. Quanto mais alto for o valor de *recall*, melhor será o efeito da classificação.

$$REC = \frac{TP}{TP + FN} \tag{5.3}$$

- **F1 Score (F1)**: representa a média harmônica de *recall* e precisão.

$$F1 = 2 \times \frac{PREC \times REC}{PREC + REC} \tag{5.4}$$

Além das tradicionais métricas de Acurácia, Precisão, Recall, utilizamos também o **Coeficiente de Correlação de Matthews** (MCC) (Chicco e Jurman, 2020), particularmente útil quando as duas classes estão desequilibradas, ou seja, uma classe aparece muito mais que a outra. O Coeficiente de Correlação de Matthews (MCC) mede a qualidade de algoritmos de classificação binária. Seu valor está entre -1 e +1, onde -1 significa predição inversa e +1 significa predição perfeita (Chicco e Jurman, 2020), mas a representaremos em valores percentuais, indicando a predição inversa quando necessário.

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \tag{5.5}$$

É importante destacar que, no contexto de detecção de *malwares* em Android, o *recall* é uma das métricas mais relevantes, pois ela representa a porcentagem de aplicativos reconhecidamente maliciosos e detectados como tal.

No entanto, nesta dissertação utilizamos o MCC como métrica de comparação. O MCC fornece uma medida balanceada que pode ser usada mesmo se as classes tiverem tamanhos diferentes, combinando precisão e a cobertura da predição de maneira equilibrada, produzindo uma pontuação alta somente se o classificador obtiver resultados

ótimos em todas as quatro células da matriz de confusão (Cao et al., 2020; Chicco e Jurman, 2020; Chicco et al., 2021; Jurman et al., 2012).

5.2.2 Parâmetros Fixos

Para a execução dos métodos, parâmetros como tamanho máximo da regra e confiança foram fixados.

O **tamanho máximo da regra** representa a quantidade de características que integrarão as regras de associação geradas. O Algoritmo 3, na linha 7, faz uma poda nas regras de associação geradas através da exclusão dos superconjuntos. Com isso, nos experimentos iniciais percebeu-se que regras de associação compostas com mais de 4 (quatro) características, pouco acrescem ao método EQAR em comparação a complexidade computacional exigida para seu tratamento.

Utilizamos o valor de **confiança mínima** em 95%, uma valor adequado (Sun et al., 2016) para selecionarmos as características que ocorrem simultaneamente na maioria das amostras dos conjuntos de dados, a fim de caracterizar de maneira mais refinada o comportamento das amostras.

Com relação aos algoritmos de aprendizagem de máquina RF e SVM, utilizamos a execução padrão apresentada na biblioteca *scikit-learn*, sem qualquer alteração de parâmetro ou hiperparâmetro. O RF é executado com 100 árvores na floresta e considera a raiz quadrada da quantidade de características ao procurar a melhor divisão entre as classes. Já o SVM utiliza o kernel *Radial Basis Function* (RBF) e heurística de redução para diminuir o tempo de treino.

Nos demais métodos, mantivemos os valores padrões, recomendados nas respectivas documentações.

5.2.3 Validação do Método

Para validação do método, utilizamos da técnica de validação cruzada, que consiste em dividir o conjunto de dados em K partições (*folds*). A função de predição é aprendida usando $K-1$ partições, e a partição deixada de fora é usada para teste. Como o número de amostras nos conjuntos de dados geralmente é desbalanceado, optamos por utilizar sua variação estratificada, pois mantém a proporção original de cada classe na etapa de teste. Geralmente, realiza-se a validação cruzada usando $K = 5$ ou $K = 10$, uma vez que estes valores produzem estimativas de taxa de erro de teste que não sofrem de viés ou variância excessivamente altos (James et al., 2013). Por padrão, a biblioteca scikit-learn (Pedregosa et al., 2011) utiliza $K = 5$, e por isso decidimos manter este

valor.

Qualquer conjunto de dados que apresente uma distribuição desigual entre suas classes pode ser considerado desbalanceado (He e Garcia, 2009). No entanto, a percepção comum é que conjuntos de dados desbalanceados exibem desequilíbrios significativos e, em alguns casos, extremos, não sendo incomuns os desequilíbrios de 100:1, 1.000:1 e 10.000:1.

5.3 Ambiente de Avaliação

No ambiente de avaliação, utilizamos um computador com processador Intel Xeon E5-4617 Octa-core de 2.90GHz, com 32GB RAM e 150GB de HD para a execução dos experimentos. O sistema operacional utilizado foi o Linux Ubuntu 20.04.3 LTS. Para a implementação e automação da execução dos métodos, utilizamos a linguagem Python (versão 3.8.10) e as bibliotecas NumPy (versão 1.22.1), Pandas (versão 1.3.5) e scikit-learn (versão 1.0.2).

A biblioteca **scikit-learn** (Pedregosa et al., 2011) integra uma ampla gama de algoritmos de aprendizagem de máquina de última geração para problemas supervisionados e não supervisionados de média escala. A ênfase é colocada na facilidade de uso, desempenho, documentação e consistência da API, incentivando seu uso em ambientes acadêmicos e comerciais. A biblioteca **pandas** (McKinney et al., 2010) visa facilitar o trabalho com conjuntos de dados e fornecer uma gama de funções para análise e manipulação de dados e implementação de modelos estatísticos.

Os métodos CMAR e o CPAR, implementados em linguagem R, foram integrados na ferramenta através da biblioteca rpy2[7] (versão 3.5.2). A biblioteca **pyFIM** (Borgelt, 2012), utilizada no EQAR, disponibiliza várias implementações de mineração de conjunto de itens frequentes e regras de associação.

A Tabela 5.3 apresenta também as linguagens e bibliotecas de implementação específica de cada método de classificação utilizado para avaliação.

[7] https://pypi.org/project/rpy2/
[8] https://github.com/jirifilip/pyARC
[9] https://github.com/ianjjohnson/arulesCBA
[10] https://borgelt.net/pyfim.html

Tabela 5.3: Implementação dos Métodos

Método	Linguagem	Bibliotecas/Pacotes
CBA	Python (versão 3.8.10)	pyARC[8] (versão 1.1.4)
CMAR	R (versão 4.2.1)	arulesCBA[9] (versão 1.2.4)
CPAR	R (versão 4.2.1)	arulesCBA (versão 1.2.4)
EQAR	Python (versão 3.8.10)	pyFIM[10] (versão 6.28)
RF	Python (versão 3.8.10)	scikit-learn (versão 1.0.2)
SVM	Python (versão 3.8.10)	scikit-learn (versão 1.0.2)

Capítulo 6

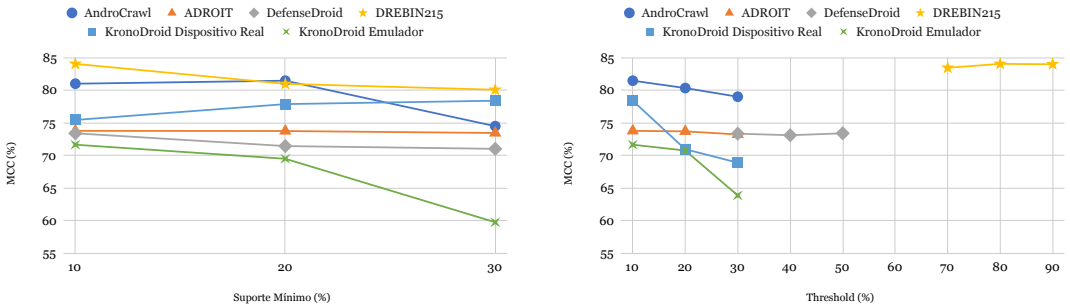
Resultados

Neste capítulo, primeiramente apresentamos os resultados do ajuste do suporte e do *threshold* do EQAR (Seção 6.1), necessários à etapa de classificação. Na sequência (Seção 6.2), apresentamos o desempenho dos métodos de classificação baseados em regras de associação (CBA, CPAR, CMAR e EQAR) e comparamos os resultados com o *Random Forest* (RF) e o *Support Vector Machine* (SVM), que são modelos de aprendizagem de máquina mais utilizados para detecção e *malwares* Andoid (Kouliaridis e Kambourakis, 2021; Liu et al., 2020). Por fim (Seção 6.3), utilizando-se de métodos de seleção de características, comparamos os resultados obtidos pelo EQAR com os *datasets* originais e reduzidos.

6.1 Suporte Mínimo e *Threshold*

O valor de suporte desempenha um papel importante na predição geral de classificadores derivados de técnicas de classificação associativa. Se utilizarmos um valor de suporte muito alto, corremos o risco de ignorar regras de boa qualidade. Já com um suporte muito baixo, potencializamos o problema de *overfitting*, levando a regras redundantes que poderão consumir mais tempo de processamento (Thabtah et al., 2005). Portanto, o primeiro passo é definirmos os valores de **suporte mínimo** e ***threshold***, utilizados pelo EQAR, mais adequados para o processo de classificação.

Logo, avaliamos o suporte mínimo com os valores de 10%, 20% e 30%, um intervalo compatível com a literatura, e o *threshold* com os valores de 10% a 90%, em intervalos de 10%. No gráfico da Figura 6.1 apresentamos a evolução dos valores do MCC para o suporte mínimo (Figura 6.1a) e *threshold* (Figura 6.1b), e na Tabela 6.1 o sumário dos parâmetros utilizados para a avaliação de cada *dataset*.



(a) Gráfico de Suporte Mínimo (b) Gráfico de *Threshold*

Figura 6.1: Avaliação de Suporte Mínimo e *Threshold*

Tabela 6.1: Parâmetros Utilizados

Dataset	Suporte Mínimo (%)	*Threshold* (%)	Métrica de Qualidade
AndroCrawl	20	10	Prec
ADROIT	10	10	C1
DefenseDroid	10	50	C2
DREBIN-215	10	80	Prec
KronoDroid Disp. Real	30	10	C2
KronoDroid Emulador	10	10	C1

Como podemos observar, quatro dos seis *datasets* em avaliação apresentaram melhores resultados com suporte e/ou *threshold* em 10%. Entretanto, um *threshold* de 50% e 80% levam a melhores resultados para os *datasets* DefenseDroid e DREBIN-215, respectivamente. O principal motivo dessa divergência entre os *datasets* é o fato de o DREBIN-215 ser menos esparso e o DefesenseDroid conter uma quantidade de características muito superior aos demais *datasets*, ambos demandando um *threshold* maior para melhor generalizar os dados.

6.2 Desempenho dos Métodos de Classificação

Esta seção apresenta os resultados obtidos pelos métodos de classificação e aprendizagem de máquina aplicados. Antes, é importante observar que os resultados apresentados são específicos para o *dataset* avaliado e podem não necessariamente ser generalizados para os outros conjuntos de dados.

Além disso, a escolha das métricas de avaliação pode ter impacto na interpretação dos resultados, sendo importante considerar cuidadosamente as métricas relevantes.

Por fim, uma vez que, com exceção do ADROIT, o RF e SVM apresentaram resultados superiores, nossas observações se limitaram aos modelos de classificação baseados em regras.

6.2.1 AndroCrawl

O resultado das avaliações dos modelos para o *dataset* AndroCrawl é apresentado na Figura 6.2. Vale destacar que o CMAR não apresentou resultados para este *dataset*. A implementação utilizada apresentou um erro de *overflow*, que acreditamos ter sido causada pela FP-Tree ao processar as amostras do conjunto de dados, que é próxima de 100 mil.

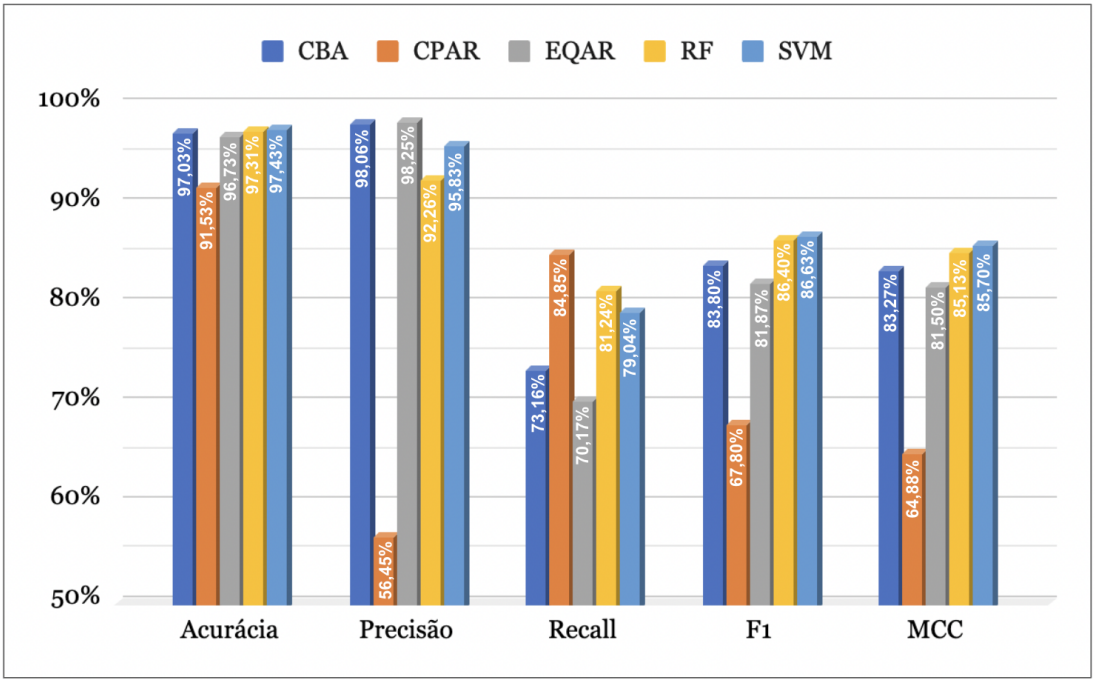


Figura 6.2: Resultados AndroCrawl

Nenhum dos modelos de regras ou aprendizagem mostrou-se dominante nas cinco métricas avaliadas. Dos três modelos baseados em regras de associação, o CBA apresentou altos valores de acurácia, precisão, F1 e MCC. O CPAR, por outro lado, apresentou valores relativamente baixos em quase todas as métricas, com exceção ao *recall* (melhor resultado entre os cinco modelos). O EQAR também apresentou valores de acurácia, precisão (seu melhor resultado), F1 e MCC altos, mas obteve o menor valor de *recall* entre todos os modelos.

Analisando melhor os modelos baseados em regras, o CBA tem o melhor desempenho geral, com acurácia de 97,03%, precisão de 98,06%, *recall* de 73,16%, F1 de 83,80% e MCC de 83,27%. Ter os valores mais altos para precisão, *recall* e MCC, indica ser o melhor em predizer corretamente a classe das amostras.

O CPAR tem o pior desempenho geral, com acurácia de 91,53%, precisão de 56,45%, *recall* de 84,85%, F1 de 67,80% e MCC de 64,88%. Sua precisão muito baixa se justifica pela porcentagem de falsos positivos apresentada pelo método. Já o seu MCC relativamente baixo indica sua ineficácia em predizer corretamente as amostras para o *dataset* AndroCrawl.

O EQAR tem acurácia de 96,73%, precisão de 98,25%, *recall* de 70,17%, F1 de 81,87% e MCC de 81,50%, tendo o menor *recall*, mas a maior precisão entre os modelos. O baixo valor de *recall* evidencia que o EQAR está falhando em detectar algumas amostras maliciosas, resultando na classificação incorreta dessas amostras como falsos negativos. No entanto, sua melhor precisão indica que gera menos falsos positivos.

6.2.2 ADROIT

O resultado das avaliações dos modelos para o *dataset* ADROIT é apresentado na Figura 6.3.

Assim como no AndroCrawl, nenhum dos modelos mostrou-se dominante nas cinco métricas avaliadas. Dos quatro modelos baseados em regras de associação, o EQAR obteve: (i) a maior acurácia (89,21%), o que indica que ele teve a maior proporção de predições corretas; (ii) a maior precisão (95,45%), o que indica que a maioria das predições positivas feitas por esse modelo eram de fato amostras maliciosas; e (iii) o maior MCC (73,81%), indicando ser o melhor classificador para este *dataset*.

Por outro lado, o EQAR obteve um desempenho bem modesto em relação ao *recall* (66,97%), ficando atrás do CPAR (68,40%) e CBA (71,53%), mas ainda à frente do CMAR (54,10%). Por fim, a métrica de F1 foi mais equilibrada, com o EQAR obtendo um valor de 78,71%, atrás apenas do CBA (79,63%).

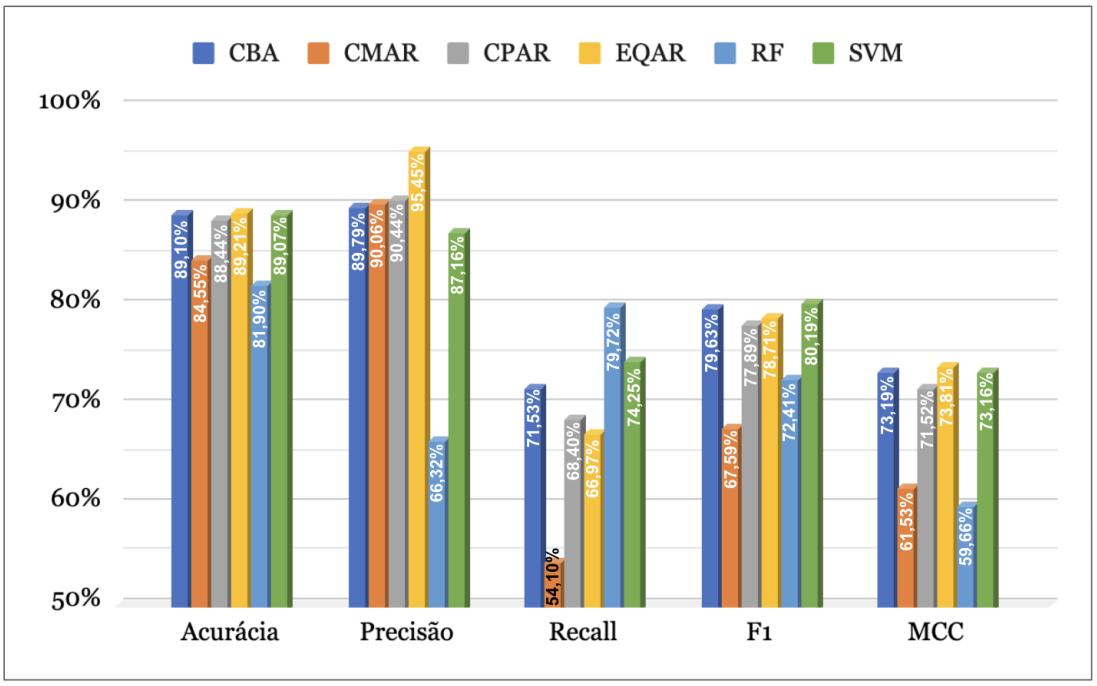


Figura 6.3: Resultados ADROIT

Dos quatro modelos, o CBA e o CPAR apresentaram resultados estáveis, com valores altos em quase todas as métricas. O CMAR foi o pior dos modelos, obtendo MCC de 61,53%, maior somente que o MCC do RF (59,66%).

Acreditamos que o desempenho do EQAR para este conjunto de dados deve-se à porcentagem de amostras duplicadas e pares de características com alta concordância, que apresentam os maiores valores dentre os *datasets* avaliados, o que pode também ter levado ao enviesamento do modelo RF.

6.2.3 DefenseDroid

O resultado das avaliações dos modelos para o *dataset* DefenseDroid é apresentado na Figura 6.4.

Sobre os modelos que usam regras de associação, o CPAR e o EQAR se mostraram estáveis em todas as métricas, tendo o CBA apresentado os maiores valores, com acurácia de 89,50%, *recall* de 92,20% e MCC de 79,12%. Contudo o CBA executou em aproximadamente **107 horas** de processamento, enquanto CMAR, CPAR e EQAR executaram, em média, em 1700, 1450 e 240 segundos, respectivamente.

O EQAR tem desempenho semelhante ao CPAR, com alta acurácia (86,71%), *recall*

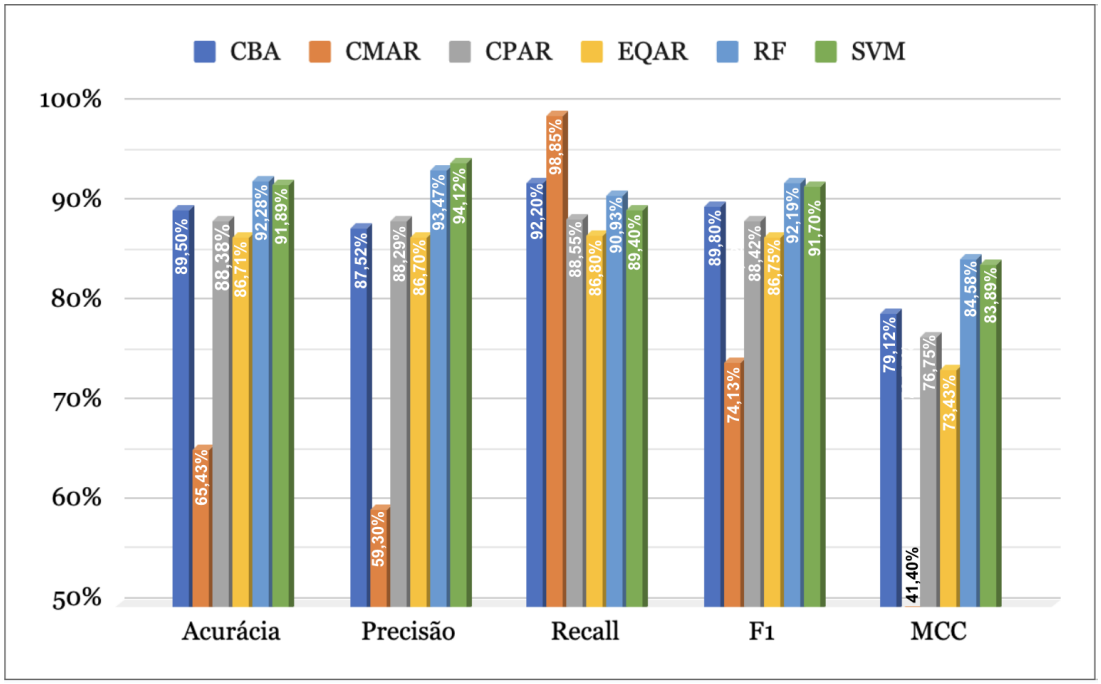


Figura 6.4: Resultados DefenseDroid

(86,80%), F1 (86,75%) e MCC (73,43%), ficando atrás do CPAR, via de regra, menos do que 2 pontos percentuais. Tal proximidade indica que ambos os modelos são bons em evitar falsos positivos e apresentam predições confiáveis. Vale destacar que na métrica F1, o CPAR é mais equilibrado do que o EQAR.

Já o CMAR provou-se inadequado ao avaliar o DefenseDroid. Excluindo-se a métrica de *recall*, onde obteve 98,85%, o que indica que ele identifica corretamente uma alta proporção de verdadeiros positivos, todas as outras métricas foram as piores entre todos os modelos. Sua acurácia, precisão, F1 e MCC são relativamente baixas em comparação aos outros modelos. Isso sugere que o CMAR pode ser mais conservador em suas predições, enquanto os demais modelos podem ser mais agressivos.

6.2.4 DREBIN-215

O resultado das avaliações dos modelos para o *dataset* DREBNI-215 é apresentado na Figura 6.5.

Em relação aos quatro modelos baseados em regras de associação, o EQAR apresentou um desempenho alto, figurando como o segundo melhor modelo de classificação. Em termos de acurácia, com 92,61%, o EQAR mostra que é capaz de classificar corre-

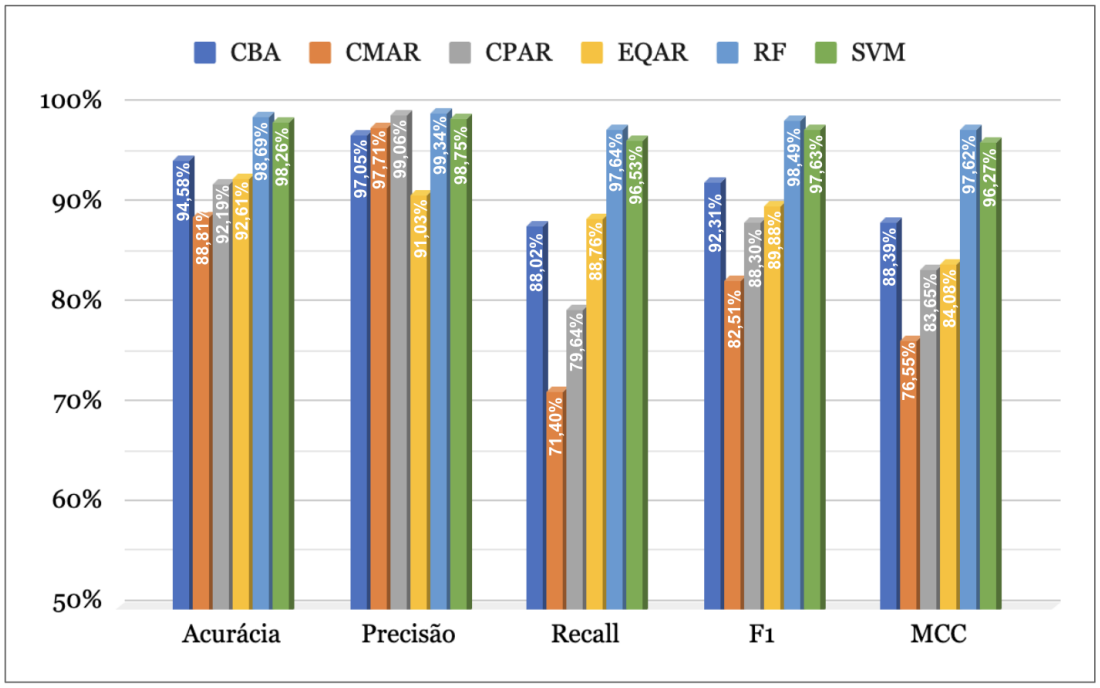


Figura 6.5: Resultados DREBIN-215

tamente a grande maioria das amostras do conjunto de teste. Nas métricas de precisão e *recall*, o EQAR apresenta precisão de 91,03% e *recall* de 88,76%, o que indica que ele é capaz de identificar corretamente a maioria das amostras positivas, mas com uma taxa mais elevada de falsos positivos em relação aos outros modelos apresentados. Por outro lado, em termos de F1 e MCC, o EQAR apresenta desempenho mais equilibrado, com valores de 89,88% e 84,08%, respectivamente.

Comparado com os outros modelos, o EQAR apresenta desempenho relativamente bom em relação à maioria deles. Em termos de *recall*, o EQAR supera o CBA, que apresenta valor de 88,02% em comparação com os 88,76% do EQAR. Contudo, o CBA apresenta o melhor equilíbrio entre as métricas, alcançando MCC de 88,39% frete a 84,08% do EQAR.

Observando as métricas de precisão e *recall*, o CPAR apresenta bom desempenho, com valores de 99,06% e 79,64%, respectivamente. Isso indica que ele é capaz de identificar a grande maioria das instâncias positivas, com baixo índice de falsos positivos, enquanto ainda mantém *recall* razoável. O CMAR tem o desempenho mais baixo, com acurácia de 88,81% e valores mais baixos de *recall* e MCC (76,55%).

Por fim, em relação às métricas de F1 e MCC, o CPAR apresenta bons resultados, com valores de 88,30% e 83,65%, respectivamente. O desempenho do CPAR nessas

métricas é semelhante ao do EQAR.

6.2.5 KronoDroid Dispositivo Real

Os resultados para o *dataset* KronoDroid Dispositivo Real são apresentados na Figura 6.6.

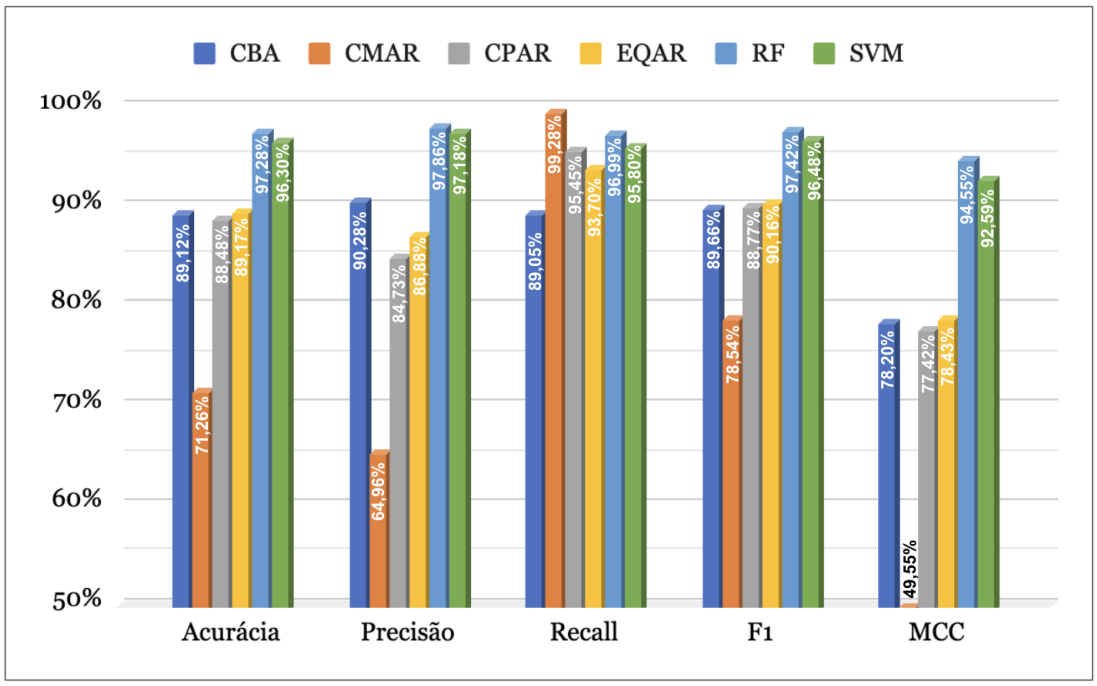


Figura 6.6: Resultados KronoDroid Dispositivo Real

Em relação aos quatro modelos baseados em regras de associação, nota-se que o CBA, CPAR e EQAR são similares. O EQAR tem acurácia de 89,17%, com precisão de 86,88% e *recall* de 93,70%, sendo seus resultados bastante semelhantes ao CBA, que possui acurácia de 89,12%, precisão de 90,28% e *recall* de 89,05%. Ambos os modelos apresentam F1 de aproximadamente 90% e MCC de aproximadamente 78%, sugerindo que podem ser adequados para classificação. O CPAR apresenta acurácia ligeiramente menor que EQAR e CBA, com precisão de 84,73% e *recall* de 95,45%. Seu F1 e MCC são 89,77% e 77,42% respectivamente, indicando que o modelo tem bom equilíbrio entre a identificação correta de amostras positivas e negativas.

No entanto, o CMAR apresenta acurácia consideravelmente menor (71,26%), com precisão de 64,96% e *recall* de 99,28%. Isso significa que o CMAR tem dificuldade em distinguir amostras negativas das positivas, levando a um grande número de falsos

positivos.

Em geral, o CBA parece ser o melhor modelo em termos de acurácia e precisão, enquanto o EQAR apresenta um desempenho semelhante e com melhor equilíbrio entre as métricas. O CPAR também apresenta desempenho razoável. Por outro lado, o CMAR parece ter dificuldades em classificar corretamente as amostras neste *dataset*, o que pode limitar sua utilidade na tarefa de classificação de *malwares*.

6.2.6 KronoDroid Emulador

Os resultados para o *dataset* KronoDroid Emulador são apresentados na Figura 6.7.

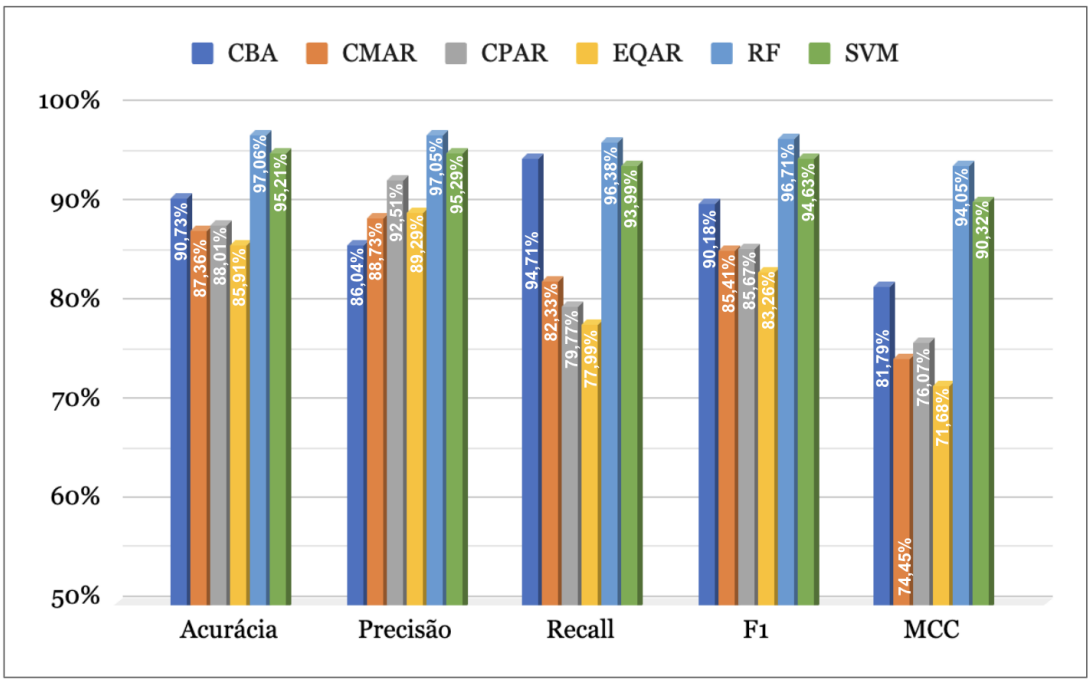


Figura 6.7: Resultados KronoDroid Emulador

No KronoDroid Emulador, o CBA apresenta-se com o melhor desempenho geral, tendo a maior acurácia (90,73%), o maior *recall* (94,74%) e o maior F1 (90,18%), dentre os quatro modelos. Esses resultados sugerem que o CBA, mesmo com a menor precisão (86,04%), foi capaz de classificar corretamente a maioria das amostras e de balancear de forma eficiente a taxa de falsos positivos e falsos negativos.

Os modelos CMAR, CPAR e EQAR apresentam desempenho inferior ao do CBA, mas ainda assim são capazes de obter resultados satisfatórios. O CMAR é o segundo melhor em termos de acurácia (87,36%) e MCC (85,41%), enquanto o CPAR apresenta

a maior precisão (92,51%) e o EQAR tem o melhor F1 (83,26%).

O F1 e o MCC do EQAR (83,26% e 71,68%, respectivamente) foram os menores entre os valores considerados, indicando que ele foi o modelo com menor correlação entre as classificações preditas e as reais. Já em relação à precisão, o EQAR obteve o segundo melhor resultado (89,29%), atrás do CPAR. No entanto, o EQAR teve o menor *recall* (77,99%). Por outro lado, a acurácia do EQAR (85,91%) não é significativamente diferente da acurácia do CMAR (87,36%) e ligeiramente inferior à acurácia do CPAR (88,01%). Isso sugere que, em termos de capacidade geral de classificação, o EQAR pode estar no mesmo nível dos demais modelos, mas apresentando uma limitação específica na identificação de amostras maliciosas, o que afeta sua capacidade de classificar corretamente as amostras.

6.3 Utilizando Seleção de Características

A qualidade das características em um *dataset* pode influenciar no treino e predição. Logo, a seleção de características é uma das partes mais importantes da construção de qualquer modelo. A seleção de características é feita para que a fase de treino seja executada de forma mais rápida, reduzindo a complexidade do modelo e tornando a interpretação dos dados mais fácil. Além disso, melhora a qualidade do modelo desde que seja escolhido um subconjunto de características correto, minimizando o problema de *overfitting* (HR, 2019).

É crucial entender a importância da seleção de características ao criar um modelo de classificação. Em problemas da vida real, é quase raro que todas as características no conjunto de dados sejam úteis para construir um modelo. Adicionar características redundantes reduz a capacidade de generalização do modelo e também pode reduzir a precisão geral de um classificador. Além disso, adicionar mais características ao conjunto de dados aumenta a complexidade geral do modelo (Gupta, 2023).

Com intuito de averiguarmos o comportamento do modelo proposto, para este trabalho usamos o PRNR e o limiar de variância como métodos de seleção de características. Este devido a simplicidade e amplo uso, aquele devido a utilização do conceito de suporte, similar ao modelo proposto.

O limiar de variação remove todas as características cuja variação não atende ao limite mínimo estabelecido, e podem não ter grande utilidade para o modelo. Por padrão, remove todas as características de variância zero. Assume que as características com uma variância mais alta podem conter informações mais úteis, sem considerar a relação entre características ou entre as características e a classe.

A classificação de permissão com taxa negativa ou PRNR (*Permission Ranking with Negative Rate*) apresentada por Sun et al. (2016), fornece uma classificação sobre as permissões dos aplicativos Android. A abordagem opera em duas matrizes, M e B, que representam a porção do *dataset* de amostras maliciosas e benignas, respectivamente. M_{ij} representa se a permissão j é solicitada pela amostra maliciosa i e B_{ij} representa se a permissão j é solicitada pela amostra benigna i.

Uma vez que o tamanho de B pode ser muito maior que o tamanho de M, o PRNR calcula o suporte de cada permissão no conjunto de dados maior e, em seguida, dimensiona proporcionalmente o suporte para corresponder ao do conjunto de dados menor, equilibrando as duas matrizes através da Equação 6.1:

$$S_B(P_j) = \frac{\sum_i B_{ij}}{Tamanho(B_j)} \times Tamanho(M_j) \tag{6.1}$$

onde P_j denota a j-ésima permissão e $S_B(P_j)$ representa o suporte da j-ésima permissão na matriz B. O PRNR pode então ser implementado usando a Equação 6.2:

$$R(P_j) = \frac{\sum_i M_{ij} - S_B(P_j)}{\sum_i M_{ij} + S_B(P_j)} \tag{6.2}$$

onde $R(P_j)$ representa a taxa de j-ésima permissão. O resultado de $R(P_j)$ tem um valor que varia entre [-1, 1]. Se $R(P_j) = 1$, isso significa que a permissão P_j é usada apenas no conjunto de dados malicioso, sendo então uma permissão de alto risco. Se $R(P_j) = -1$, isso significa que a permissão P_j é usada apenas no conjunto de dados benigno, logo uma permissão de baixo risco. Se $R(P_j) = 0$, isso significa que P_j tem muito pouco impacto na eficácia da detecção de *malwares*.

Em nossa avaliação, o limiar de variância foi fixado em 20% e aplicado separadamente na porção do *dataset* correspondente a cada classe. Já o PRNR foi aplicado sobre a totalidade de características e de amostras presentes em cada conjunto de dados e selecionadas as características com taxas ≥ 0.2 ou ≤ -0.2, removendo aquelas com taxas próximas ou igual a zero.

A Tabela 6.2 apresenta a composição do *datasets* após a aplicação dos métodos de seleção de características. Podemos notar que o limiar de variância é mais agressivo na seleção, principalmente em relação as permissões.

Nas Tabelas 6.3 e 6.4 são apresentados os resultados obtidos pelo métodos EQAR, RF e SVM nos *datasets* reduzidos. No geral, os resultados mostram uma melhora significativa no uso do EQAR com os *datasets* reduzidos, ainda abaixo dos modelos RF e SVM, contudo com valores mais próximos.

Tabela 6.2: *Datasets* Reduzidos

	Características **Limiar de Variância**		Características **PRNR**	
Dataset	**Qtde.**	**Tipos**	**Qtde.**	**Tipos**
AndroCrawl	10	Chamadas de API (8) Intenções (1) Permissões (1)	35	Chamadas de API (15) Intenções (2) Permissões (18)
ADROIT	14	Permissões	149	Permissões
DefenseDroid	24	Permissões (22) Intenções (2)	2767	Permissões (1422) Intenções (1345)
DREBIN-215	58	Chamadas de API (43) Permissões (13) Comandos do Sistema (1) Intenções (1)	170	Chamadas de API (56) Permissões (87) Comandos do Sistema (4) Intenções (23)
KronoDroid Disp. Real	35	Permissões (14) Chamadas de Sistema (21)	188	Permissões (127) Chamadas de Sistema (61)
KronoDroid Emulador	36	Permissões (11) Chamadas de Sistema (25)	187	Permissões (129) Chamadas de Sistema (58)

Tabela 6.3: Resultados com Limiar de Variância

Dataset	Método	Acurácia (%)	Precisão (%)	Recall (%)	F1 (%)	MCC (%)
AndroCrawl	EQAR	98,82	100,00	88,78	94,06	93,61
	RF	99,24	100,00	92,73	96,23	95,89
	SVM	99,24	100,00	92,72	96,22	95,88
ADROIT	EQAR	90,45	100,00	67,93	80,91	77,33
	RF	91,30	100,00	70,80	82,91	79,37
	SVM	91,30	100,00	70,80	82,91	79,37
DefenseDroid	EQAR	93,15	99,94	86,38	92,67	87,12
	RF	94,61	98,32	90,80	94,41	89,49
	SVM	94,53	99,81	89,25	94,24	89,57
DREBIN-215	EQAR	99,60	100,00	98,92	99,15	99,46
	RF	99,79	99,95	99,48	99,71	99,54
	SVM	99,59	100,00	98,90	99,45	99,13
KronoDroid Disp. Real	EQAR	98,62	100,00	97,40	98,68	97,28
	RF	99,51	99,97	99,10	99,54	99,02
	SVM	99,50	99,97	99,08	99,52	98,99
KronoDroid Emulador	EQAR	99,16	100,00	98,13	99,06	98,32
	RF	99,79	99,99	99,53	99,76	99,57
	SVM	99,77	99,98	99,50	99,74	99,53

Tabela 6.4: Resultados com PRNR

Dataset	Método	Acurácia (%)	Precisão (%)	Recall (%)	F1 (%)	MCC (%)
AndroCrawl	EQAR	99,22	100,00	92,62	96,17	95,82
	RF	99,91	100,00	99,14	99,57	99,52
	SVM	99,90	99,87	99,14	99,51	99,45
ADROIT	EQAR	95,73	100,00	85,66	92,28	89,86
	RF	97,87	100,00	92,83	96,28	94,92
	SVM	97,87	100,00	92,83	96,28	94,92
DefenseDroid	EQAR	97,04	100,00	94,08	96,95	94,24
	RF	98,41	100,00	96,82	98,38	96,86
	SVM	98,07	99,95	96,20	98,04	96,21
DREBIN-215	EQAR	97,19	100,00	92,41	96,06	94,06
	RF	99,25	100,00	97,97	98,97	98,39
	SVM	99,09	100,00	97,54	98,75	98,05
KronoDroid Disp. Real	EQAR	99,79	100,00	99,60	99,80	99,58
	RF	99,96	100,00	99,92	99,96	99,91
	SVM	99,94	100,00	99,88	99,94	99,88
KronoDroid Emulador	EQAR	99,38	100,00	98,63	99,31	98,76
	RF	99,92	100,00	99,81	99,91	99,83
	SVM	99,90	100,00	99,79	99,89	99,81

Apenas o ADROIT (77,33%) e o DefenseDroid (87,12%) reduzidos pelo limiar de variância apresentam MCC abaixo de 90%. Em compensação, todos os *datasets* reduzidos do KronoDroid (Dispositivo Real e Emulador) produziram F1 e MCC mínimo de 97%.

Curiosamente, salvo o conjunto de dados do DefenseDroid reduzido pelo limiar de variância, todos os demais *datasets* reduzidos alcançaram 100% de precisão em pelo menos um dos métodos ou seja, sem amostras classificadas como falsos positivos. Creditamos a esse fato a qualidade das características selecionadas para esses conjuntos de dados.

Com exceção do DREBIN-215, os *datasets* reduzidos pelo PRNR alcançaram resultados melhores que aqueles reduzidos pelo limiar de variância.

Pelos resultados apresentados, o método EQAR está perdendo muitas instâncias verdadeiras positivas, classificando-as como falsos negativos. Este fato fica claro quando analisamos os valores de *recall*, que é a métrica de avaliação mais distante dos valores apresentados pelo RF e SVM, prejudicando assim o MCC do EQAR.

6.4 Discussão

Em primeiro lugar, é importante observar que a avaliação de diferentes métodos de classificação baseados em regras em vários conjuntos de dados é crucial para o desenvolvimento de modelos eficazes de classificação de aplicativos maliciosos. Os resultados obtidos dessas avaliações fornecem informações sobre os pontos fortes e fracos de diferentes métodos, que podem orientar futuros esforços de pesquisa.

6.4.1 Questão de Pesquisa 1

> *Os algoritmos de mineração de regra de associação são mais eficientes do que os algoritmos de aprendizagem de máquina na detecção de malwares Android?*

De modo geral, os métodos de aprendizagem de máquina são mais eficientes que os métodos baseados em mineração de regras de associação. Contudo, ambos os tipos de métodos têm suas próprias vantagens e desvantagens, e a escolha do método mais adequado depende das características do conjunto de dados.

Os algoritmos de mineração de regras de associação são capazes de identificar padrões frequentes em grandes conjuntos de dados. Eles podem ser úteis para identificar associações entre diferentes características. No entanto, podem ter problemas com

datasets complexos ou com muitas características, como no caso do DefenseDroid. Os modelos de aprendizagem de máquina, por outro lado, se ajustam melhor aos conjuntos de dados que utilizam características mais específicas, como as chamadas de sistemas do KronoDroid.

Como apresentado na Seção 6.3, a seleção de características adequadas é capaz de fazer com que os métodos de classificação baseados em regras de associação obtenham resultados melhores, semelhantes ao obtidos por métodos de aprendizagem de máquina. Portanto, é importante considerar cuidadosamente as características do conjunto de dados antes de escolher qual método usar para a detecção de *malwares* Android. Além disso, é possível que uma combinação de diferentes algoritmos e técnicas possa produzir os melhores resultados.

6.4.2 Questão de Pesquisa 2

> *Qual melhor algoritmo de mineração de regra de associação para detecção de malwares Android?*

Com base no MCC dos resultados apresentados, podemos notar que o EQAR teve um bom desempenho no conjunto de dados ADROIT. Isso indica que o EQAR é um método promissor para detectar amostras de *malware* em conjuntos de dados semelhantes. No entanto, também é importante observar que o desempenho do EQAR foi inferior a todos os outros métodos de classificação baseados em regras apenas no KronoDroid Emulador, onde também obteve o seu menor valor de MCC.

Vale ressaltar que, embora os conjuntos de dados do KronoDroid sejam semelhantes, a porcentagem de pares com alta discordância pode ter influenciado a degradação das métricas do EQAR no KronoDroid Emulador. Isso destaca a importância de selecionar e preparar cuidadosamente conjuntos de dados para avaliar métodos de classificação baseados em regras.

Entre todos os modelos e *datasets* avaliados, o CMAR obteve o menor valor de MCC (41,40%) para o DefenseDroid e, com exceção do conjunto de dados KronoDroid Emulador, também apresentou o pior desempenho para os demais *datasets*. Esses resultados sugerem que o CMAR não é um método eficaz para a detecção de amostras maliciosas neste conjunto de dados.

Dos seis conjuntos de dados avaliados, o CBA apresentou o melhor desempenho em quatro deles. Os resultados mais próximos ao do CBA foram obtidos pelo EQAR para os *datasets* AndroCrawl e DREBIN-215, e pelo CPAR para os conjuntos de dados DefenseDroid e KronoDroid Emulador.

Como um ponto a se destacar, o *dataset* DREBIN-215 gerou mais regras para amostras benignas do que para amostras maliciosas. Isso sugere que pode haver alguma sobreposição entre as características das amostras benignas e maliciosas, o que torna o desenvolvimento de modelos eficazes de classificação baseados em regras mais desafiador.

No geral, as métricas de avaliação sugerem que o CBA teve o melhor desempenho, enquanto o CMAR teve o pior. No entanto, é importante destacar que o desempenho de um método pode variar de acordo com o *dataset* utilizado, devido às diferenças nas características e na composição, sendo necessário levar esses fatores em consideração quando da escolha do métodos mais adequado para a detecção de *malwares* Android.

Capítulo 7

Conclusão

A detecção de aplicativos maliciosos voltados para o sistema operacional Android se tornou uma das áreas mais importantes de pesquisa científica na área de segurança de dispositivos móveis, devido à proliferação dessas ameaças. Dessa forma, faz-se necessário encontrar soluções eficazes para lidar com o aumento do número de aplicativos maliciosos direcionados a dispositivos Android.

Neste trabalho, foi implementado um modelo de detecção de *malwares* baseado em mineração de regras de associação. Ao nosso conhecimento, esta é a primeira vez que esta abordagem é utilizada para a detecção de *malwares* Android.

Embora o EQAR apresente um desempenho geralmente inferior aos métodos de aprendizagem de máquina ao ser avaliado com os *datasets* originais, quando utilizado com as características mais relevantes, identificadas por métodos de seleção de características, o EQAR obtém resultados semelhantes, mas ainda inferiores, aos obtidos pelo SVM e pelo RF.

De modo geral, o EQAR obteve resultados compatíveis com os demais métodos que fazem uso de regras de associação, sendo por vezes superior a estes, apresentando equilíbrio entre as métricas.

Pelos resultados apresentados, pode-se confirmar a dependência de um conjunto de dados apropriado para o treinamento do modelo, seja do ponto de vista da quantidade ou da qualidade/importância das características. Contudo, a interpretabilidade do modelo é de fácil entendimento uma vez que as regras de associação geradas permitem a identificação clara das características que as compõem e a qual classe estão associadas.

Como contribuições do trabalho podemos destacar:

1. Uma análise empírica e compreensiva de quatro métodos de classificação baseados em regras de associação no contexto específico de detecção de *malwares* Android;

2. A proposta e implementação do método EQAR para problemas de classificação baseados em regras de associação;

3. Uma comparação dos resultados dos quatro métodos de classificação baseados em regras de associação com os modelos de aprendizagem de máquina *Random Forest* (RF) e *Support Vector Machine* (SVM) para caracterizar e quantificar, com dados numéricos, o potencial desses métodos na detecção de *malwares* Android.

Entre os grandes desafios enfrentados durante esse trabalho podemos destacar a dificuldade de encontrar *datasets* com amostras atualizadas ou que possam ser facilmente comparados em termos de características e amostras, principalmente devido a falta de metadados nesses *datasets*, a grande quantidade de características do Android e a falta padronização quanto a apresentação dos dados.

Para o futuro, pretendemos criar nosso próprio *dataset*, padronizado com o apresentação das características encontradas na documentação oficial do Android. Com ele, poderemos avaliar outras abordagens para geração de itens frequentes, buscando reduzir os falsos positivos e falsos negativos por meio da análise das amostras que não foram classificadas corretamente e da descoberta das razões por trás da classificação incorreta, além de expandir o método proposto para classificação multiclasses.

Referências Bibliográficas

Abdellatif, S., Ben Hassine, M. A., Ben Yahia, S., e Bouzeghoub, A. (2018). ARCID: A New Approach to Deal with Imbalanced Datasets Classification. Em *SOFSEM*.

Adebayo, O. S. e Abdul Aziz, N. (2019). Improved Malware Detection Model with Apriori Association Rule and Particle Swarm Optimization. *Security and Communication Networks*, 2019.

Adugna, T., Xu, W., e Fan, J. (2022). Comparison of random forest and support vector machine classifiers for regional land cover mapping using coarse resolution fy-3c images. *Remote Sensing*, 14(3):574.

Agarwal, R. C., Aggarwal, C. C., e Prasad, V. (2001). A Tree Projection Algorithm for Generation of Frequent Item Sets. *Journal of parallel and Distributed Computing*, 61(3):350–371.

Agrawal, R., Imieliński, T., e Swami, A. (1993). Mining Association Rules Between Sets of Items in Large Databases. Em *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pg. 207–216, New York, NY, USA. Association for Computing Machinery.

Agrawal, R. e Srikant, R. (1994). Fast Algorithms for Mining Association Rules. Em *Proc. 20th Int. Conf. Cery large Data Bases, VLDB*, volume 1215, pgs. 487–499. Citeseer.

Al-Fawa'reh, M., Saif, A., Jafar, M. T., e Elhassan, A. (2020). Malware Detection by Eating a Whole APK. Em *2020 15th International Conference for Internet Technology and Secured Transactions (ICITST)*, pgs. 1–7.

Ali, Y., Farooq, A., Alam, T. M., Farooq, M. S., Awan, M. J., e Baig, T. I. (2019). Detection of Schistosomiasis Factors Using Association Rule Mining. *IEEE Access*, 7:18618.

Alsoghyer, S. e Almomani, I. (2020). On the Effectiveness of Application Permissions for Android Ransomware Detection. Em *2020 6th Conference on Data Science and Machine Learning Applications (CDMA)*, pgs. 94–99.

Alzaylaee, M. K., Yerima, S. Y., e Sezer, S. (2017). Improving Dynamic Analysis of Android Apps Using Hybrid Test Input Generation.

Android Open Source Project (2023). Android Runtime (ART) and Dalvik. Disponível em https://source.android.com/docs/core/runtime. Acessado em 10 de Janeiro de 2023.

Arkin, H. e Colton, R. R. (1970). *Statistical Methods*. Barnes and Noble, New York.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., e Siemens, C. (2014). DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. Em *NDSS*, volume 14, pgs. 23–26.

Baralis, E., Cerquitelli, T., Chiusano, S., e Grand, A. (2013). P-Mine: Parallel Itemset Mining on Large Datasets. Em *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*, pgs. 266–271. IEEE.

Bishop, C. M. e Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4. Springer.

Borgelt, C. (2012). Frequent Item Set Mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):437–456.

Brin, S., Motwani, R., Ullman, J. D., e Tsur, S. (1997). Dynamic Itemset Counting and Implication Rules for Market Basket Data. *SIGMOD Rec.*, 26(2):255–264.

Bruha, I. e Kockova, S. (1993). Quality of Decision Rules: Empirical and Statistical Approaches. *Informatica*, 17:233–243.

Buczak, A. L. e Guven, E. (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176.

Cao, C., Chicco, D., e Hoffman, M. M. (2020). The MCC-F1 Curve: a Performance Evaluation Technique for Binary Classification. *arXiv preprint arXiv:2006.11278*.

Chee, C.-H., Jaafar, J., Aziz, I. A., Hasan, M. H., e Yeoh, W. (2019). Algorithms for Frequent Itemset Mining: A Literature Review. *Artificial Intelligence Review*, 52(4):2603–2621.

Chicco, D. e Jurman, G. (2020). The Advantages of the Matthews Correlation Coefficient (MCC) over F1 Score and Accuracy in Binary Classification Evaluation. *BMC genomics*, 21(1):1–13.

Chicco, D., Tötsch, N., e Jurman, G. (2021). The Matthews Correlation Coefficient (MCC) is More Reliable than Balanced Accuracy, Bookmaker Informedness, and Markedness in Two-class Confusion Matrix Evaluation. *BioData mining*, 14(1):1–22.

Christensen, D. (1999). Measuring Confirmation. *The Journal of Philosophy*, 96(9):437–461.

Clark, P. e Boswell, R. (1991). Rule Induction with CN2: Some Recent Improvements. Em *European Working Session on Learning*, pgs. 151–163. Springer.

Colaco, C. W., Bagwe, M. D., Bose, S. A., e Jain, K. (2021). DefenseDroid: A Modern Approach to Android Malware Detection. *Strad Research*, 8(5):271–282.

Ed-Daoudy, A. e Maalmi, K. (2020). Breast Cancer Classification with Reduced Feature Set Using Association Rules and Support Vector Machine. *Network Modeling Analysis in Health Informatics and Bioinformatics*, 9(1):34.

Feddaoui, I., Felhi, F., e Akaichi, J. (2016). EXTRACT: New Extraction Algorithm of Association Rules from Frequent Itemsets. Em *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pgs. 752–756. IEEE.

Galib, A. H. e Hossain, B. M. (2020). Significant API Calls in Android Malware Detection Using Feature Selection Techniques and Correlation Based Feature Elimination. Em *SEKE*, pgs. 566–571.

Gayathri, G. S. (2017). Performance Comparison of Apriori, ECLAT And FP-Growth Algorithm for Association Rule Learning. *International Journal of Computer Science and Mobile Computing*, 6(2):81–89.

GDATA (2022). G DATA Mobile Security Report: Conflict in Ukraine causes decline in malicious Android apps. Disponível em https://presse.gdata.de/news-g-

data-mobile-security-report-conflict-in-ukraine-causes-decline-in-malicious-android-apps-?id=163590. Acessado em 08 de Março de 2023.

Geng, L. e Hamilton, H. J. (2006). Interestingness Measures for Data Mining: A Survey. *ACM Computing Surveys (CSUR)*, 38(3):9–es.

Gu, X., Zhu, Y., Zhou, S., Wang, C., Qiu, M., e Wang, G. (2016). A Real-Time FPGA-Based Accelerator for ECG Analysis and Diagnosis Using Association-Rule Mining. *ACM Trans. Embed. Comput. Syst.*, 15(2).

Guerra-Manzanares, A., Bahsi, H., e Nõmm, S. (2021). KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization. *Computers & Security*, 110:102399.

Gupta, A. (2023). Feature Selection Techniques in Machine Learning (Updated 2023). Disponível em https://www.analyticsvidhya.com/blog/2020/10/feature-selection-techniques-in-machine-learning/. Acessado em 09 de Fevereiro de 2023.

Han, J., Kamber, M., e Pei, J. (2012). *Data Mining Concepts and Techniques*. Morgan Kaufmann Publishers, Waltham, Mass., 3rd edition.

Han, J., Pei, J., e Yin, Y. (2000). Mining Frequent Patterns Without Candidate Generation. *ACM Sigmod Record*, 29(2):1–12.

He, H. e Garcia, E. A. (2009). Learning from Imbalanced Data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284.

Heaton, J. (2016). Comparing Dataset Characteristics that Favor the Apriori, ECLAT or FP-Growth Frequent Itemset Mining Algorithms. Em *SoutheastCon 2016*, pgs. 1–7. IEEE.

Hoseini, M. S., Shahraki, M. N., e Neysiani, B. S. (2015). A New Algorithm for Mining Frequent Patterns in Can Tree. Em *2nd International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, pgs. 843–846. IEEE.

Houtsma, M. e Swami, A. (1995). Set-oriented Mining for Association Rules in Relational Databases. Em *11th International Conference on Data Engineering*, pgs. 25–33. IEEE.

HR, S. (2019). Static Analysis of Android Malware Detection using Deep Learning. Em *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, pgs. 841–845.

James, G., Witten, D., Hastie, T., e Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R.* Springer.

Janssen, F. e Fürnkranz, J. (2010). On the Quest for Optimal Rule Learning Heuristics. *Machine Learning*, 78(3):343–379.

Jeeva, S. C. e Rajsingh, E. B. (2016). Intelligent Phishing URL Detection using Association Rule Mining. *Human-centric Computing and Information Sciences*, 6(1):1–19.

Jiang, X., Mao, B., Guan, J., e Huang, X. (2020). Android Malware Detection Using Fine-grained Features. *Scientific Programming*, 2020.

Joyce, J. M. (1999). *The Foundations of Causal Decision Theory.* Cambridge University Press.

Jurman, G., Riccadonna, S., e Furlanello, C. (2012). A Comparison of MCC and CEN Error Measures in Multi-class Prediction. *PLoS One*, 7(8):e41882.

Kaur, M. e Kang, S. (2016). Market Basket Analysis: Identify the Changing Trends of Market Data Using Association Rule Mining. *Procedia Computer Science*, 85:78–85.

Kim, Y. S. e Yum, B.-J. (2011). Recommender System based on Click Stream Data Using Association Rule Mining. *Expert Systems with Applications*, 38(10):13320–13327.

Kouliaridis, V. e Kambourakis, G. (2021). A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection. *Information*, 12(5):185.

Lenca, P., Vaillant, B., Meyer, P., e Lallich, S. (2007). Association Rule Interestingness Measures: Experimental and Theoretical Studies. Em *Quality Measures in Data Mining*, pgs. 51–76. Springer.

Li, H. e Sheu, P. C.-Y. (2021). A Scalable Association Rule Learning Heuristic for Large Datasets. *Journal of Big Data*, 8(1):1–32.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W., e Ye, H. (2018). Significant Permission Identification for Machine-learning-based Android Malware Detection. *IEEE Transactions on Industrial Informatics*, 14(7):3216–3225.

Li, W., Han, J., e Pei, J. (2001). CMAR: Accurate and Efficient Classification based on Multiple Class-association Rules. Em *IEEE ICDM*, pgs. 369–376.

Lin, X. (2014). Mr-Apriori: Association Rules Algorithm based on MapReduce. Em *5th International Conference on Software Engineering and Service Science*, pgs. 141–144. IEEE.

Lindorfer, M., Neugschwandtner, M., e Platzer, C. (2015). MARVIN: Efficient and Comprehensive Mobile App Classification through Static and Dynamic Analysis. Em *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 2, pgs. 422–433.

Lindorfer, M., Neugschwandtner, M., Weichselbaum, L., Fratantonio, Y., Veen, V. v. d., e Platzer, C. (2014). ANDRUBIS – 1,000,000 Apps Later: A View on Current Android Malware Behaviors. Em *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*, pgs. 3–17.

Liu, B., Hsu, W., Ma, Y., et al. (1998). Integrating Classification and Association Rule Mining. Em *KDD*, volume 98, pgs. 80–86.

Liu, B., Ma, Y., Wong, C. K., e Yu, P. S. (2003). Scoring the Data Using Association Rules. *Applied Intelligence*, 18(2):119–135.

Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., e Liu, H. (2020). A Review of Android Malware Detection Approaches Based on Machine Learning. *IEEE Access*, 8:124579–124607.

Malik, S. e Khatter, K. (2016). System Call Analysis of Android Malware Families. *Indian Journal of Science and Technology*, 9.

Martín, A., Calleja, A., Menéndez, H. D., Tapiador, J., e Camacho, D. (2016). ADROIT: Android Malware Detection using Meta-information. Em *Symposium Series on Computational Intelligence (SSCI)*, pgs. 1–8. IEEE.

Martín, A., Rodríguez-Fernández, V., e Camacho, D. (2018). CANDYMAN: Classifying Android Malware Families by Modelling Dynamic Traces with Markov Chains. *Engineering Applications of Artificial Intelligence*, 74:121–133.

McKinney, W. et al. (2010). Data Structures for Statistical Computing in Python. Em *9th Python in Science Conference*, volume 445, pgs. 51–56. Austin, TX.

Mirzaei, O., Suarez-Tangil, G., Tapiador, J., e de Fuentes, J. M. (2017). Triflow: Triaging Android Applications Using Speculative Information Flows. Em *Asia Conference on Computer and Communications Security*, pgs. 640–651.

Muttoo, S. K. e Badhani, S. (2017). Android malware detection: state of the art. *International Journal of Information Technology*, 9:111–117.

Nadimi-Shahraki, M.-H. e Mansouri, M. (2017). Hp-Apriori: Horizontal Parallel-apriori Algorithm for Frequent Itemset Mining from Big Data. Em *2nd International Conference on Big Data Analysis (ICBDA)*, pgs. 286–290. IEEE.

Park, J. S., Chen, M.-S., e Yu, P. S. (1995). An Effective Hash-based Algorithm for Mining Association Rules. *Acm sigmod record*, 24(2):175–186.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., e Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Piatetsky-Shapiro, G. (1991). Discovery, Analysis and Presentation of Strong Rules. Em Piatetsky-Shapiro, G. e Frawley, W. J., editores, *Knowledge Discovery in Databases*, pgs. 229–248. AAAI Press.

Puth, M.-T., Neuhäuser, M., e Ruxton, G. D. (2015). Effective use of Spearman's and Kendall's Correlation Coefficients for Association Between Two Measured Traits. *Animal Behaviour*, 102:77–84.

Pyun, G., Yun, U., e Ryu, K. H. (2014). Efficient Frequent Pattern Mining based on Linear Prefix Tree. *Knowledge-Based Systems*, 55:125–139.

Quinlan, J. R. (2014). *C4. 5: Programs for Machine Learning*. Elsevier.

Sadgali, I., Sael, N., e Benabbou, F. (2021). Human Behavior Scoring in Credit Card Fraud Detection. *IAES International Journal of Artificial Intelligence*, 10:698–706.

Saracino, A., Sgandurra, D., Dini, G., e Martinelli, F. (2018). MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Transactions on Dependable and Secure Computing*, 15(1):83–97.

Scalas, M., Maiorca, D., Mercaldo, F., Visaggio, C. A., Martinelli, F., e Giacinto, G. (2019). On the Effectiveness of System API-related Information for Android Ransomware Detection. *Computers & Security*, 86:168–182.

Sharma, A. e Dash, S. K. (2014). Mining API Calls and Permissions for Android Malware Detection. Em *Cryptology and Network Security*, pgs. 191–205. Springer International Publishing.

Shen, F., Vecchio, J. D., Mohaisen, A., Ko, S. Y., e Ziarek, L. (2017). Android Malware Detection Using Complex-Flows. Em *37th International Conference on Distributed Computing Systems (ICDCS)*, pgs. 2430–2437. IEEE.

Sinha, G. e Ghosh, S. (2014). Identification of Best Algorithm in Association Rule Mining based on Performance. *International Journal of Computer Science and Mobile Computing*, 3(11):38–45.

Sisto, A. (2013). AndroCrawl: Studying Alternative Android Marketplaces. Master's thesis, Politecnico di Milano.

Song, M. e Rajasekaran, S. (2006). A Transaction Mapping Algorithm for Frequent Itemsets Mining. *Transactions on Knowledge and Data Engineering*, 18(4):472–481.

StatCounter (2023). Mobile Operating System Market Share Worldwide. Disponível em https://gs.statcounter.com/os-market-share/mobile/worldwide. Acessado em 07 de Março de 2023.

Statista (2023). Google Play Store: Number of Apps 2023. Disponível em https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/. Acessado em 07 de Março de 2023.

Stepanov, N., Alekseeva, D., Ometov, A., e Lohan, E. S. (2020). Applying Machine Learning to LTE Traffic Prediction: Comparison of Bagging, Random Forest, and SVM. Em *2020 12th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, pgs. 119–123.

Sun, L., Li, Z., Yan, Q., Srisa-an, W., e Pan, Y. (2016). SigPID: Significant Permission Identification for Android Malware Detection. Em *11th International Conference on Malicious and Unwanted Software (MALWARE)*, pgs. 1–8. IEEE.

Surendran, R., Thomas, T., e Emmanuel, S. (2021). On Existence of Common Malicious System Call Codes in Android Malware Families. *IEEE Transactions on Reliability*, 70(1):248–260.

Telikani, A., Tahmassebi, A., Banzhaf, W., e Gandomi, A. H. (2021). Evolutionary Machine Learning: A Survey. *ACM Computing Surveys (CSUR)*, 54(8):1–35.

Thabtah, F. (2007). A Review of Associative Classification Mining. *The Knowledge Engineering Review*, 22(1):37–65.

Thabtah, F., Cowling, P., e Peng, Y. (2005). MCAR: Multi-class Classification based on Association Rule. Em *The 3rd ACS/IEEE International Conference onComputer Systems and Applications*, pgs. 33–.

Tightiz, L., Nasab, M. A., Yang, H., e Addeh, A. (2020). An Intelligent System based on Optimized ANFIS and Association Rules for Power Transformer Fault Diagnosis. *ISA Transactions*, 103:63–74.

Vaishanav, L., Chauhan, S., Kumari, S., Sankhla, M. S., e Kumar, R. (2017). Behavioural Analysis of Android Malware and Detection. *International Journal of Computer Trends and Technology (IJCTT)*, 47(3).

Vani, K. (2015). Comparative Analysis of Association Rule Mining Algorithms Based on Performance Survey. *International Journal of Computer Science and Information Technologies*, 6(4):3980–3985.

Varsha, M. V., Vinod, P., e Dhanya, K. A. (2015). Heterogeneous Feature Space for Android Malware Detection. Em *8th International Conference on Contemporary Computing (IC3)*, pgs. 383–388.

Verbraeken, J., Wolting, M., Katzy, J., Kloppenburg, J., Verbelen, T., e Rellermeyer, J. S. (2020). A Survey on Distributed Machine Learning. *ACM Comput. Surv.*, 53(2).

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

Wang, X., Wang, W., He, Y., Liu, J., Han, Z., e Zhang, X. (2017). Characterizing Android Apps Behavior for Effective Detection of Malapps at Large Scale. *Future Generation Computer Systems*, 75:30–45.

Wróbel, Ł., Sikora, M., e Michalak, M. (2016). Rule Quality Measures Settings in Classification, Regression and Survival Rule Induction — An Empirical Approach. *Fundamenta Informaticae*, 149(4):419–449.

Xu, Z., Ren, K., e Song, F. (2019). Android Malware Family Classification and Characterization Using CFG and DFG. Em *2019 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pgs. 49–56.

Yagin, F. H., Yağin, B., ARSLAN, A., e ÇOLAK, C. (2021). Comparison of Performances of Associative Classification Methods for Cervical Cancer Prediction: Observational Study. *Turkiye Klinikleri Journal of Biostatistics*, 13:266–272.

Yerima, S. Y., Sezer, S., e Muttik, I. (2015). High Accuracy Android Malware Detection Using Ensemble Learning. *IET Information Security*, 9(6):313–320.

Yildiz, O. e 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.

Yin, X. e Han, J. (2003). CPAR: Classification based on Predictive Association Rules. Em *International Conference on Data Mining*, pgs. 331–335. SIAM.

Zaki, M. J. (2000). Scalable Algorithms for Association Mining. *Transactions on Knowledge and Data Engineering*, 12(3):372–390.

Zhang, M. e He, C. (2010). Survey on Association Rules Mining Algorithms. Em *Advancing Computing, Communication, Control and Management*, pgs. 111–118. Springer.

Zhao, Q. e Bhowmick, S. S. (2003). Association Rule Mining: A Survey. *Nanyang Technological University, Singapore*, 135.

Zou, D., Wu, Y., Yang, S., Chauhan, A., Yang, W., Zhong, J., Dou, S., e Jin, H. (2021). IntDroid: Android Malware Detection Based on API Intimacy Analysis. *ACM Trans. Softw. Eng. Methodol.*, 30(3).