UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARCELLO CAINELLI FILHO

DETECÇÃO DE MALWARES EM DISPOSITIVOS MÓVEIS UTILIZANDO MACHINE LEARNING

BAURU
2018

MARCELLO CAINELLI FILHO

DETECÇÃO DE MALWARES EM DISPOSITIVOS MÓVEIS UTILIZANDO MACHINE LEARNING

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Kelton Augusto Pontara da Costa

BAURU
2018

Marcello Cainelli Filho Detecção de Malwares em dispositivos móveis utilizando Machine Learning/ Marcello Cainelli Filho. – Bauru, 2018- 38 p. : il. (algumas color.) ; 30 cm.
Orientador: Prof. Dr. Kelton Augusto Pontara da Costa
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Ciência da Computação, 2018.
1. Dispositivos móveis. 2. Inteligência Artificial. 3. Support Vector Machines. 4. Malwares. 5. Classificador de padrões. 6. Intent-filters. 7. Matlab. 8. Android.

Marcello Cainelli Filho

Detecção de Malwares em dispositivos móveis utilizando Machine Learning

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. Kelton Augusto Pontara da Costa
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Profa. Dra. Simone das Graças Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Profa. Dra. Andrea Carla Gonçalves Vianna
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Computação

Bauru, 13 de novembro de 2018.

Agradecimentos

Agradeço à minha família por sempre me apoiar nas minhas escolhas e abrirem caminhos para eu seguir meus sonhos.

Agradeço o auxílio de Martin Styk na etapa de desenvolvimento deste projeto.

Agradeço também aos meus amigos e professores que fizeram parte desta etapa da minha vida. Em especial agradeço ao meu orientador, prof. Kelton, que me auxiliou em todas as etapas que precisei para o desenvolvimento deste trabalho.

"Remember to look up at the stars and not down at your feet."
Stephen Hawking
"It's kind of fun to do the impossible."
Walt Disney

Resumo

A evolução tecnológica em dispositivos móveis possibilitou inúmeras facilidades no mundo atual que atraíram indivíduos mal-intencionados para este cenário, tornando uma tarefa extremamente difícil de lidar com os perigos dos malwares que infectam os dispositivos. Este projeto visa o estudo e aplicação de soluções utilizando a Inteligência Artificial para diminuir a propagação assustadora de malwares em dispositivos móveis. Para isso, foram desenvolvidos um classificador de padrões baseado na técnica Support Vector Machines utilizando o software Matlab e uma aplicação para dispositivos Android que, utilizando o classificador, consiga identificar por meio da análise de intent-filters pertencentes a um Manifesto Android se uma aplicação é considerada como um malware ou benigna. Após a finalização do projeto, os resultados se mostraram promissores visto as análises que os avaliadores (acurácia, curvas ROC e matriz de confusão) apresentaram.

Palavras-chave: Dispositivos móveis; Inteligência Artificial; Support Vector Machines; Malwares; Classificador de padrões; Intent-filters; Matlab; Android.

Abstract

The evolution of technology in mobile devices has made it possible to offer many facilities in today's world that have attracted malicious individuals into this scenario, making it extremely difficult to deal with the dangers of malware that infect devices. This project aims at the study and application of solutions using Artificial Intelligence to reduce the frightening spread of malware on mobile devices. For this, a standards classifier was developed based on the Support Vector Machines technique using Matlab software and an application for Android devices that, using the classifier, can identify by means of the analysis of intent-filters belonging to an Android Manifest if an application is considered as malware or benign. After the project was finished, the results were promising, considering the analyzes that the evaluators (accuracy, ROC curves and confusion matrix) presented.

Keywords: Mobile Devices; Artificial Intelligence; Support Vector Machines; Malwares; Pattern classifier; Intent-filters; Matlab; Android.

Lista de figuras

Lista de tabelas

Lista de abreviaturas e siglas

SVM	Support Vector Machines
KNN	K-nearest-neighbors
IDE	Integrated Development Environment
JNI	Java Native Interface
IA	Inteligência Artificial
RNA	Rede Neural Artificial

Sumário

1 Introdução

A evolução e popularização dos dispositivos móveis tornou possível realizar tarefas que antes só podiam ser feitas fisicamente ou nos computadores domésticos, como acessar bancos, navegar na internet, enviar e receber mensagens. Assim, do mesmo modo que os computadores, os dispositivos móveis estão sujeitos a diversas ameaças de segurança, como os *malwares*. Embora não seja preocupação de grande parte da população que utiliza smartphones e outros dispositivos, um aparelho infectado por um malware pode causar grandes danos ao seu dono, como ter sua conta bancária roubada, tornar o dispositivo totalmente inutilizável e roubar informações privadas (PENG; YU; YANG, 2013).

Shabtai et al. (2011) afirmam que diferente dos computadores, o poder de processamento e a capacidade energética é muito menor nestes dispositivos, o que gerou uma dificuldade muito maior no quesito segurança em relação aos computadores convencionais, já que a maioria das soluções para computadores (antivírus) são inaplicáveis para uso em dispositivos móveis em razão do grande consumo de memória e energia. Visto estas dificuldades diversas, se viu necessário buscar novas técnicas para detectores de malware em dispositivos móveis. A evolução tecnológica é constante e ocorre a passos largos, assim como o invento dos computadores e smartphones, também surgiu a Inteligência artificial, que pode se definir como "a capacidade das máquinas de pensarem como seres humanos – de terem o poder de aprender, raciocinar, perceber, deliberar e decidir de forma racional e inteligente" (SALESFORCE, 2018).

Dentro da IA surgiu um método de análise de dados que automatiza o desenvolvimento de modelos analíticos, chamado *Machine Learning*. Este método utiliza algoritmos que quando expostos a determinados dados eles aprendem de maneira interativa, podendo até mesmo se adaptar quando expostos a dados novos. Isto torna possível criar máquinas que aprendam com os cálculos anteriores e forneçam resultados de alta confiabilidade (SAS, 2018).

Este projeto tem como foco uma aplicação para sistemas Android que, por meio de um método de aprendizado de máquina, seja capaz identificar se o dispositivo está infectado por um aplicativo malicioso analisando informações do Arquivo Manifesto (explicado na seção 2) de cada aplicação instalada no dispositivo.

1.1 PROBLEMA

A segurança na área de dispositivos móveis não acompanhou a sua alta produção e comercialização, muitas vezes por certa negligência por parte de algumas empresas

que, ao tentar diminuir custos para tornar o produto mais acessível, abrem mão também de aspectos importantes como a segurança e por consequência, acabam abrindo espaço para indivíduos com intenções maliciosas que infectam milhares de aparelhos por meio da internet, causando enorme prejuízo aos seus donos.

Segundo a TIINSIDE (2017), uma pesquisa feita pela empresa Avast durante o segundo trimestre de 2017 mostrou uma quantidade de 1,2 milhão até 1,7 milhão de ciberataques por mês e uma média de 788 variações de vírus, sendo esta média 22,2% maior do que no ano de 2016 durante o mesmo período. A quantidade de ataques por vírus em smartphones indica que as abordagens de segurança presentes nestes dispositivos não estão sendo tão bem-sucedidas, portanto explorar novas soluções se torna necessário, sendo uma delas a integração com técnicas de IA para fins de detecção dos malwares naqueles dispositivos que estiverem infectados.

1.2 JUSTIFICATIVA

Segundo uma análise feita em 2017 pela empresa russa de segurança digital Kaspersky, o Brasil se encontra em oitavo em uma lista de 10 países que são os mais atacados por vírus para smartphone no mundo. Somente entre os meses de janeiro e março de 2017, a empresa identificou 1,333 milhão de ataques de vírus em smartphones no mundo (SCRIVANO, 2017).

Portanto, observando o grande desafio que ferramentas de detecção de intrusão e *malwares* encontram e a massiva quantidade de ataques que ocorrem todos os dias contra dispositivos móveis, é necessário incentivar projetos que tenham como objetivo encontrar outras maneiras ou meios eficazes de detectar malwares nestes aparelhos utilizando técnicas inteligentes.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Desenvolver um *software* para smartphones que, utilizando a técnica de Inteligência Artificial *Support Vector Machines* (explicada na seção 2), consiga identificar por meio da extração e análise dos *intent-filters* do arquivo manifesto de Aplicações Android, se o dispositivo se encontra infectado por algum *malware*.

1.3.2 Objetivos Específicos

a) Estudar aprendizagem de máquina e a metodologias para classificação de dados;

b) Buscar uma base de dados;

c) Desenvolver o modelo classificador no Matlab;

d) Desenvolver a aplicação Android com integração do modelo gerado.

Nos próximos capítulos, será feita uma fundamentação teórica com os conceitos básicos para o entendimento do projeto, detalhes sobre cada etapa do desenvolvimento e os resultados obtidos.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados e descritos conceitos abordados ao longo do trabalho, pertencentes ao domínio de dispositivos Android, segurança em smartphones e técnicas inteligentes de classificação de padrões.

2.1 MANIFESTO ANDROID

O Manifesto Android é um arquivo texto que é responsável por apresentar todo tipo de informação importante sobre o aplicativo ao sistema Android. As informações descritas pelo Manifesto são necessárias para que o sistema consiga executar a aplicação com suas devidas permissões e configurações (CORDEIRO, 2016).

Neste arquivo, além de outros componentes do aplicativo como atividades e permissões, são também listados os *intent-filters* aceitos pela aplicação, que são um dos alvos de estudo deste trabalho. Os *intent-filters* são expressões no Arquivo Manifesto que especificam o tipo de *intents* que o componente de uma aplicação deseja receber. *Intent* é um objeto de mensagem que é utilizado para solicitar uma ação do mesmo aplicativo ou de outro que esteja instalado no dispositivo (DEVELOPERS, 2018).

Um exemplo de *intent-filter* retirado de um Arquivo Manifesto pode ser observado na Figura 1.

Figura 1 – Exemplo de *Intent-Filter*

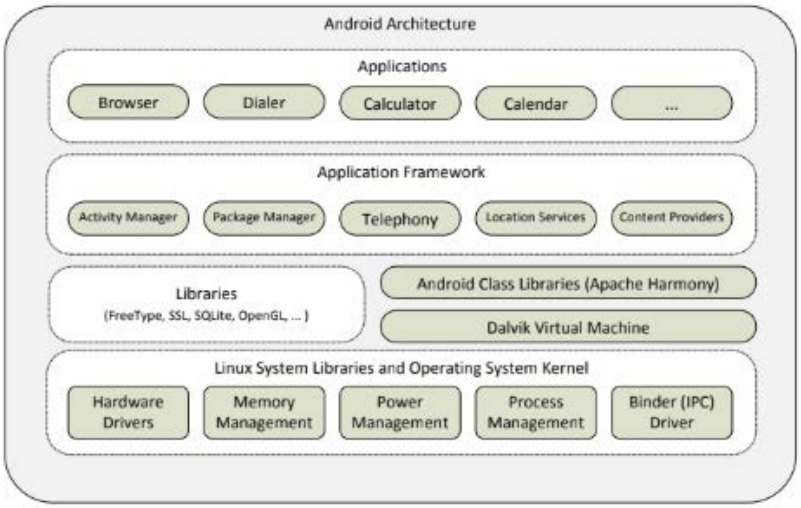
```
01. <intent-filter>
02.     <action android:name="android.intent.action.EDIT" />
03.     <action android:name="android.intent.action.INSERT" />
04.     <action android:name="android.intent.action.DELETE" />
05. </intent-filter>
```

Fonte: Cordeiro (2016)

2.2 SEGURANÇA EM SMARTPHONES ANDROID

Segundo Khan et al. (2012), o sistema *Android* tornou-se uma das plataformas open source mais destacadas para dispositivos móveis. A plataforma está longe de ser somente um sistema operacional, sendo na verdade uma "pilha de software" completa, contando com sistema, *middleware* e diversos aplicativos internos. Sua arquitetura é composta de diferentes camadas interativas, podendo ser visualizada na Figura 2.

Figura 2 – Componentes da arquitetura Android



Fonte: Khan et al. (2012)

O sistema *Android* conta com alguns mecanismos de segurança contra aplicativos mal-intencionados, como o mecanismo *Sandboxing*. Este impede que uma aplicação acesse informações confidenciais localizadas no sistema, e da mesma forma, informações armazenadas no espaço privado de outra aplicação também não podem ser acessadas. Além disso, também impede acessos não autorizados a recursos de *hardware*, como câmera, GPS e comunicação utilizando a rede. O *Android* também conta com o mecanismo de permissão de aplicativos com o objetivo de impor restrições em operações específicas que aplicativos desejam executar. Então, no momento da instalação e durante a execução do aplicativo, este irá requerer que o usuário lhe forneça a permissão para determinadas tarefas, sendo assim de responsabilidade do usuário confiar ou não na aplicação. Se permitido, o aplicativo terá acesso irrestrito às operações que lhe foi concedido permissão para acessar (KHAN et al., 2012).

O mecanismo de comunicação utilizado pelo *Android* para troca de informações entre componentes de uma mesma aplicação ou aplicativos diferentes é chamado de *intents*. De acordo com Khan et al. (2012), esse mecanismo torna possível combinar funcionalidades de diferentes aplicações.

Existem apenas dois tipos de *intents*, chamadas de explícitas e implícitas. As explícitas especificam qual componente deve ser iniciado pelo seu nome, sendo este tipo de intent normalmente utilizado para iniciar um componente no próprio aplicativo. Já as implícitas não especificam o componente a ser iniciado, apenas declaram uma ação a ser executada, tornando possível assim que outro aplicativo processe esta ação (KHAN et al., 2012).

Khan et al. (2012) afirmam que um aplicativo malicioso pode ser capaz de in-

terceptar uma *intent* implícita declarando um *intent-filter* no *Android Manifest* com as ações, categorias e dados listadas na *intent*. Isso gera uma situação em que pode ocorrer o recebimento não autorizado de *intent*, possibilitando que a aplicação obtenha acesso a todos os dados que foram enviados pela *intent* correspondente, resultando em um sequestro de atividade. Citando outra situação, uma aplicação mal-intencionada pode tentar enganar o usuário para que este instale um outro aplicativo que colabore com o primeiro para obter recursos adicionais. Assim, é solicitado ao usuário dois conjuntos diferentes de permissões, sendo um de cada aplicação, podendo parecer inofensivos separados, porém, sem o conhecimento do usuário, as aplicações compartilham suas funções por meio da troca de *intents*, podendo vir a prejudicar o dispositivo e seu usúario.

2.3 REDES NEURAIS ARTIFICIAIS

As redes neurais artificiais são modelos criados computacionalmente inspirados no sistema nervoso. Possuem capacidade de adquirir conhecimento baseando-se em informações fornecidas a elas e podem ser definidas como "Conjunto de unidades de processamento, caracterizadas por neurônios artificiais, que são interligados por um grande número de interconexões (sinapses artificiais)..." (NUNES; HERNANE; ANDRADE, 2010, p. 24).

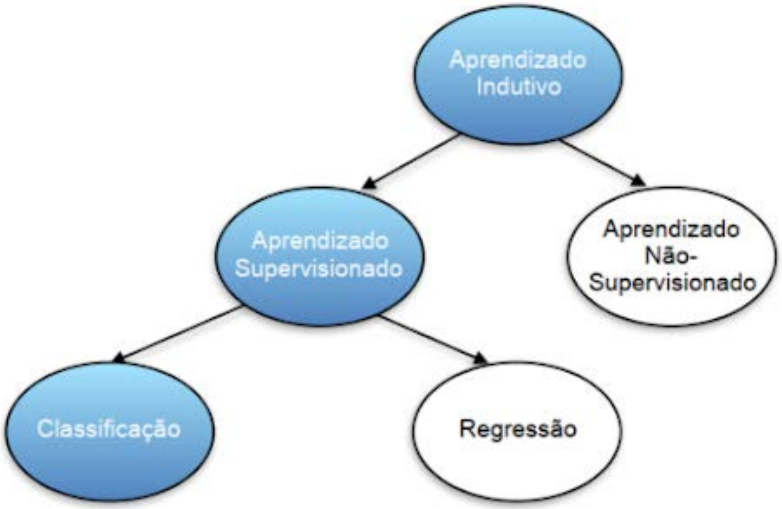
Nunes, Hernane e Andrade (2010) explicam que a criação de uma RNA depende de um processo de treinamento que é feito aplicando passos ordenados necessários para sintonização de pesos sinápticos e limiares dos neurônios da rede, de maneira que o resultado final seja a generalização das soluções que serão produzidas em suas saídas. O conjunto de passos ordenados citado anteriormente é chamado de algoritmo de aprendizagem. A aplicação dos passos fará a rede ser capaz de determinar e extrair características do sistema a ser mapeado utilizando o conjunto de amostragem fornecido para o processo treinamento.

O conjunto de amostras é dividido em duas partes, sendo uma de treinamento e a outra de teste. A primeira é composta de 60 a 90% do conjunto total e será utilizada no processo de aprendizagem da rede. Já a segunda, que compreende o restante da amostragem total, será para avaliar se os aspectos referentes à generalização de soluções já estão em níveis satisfatórios (NUNES; HERNANE; ANDRADE, 2010).

O treinamento de uma RNA possui dois tipos mais comumente utilizados, sendo eles: supervisionado e não-supervisionado. Neste trabalho, será abordado somente o treinamento supervisionado, que consiste basicamente em possuir a saída esperada para cada amostra de entrada. Ou seja, todas as amostras que serão utilizadas no treinamento da rede devem consistir em entrada e sua respectiva saída. Assim, é "como se houvesse um professor ensinando para a rede qual a resposta correta para cada amostra apresentada em suas entradas." (NUNES; HERNANE; ANDRADE, 2010, p. 52). A Figura 3 mostra

a hierarquia do aprendizado.

Figura 3 – Hierarquia do Aprendizado



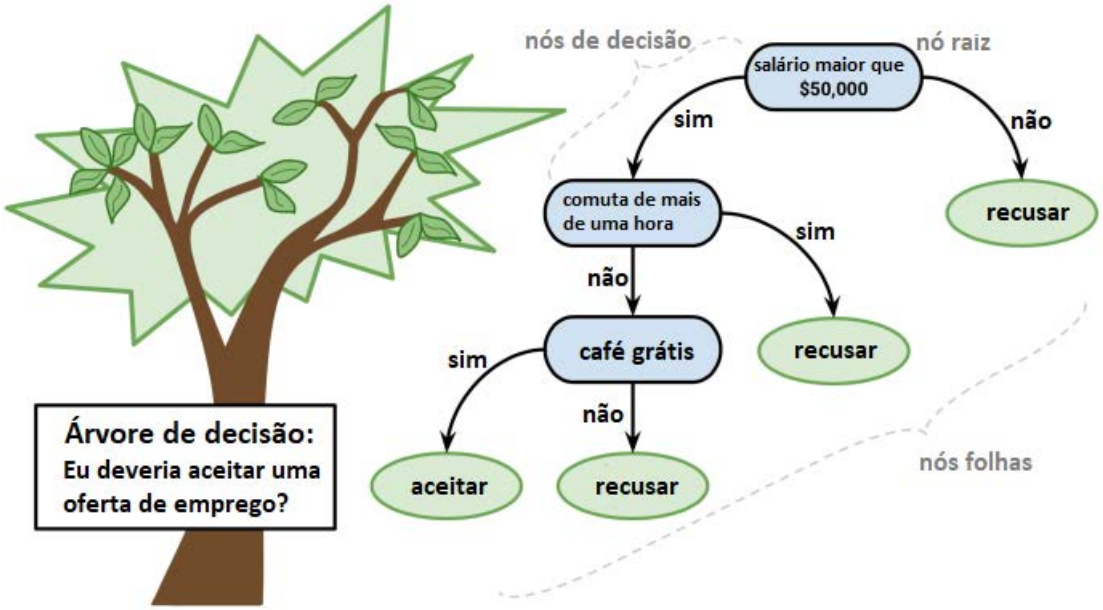
Fonte: (REZENDE, 2003 apud NAKATANI, 2017, p. 16)

2.4 TÉCNICAS DE CLASSIFICAÇÃO

Conforme citado anteriormente, este trabalho irá explorar apenas técnicas relacionadas ao treinamento supervisionado. Dentre os algoritmos de classificação mais utilizados e conhecidos, foram selecionados três para serem abordados neste projeto, sendo um deles o utilizado na etapa de implementação.

A primeira técnica a ser discutido é a Árvore de decisão. Basicamente, a partir de um conjunto de atributos e suas respectivas classes, a árvore irá estabelecer uma sequência de regras que serão utilizadas para a classificação dos dados. A vantagem desta técnica é que possui visualização simples e não requer muito trabalho na preparação dos dados. Porém, a árvore de decisão pode criar árvores tão complexas que não conseguem executar uma generalização satisfatória, além de poderem apresentar instabilidade dependendo das variações nos dados (GARG, 2018). A Figura 4 mostra o funcionamento do algoritmo usando um exemplo.

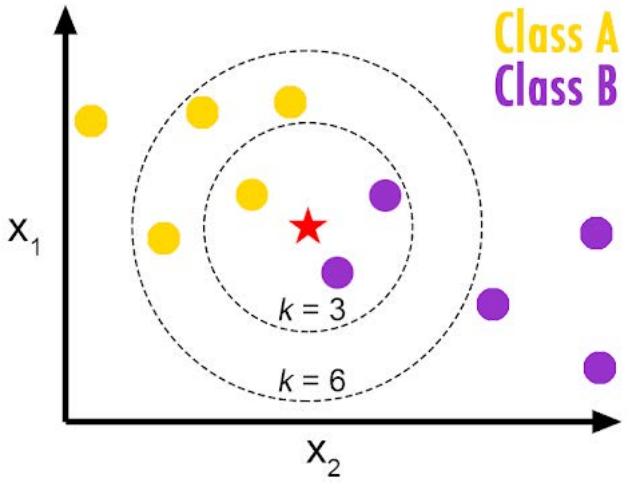
Figura 4 – Exemplo de funcionamento do Árvore de Decisão



Fonte: Korbut (2017)

A próxima técnica se chama *K-nearest-neighbors* ou *KNN*. Sua classificação funciona da seguinte forma: por meio de pontos rotulados, ele tenta utilizá-los para aprender a rotular outros pontos, ou seja, quando o algoritmo vai rotular um novo ponto, ele observa os K vizinhos (quantidade de pontos) que estão mais pertos do novo ponto e estes fazem uma votação de maioria simples envolvendo seus rótulos que define o rótulo do novo ponto. A vantagem desta técnica está na simplicidade da implementação e na eficácia caso os dados de treinamento forem muito grandes. Como desvantagem, o *KNN* apresenta um processo de aprendizagem lenta, é necessário determinar o valor de K e possui um alto custo computacional (GARG, 2018). A Figura 5 mostra o funcionamento do algoritmo usando um exemplo.

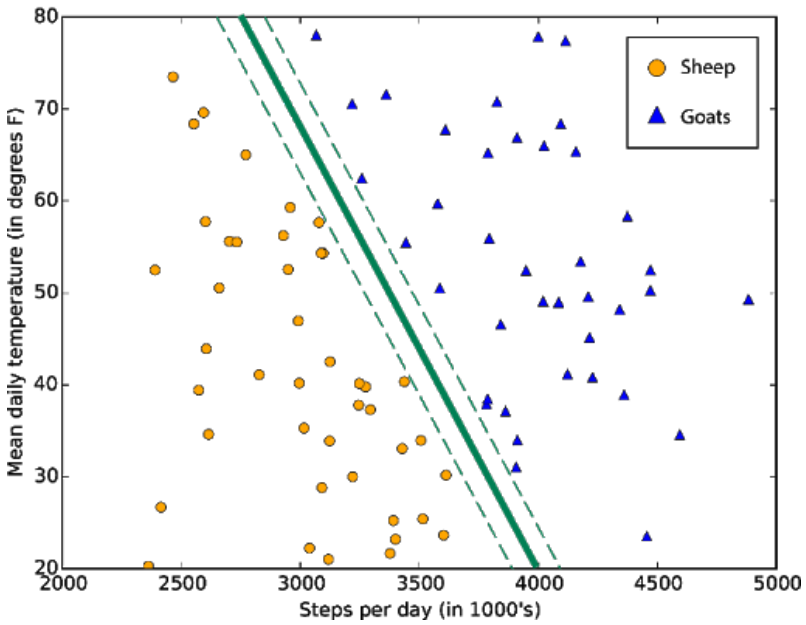
Figura 5 – Exemplo de funcionamento do *K-nearest-neighbors*



Fonte: Sawla (2018)

A terceira e última técnica a ser abordada é a *Support Vector Machines* (*SVM*). Segundo Lorena e Carvalho (2007) os *SVMs* fazem uma separação dos dados de treinamento por meio de uma lacuna o mais ampla possível. Assim, novos exemplos são colocados no mesmo espaço e categorizados baseando-se em qual lado da lacuna eles pertencem. A vantagem desta técnica é que são eficientes em termos de uso de memória e são robustas quando os dados possuem grande dimensão. Algumas das limitações dessa técnica são as dificuldades de interpretar o modelo obtido. A Figura 6 mostra o funcionamento do algoritmo usando um exemplo.

Figura 6 – Exemplo de *Support Vector Machines*



Fonte: Ericson et al. (2018)

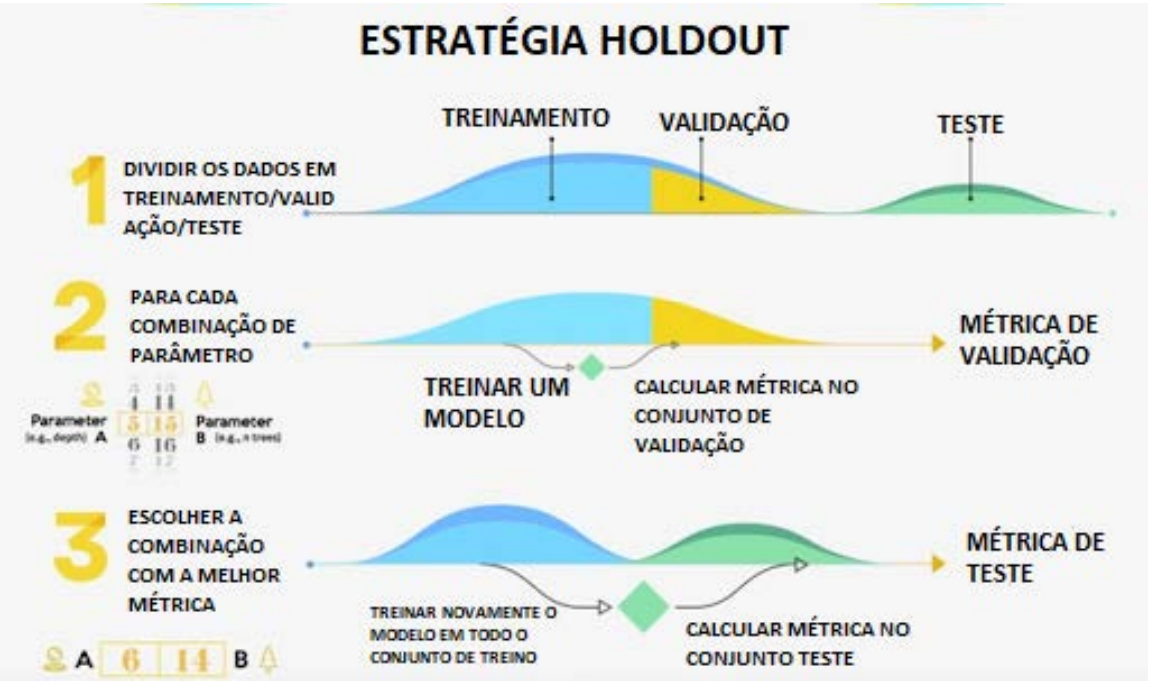
2.5 VALIDAÇÃO

O "processo de decidir se os resultados numéricos que quantificam as relações hipotetizadas entre variáveis são aceitáveis como descrições dos dados, é conhecido como validação." (GUPTA, 2017, p. 1).

Gupta (2017) enfatiza que a validação é necessária para verificar a estabilidade do modelo de aprendizado, ou seja, uma garantia de que o modelo tenha pego a maioria dos padrões dos dados corretamente, não captando muito ruído e garantindo que o modelo não esteja super-ajustando ou sub-ajustando os dados. Assim, é utiliza-se a validação cruzada, ao qual possui dois métodos mais conhecidos, um chamado *Holdout* e outro *K-fold Cross Validation*.

A estratégia *Holdout* consiste em remover uma parte dos dados que seriam utilizados para o treinamento e usá-los como teste, obtendo assim previsões do modelo gerado a partir do restante dos dados, o que gera uma estimativa de erro que informa como o modelo gerado está se comportando quando exposto a dados não vistos (conjunto de validação). Contudo, apesar de ser um método melhorado em relação a uma validação tradicional, ele ainda possui problemas relacionados à altas variações. Isso se deve pois ao reduzir os dados de treinamento, aumentam-se os riscos de perda de padrões que seriam importantes para o modelo (GUPTA, 2017). É possível observar o funcionamento da estratégia *Holdout* na Figura 7.

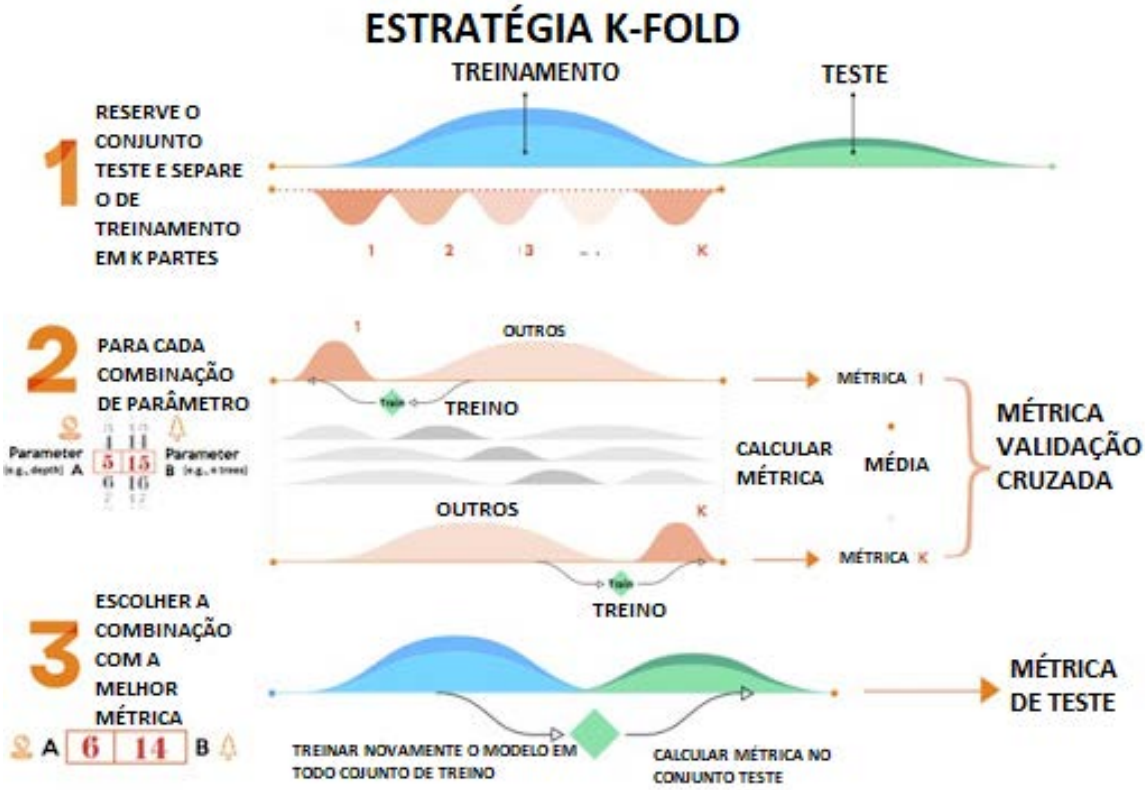
Figura 7 – Exemplo de funcionamento do *Holdout*



Fonte: Kelley (2017)

O *K-fold Cross Validation* faz exatamente o que é necessário para superar os problemas do método *Holdout*, ou seja, fornece uma ampla quantidade de dados para o treinamento e também para validação. Sua estratégia consiste em dividir os dados em K subconjuntos e então, repete-se o método *Holdout* K vezes, sendo que a cada repetição um dos subconjuntos é usado para teste e os demais para treinamento. O valor de K pode ser qualquer número positivo, porém como regra geral e evidência empírica, atribui-se o valor 5 ou 10. A única desvantagem em relação a este método é que ele possui maior custo computacional do que em relação ao método *Holdout* (GUPTA, 2017). É possível observar o funcionamento da estratégia *K-fold Cross Validation* na Figura 8.

Figura 8 – Exemplo de funcionamento do *K-fold Cross Validation*



Fonte: Kelley (2017)

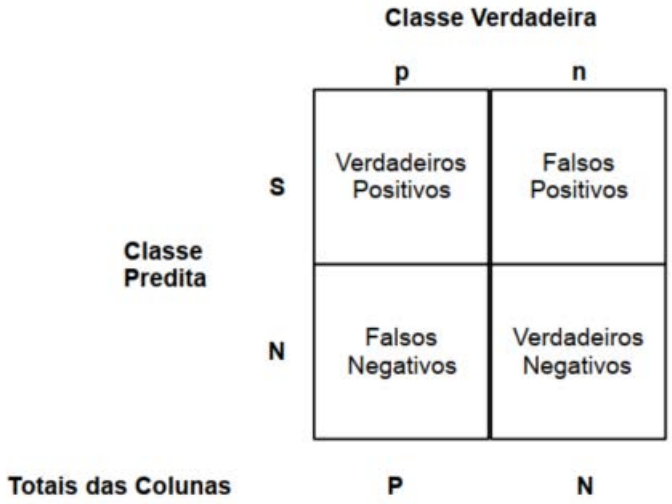
2.6 MATRIZ DE CONFUSÃO

Dado um determinado modelo, a matriz de confusão tem como objetivo mostrar o número de classificações corretas e o número de predições para cada classe. Os acertos de cada classe ficam localizados na diagonal principal da matriz (REZENDE, 2003 apud NAKATANI, 2017, p. 19).

A Figura 9 exemplifica uma matriz de confusão de duas classes (saída binária), mostrando as quantidades de falsos positivos (FP), verdadeiros positivos (TP), falsos

negativos (FN) e verdadeiros negativos (TN), além dos totais de positivos (P) e negativos (N) (FAWCETT, 2005 apud NAKATANI, 2017, p. 20).

Figura 9 – Matriz de confusão com duas classes



Fonte: (FAWCETT, 2005 apud NAKATANI, 2017, p. 20)

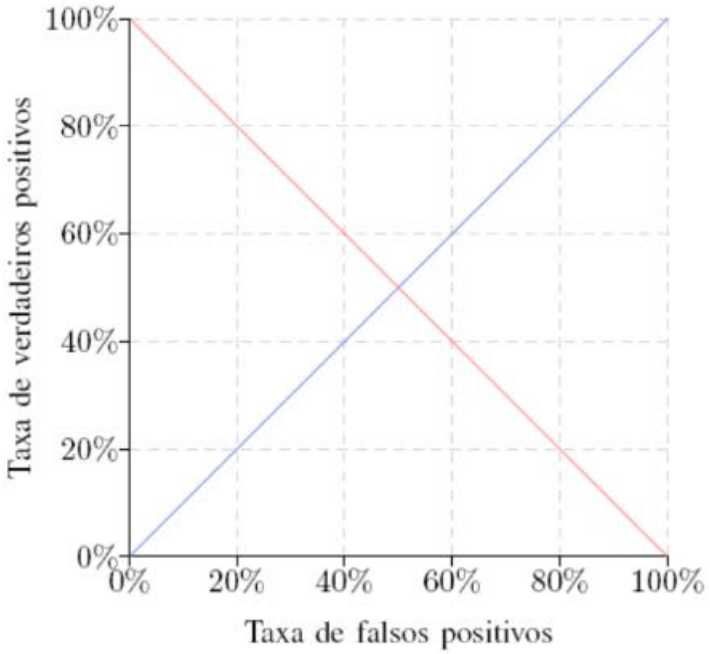
2.7 CURVA ROC

O gráfico ROC tem como base a probabilidade de detecção (taxa de verdadeiros positivos) e na probabilidade de falsos alarmes (taxa de falsos positivos). O gráfico é construído plotando os verdadeiros positivos no eixo das abscissas e os falsos positivos no eixo das ordenadas (PRATI; BATISTA; MONARD, 2008).

No gráfico ROC representa-se o modelo de classificação por meio de um ponto em seu espaço. A obtenção deste ponto é dada pelo cálculo da taxa de verdadeiros e falsos positivos a partir da sua matriz de contingência (PRATI; BATISTA; MONARD, 2008).

A Figura 10 representa um gráfico ROC. Pontos que estão localizados à esquerda da diagonal ascendente representam modelos que tem um bom desempenho (melhor que o aleatório) e pontos pertencentes à direita são modelos piores que o aleatório. Além disso, é possível observar que o ponto (100%, 0) seria o pior modelo e o ponto (0, 100%) seria o modelo ideal.

Figura 10 – Espaço ROC



Fonte: Prati, Batista e Monard (2008)

2.8 TRABALHOS CORRELATOS

Dentre os trabalhos existentes que correlacionam a detecção de malwares utilizando intents com a inteligência artificial, podem-se destacar os três descritos a seguir.

O primeiro exemplo é o artigo *Investigating the Android Intents and Permissions for Malware detection*. Os autores propõem uma análise baseada na combinação de permissões e intents, assim criam um classificador utilizando um algoritmo chamado Naive Bayes para detectar se a aplicação se trata de um malware ou não (IDREES; RAJARAJAN, 2014).

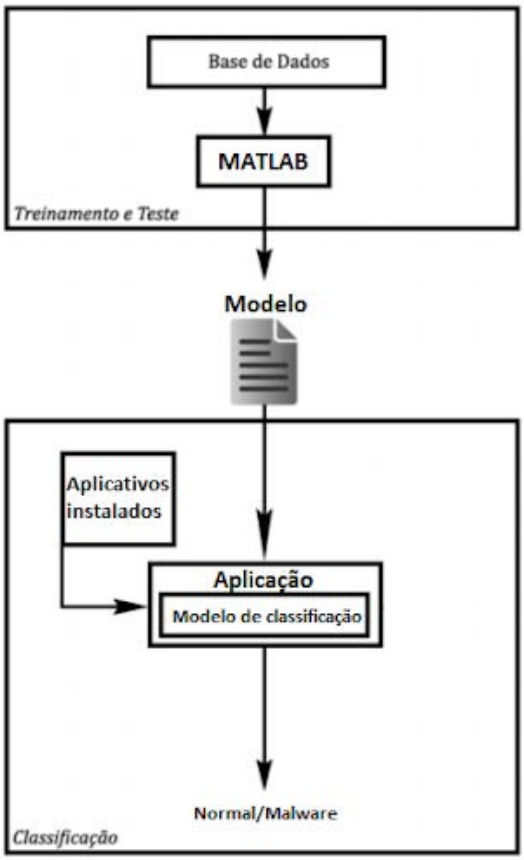
Outro trabalho se trata de um sistema chamado *AndroDialysis* que tem como foco analisar *intents* implícitas e explícitas utilizando um algoritmo de aprendizagem chamado *Bayesian Network* e como validação o *k-fold cross validation* (COSTA, 2016).

O sistema *DroidMat* promove uma análise baseada em permissões, *intents*, chamadas de API e implementação de componentes utilizando o algoritmo *KNN* para realizar a classificação em aplicativo benigno ou malicioso (WU et al., 2012).

3 DESENVOLVIMENTO

Este capítulo tem como objetivo descrever o desenvolvimento do projeto proposto neste trabalho conforme o esboço na Figura 11, sendo detalhado nas próximas seções que possuem como assuntos o método de pesquisa utilizado, a base de dados utilizada, a maneira com que o modelo classificador foi construído, as ferramentas, linguagens utilizadas e as etapas do desenvolvimento da aplicação.

Figura 11 – Esquema do projeto



Fonte: Elaborada pelo autor

3.1 MÉTODO DE PESQUISA

Para o desenvolvimento da aplicação proposta por este projeto decidiu-se utilizar a linguagem *JAVA* em conjunto com a *IDE Android Studio*, o qual possui uma melhor integração com o sistema Android, pois é sua *IDE* de desenvolvimento oficial. Outra linguagem utilizada para o desenvolvimento do modelo de classificação foi a *Matlab* em

conjunto com os *toolboxes Classification Learner e MATLAB Coder* que foram instalados e estão disponíveis na *IDE Matlab* versão 2017b.

O projeto foi dividido em três etapas principais, sendo elas: a manipulação e adequação da base de dados; o treinamento e exportação do modelo utilizando o Matlab; por fim, a aplicação Android. Estas etapas serão descritas com detalhes nas próximas seções.

3.2 BASE DE DADOS

A base de dados utilizada neste projeto se chama *Malgenome Project*. Esta base foi criada em um projeto chamado *Android Malware Genome Project*. Seus criadores catalogaram mais de mil e duzentos malwares e mais de duas mil e quinhentas aplicações benignas. As aplicações foram caracterizadas em diversos aspectos, sendo um deles os *intent-filters*, que foram utilizados para o treinamento do modelo de classificação deste projeto (ZHOU; JIANG, 2012).

Utilizando o Matlab, foi feita a preparação dos dados da base selecionando as características que seriam utilizadas para a criação do modelo classificador e removendo características desnecessárias ao estudo proposto por este trabalho.

A base *Malgenome* contém 26 tipos de *intent-filters*, o qual foram todos selecionados para utilização no classificador. Os nomes dos *intent-filters* estão listados na Figura 12.

Figura 12 – *Intent-filters* da base *Malgenome*

android.intent.action.BATTERY_LOW
android.intent.action.BATTERY_OKAY
android.intent.action.BOOT_COMPLETED
android.intent.action.CALL
android.intent.action.CALL_BUTTON
android.intent.action.CAMERA_BUTTON
android.intent.action.NEW_OUTGOING_CALL
android.intent.action.PACKAGE_ADDED
android.intent.action.PACKAGE_CHANGED
android.intent.action.PACKAGE_DATA_CLEARED
android.intent.action.PACKAGE_REMOVED
android.intent.action.PACKAGE_REPLACED
android.intent.action.PACKAGE_RESTARTED
android.intent.action.ACTION_POWER_CONNECTED
android.intent.action.ACTION_POWER_DISCONNECTED
android.intent.action.REBOOT
android.intent.action.RUN
android.intent.action.SCREEN_OFF
android.intent.action.SCREEN_ON
android.intent.action.SEND
android.intent.action.SEND_MULTIPLE
android.intent.action.SENDTO
android.intent.action.SET_WALLPAPER
android.intent.action.SHUTDOWN
android.intent.action.TIME_SET
android.intent.action.TIMEZONE_CHANGED

Fonte: Elaborada pelo autor

3.3 MODELO CLASSIFICADOR

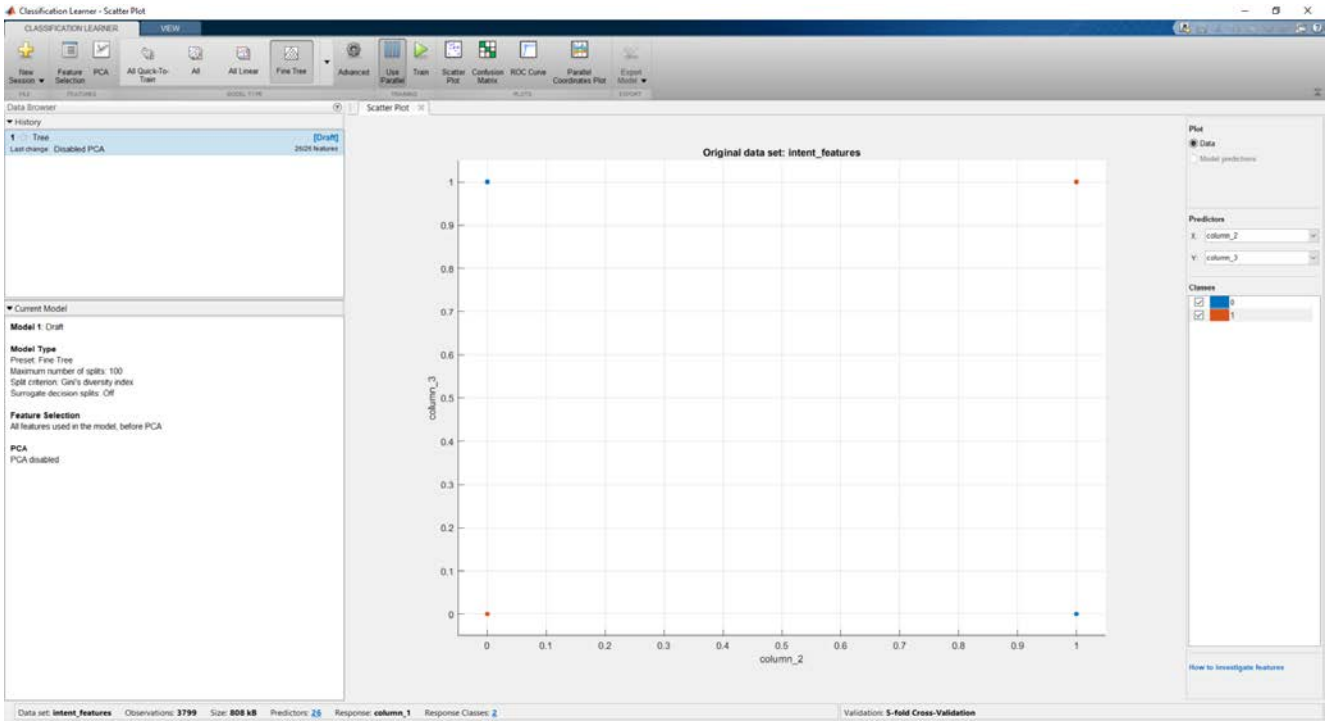
3.3.1 Toolbox Classification Learner

O treinamento do modelo classificador foi feito utilizando o *toolbox Classification Learner* que está demonstrado na figura 13. O *toolbox* possui diversos algoritmos de classificação implementados em que é possível fazer o treinamento da rede em qual algoritmo o usuário desejar, além disso também fica disponível ao usuário gerar gráficos e análises do modelo gerado como o gráfico ROC (descrito na seção 2.7) e a matriz de confusão (descrita na seção 2.6).

O algoritmo classificador determinado para este trabalho foi o *Support Vector Machines* devido ao seu bom desempenho em performance e em sua robustez ao tratar grandes quantidades de dados. Porém, para comparação de performance foram feitos treinamentos também nos algoritmos Árvore de decisão e *K-nearest-neighbors*. Os resultados de cada algoritmo se encontram no tópico de resultados (seção 3.5).

A interface do *Toolbox Classification Learner* se encontra na Figura 13

Figura 13 – *Toolbox Classification Learner*



Fonte: Elaborada pelo autor

O método de avaliação utilizado foi o *K-fold Cross Validation* com K igual a cinco. As configurações do *hardware* utilizado para realizar o treinamento do modelo foram: Processador Intel Core i7-6700HQ 2.60GHz, 8 GB de memoria RAM e NVIDIA GeForce

GTX 960M.

Após o treinamento do classificador foi necessário gerá-lo em outra linguagem para utilizá-lo de maneira externa ao Matlab, então foi utilizado o *toolbox MATLAB Coder* para exportar o modelo para a linguagem C.

3.3.2 Toolbox Matlab Coder

A geração do modelo classificador em linguagem C foi feita utilizando o *MATLAB Coder*. Sua utilização é bem intuitiva, pois basicamente o usuário deve inserir uma função escrita em linguagem *Matlab* e o arquivo do modelo. Após executar o *Toolbox* com estas entradas uma pasta é criada no local especificado com todos os arquivos em linguagem C necessários para executar o classificador externamente, não necessitando mais do Matlab.

3.4 APLICAÇÃO ANDROID

O aplicativo para sistema *Android* batizado de *Mal-Intentfier* foi totalmente desenvolvido na *IDE Android Studio* utilizando a linguagem *JAVA* e uma pequena parte em linguagem *Kotlin*.

A implementação foi dividida em cinco etapas que dependem do funcionamento da etapa anterior. Assim, cada etapa foi desenvolvida de maneira que ao final de cada uma seria possível obter uma versão onde os testes pudessem ser realizados. Estas etapas serão explicada detalhadamente nos próximos tópicos.

3.4.1 Lista de aplicações

O primeiro passo para iniciar o desenvolvimento foi obter e listar informações de todas as aplicações instaladas no dispositivo, extraindo seu nome e o nome de seu pacote de instalação. O sistema Android possui uma estrutura onde é possível identificar quais aplicações são próprias de seu sistema, portanto estas aplicações foram retiradas da lista, pois além de aumentar em grande número a quantidade de aplicações, não faria sentido avaliar aplicativos que pertencem ao próprio funcionamento do sistema.

3.4.2 Extração do Arquivo Manifesto

Como explicado na seção 2, cada aplicação possui um arquivo manifesto que engloba diversas informações a seu respeito, portanto foi necessário obter este arquivo de cada aplicativo da lista de aplicações instaladas. Para isso, foi feita uma adaptação de um código-fonte em linguagem *Kotlin* pertencente a uma aplicação *open-source* chamada *Apk Analyzer* que foi disponibilizada e teve seu uso autorizado pelo autor da aplicação

para utilização neste projeto. Assim, a partir do nome do pacote do aplicativo, foi possível extrair o arquivo manifesto das aplicações instaladas.

3.4.3 Extração de informações

Após obter o arquivo manifesto, foi necessário executar um *parser*[1] para selecionar as informações contidas no arquivo que seriam necessárias a este projeto, ou seja, identificar no manifesto se estes possuíam ou não os *intent-filters* que foram descritos na Figura 12.

O *parser* utilizado foi o *XMLPullParser* que é nativo do próprio *Android*, sendo eficiente e de utilização simplificada.

3.4.4 Incorporação do classificador

O *Android Studio* possui suporte a edição de arquivos em linguagem C, além de ferramentas de compilação para executar *scripts CMake*[2] e *ndk-build*[3]. Assim, seguindo as orientações de Pouillot (2015), criou-se um componente *JNI* que possibilita um código *Java* em execução chamar e ser chamado por códigos nativos escritos em outras linguagens, como a linguagem C. Posteriormente, utilizou-se o *ndk-build* do android para compilar o projeto que foi gerado pelo *Matlab* em linguagem C em uma biblioteca própria para poder ser utilizada pelo *Android*.

3.4.5 Classificação

A quinta e última etapa é responsável por enviar as informações coletadas dos aplicativos em forma de um vetor binário para ser analisado pelas funções dentro da biblioteca gerada com o classificador. Após a análise, é obtida a resposta se a aplicação foi ou não considerada como mal-intencionada. Em seguida, caso exista aplicações classificadas como *malwares*, uma tela é aberta mostrando os nomes destas aplicações, como mostra a Figura 18 que se encontra na seção de resultados.

3.4.6 Interface

A interface do *Mal-Intentfier* foi construída com o objetivo de ser o mais intuitiva possível para que o usuário não tenha dificuldades no uso da aplicação, podendo ser observada na Figura 14.

---

[1] analisador de textos

[2] família de ferramentas de plataforma aberta e código aberto projetadas para compilar softwares em linguagem C

[3] é um script do Android responsável por invocar um script de compilação

Figura 14 – Tela inicial da aplicação *Mal-Intentfier*

Fonte: Elaborada pelo autor

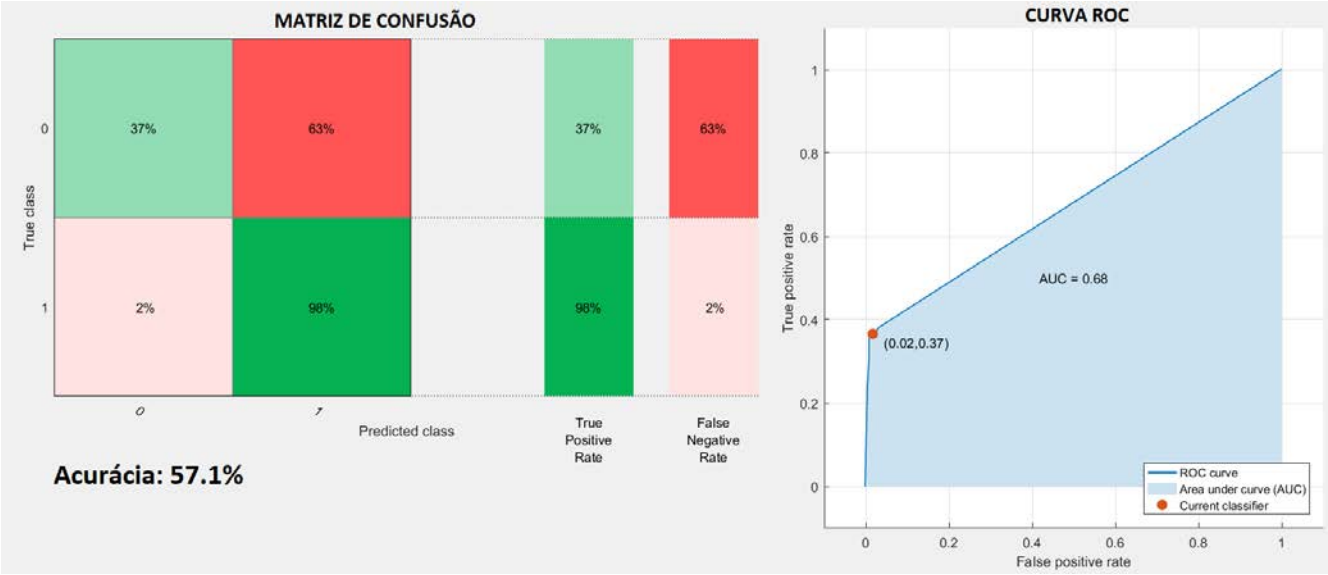
3.5 EXPERIMENTOS E RESULTADOS

Os treinamentos e testes de cada algoritmo proposto foram elaborados e geradas as análises estatísticas dos resultados. Após comparação, foi possível observar que os algoritmos *Support Vector Machines* e Árvore de Decisão apresentaram resultados melhores e próximos, enquanto o *KNN* teve um resultados inferiores. As análises estatísticas geradas foram a acurácia, gráfico ROC e a matriz de confusão que se encontram nas figuras 15, 16 e 17. Os tempos de duração de treinamento e predição de cada método se encontram na tabela 1.

Analisando as acurácias e as matrizes de confusão é possível observar que o *SVM* e a Árvore de Decisão tiveram resultados satisfatórios em relação às taxas de acurácia, verdadeiros positivos e também em relação aos falsos negativos, ambos ficando bem acima do *KNN* ao se comparar estes resultados.

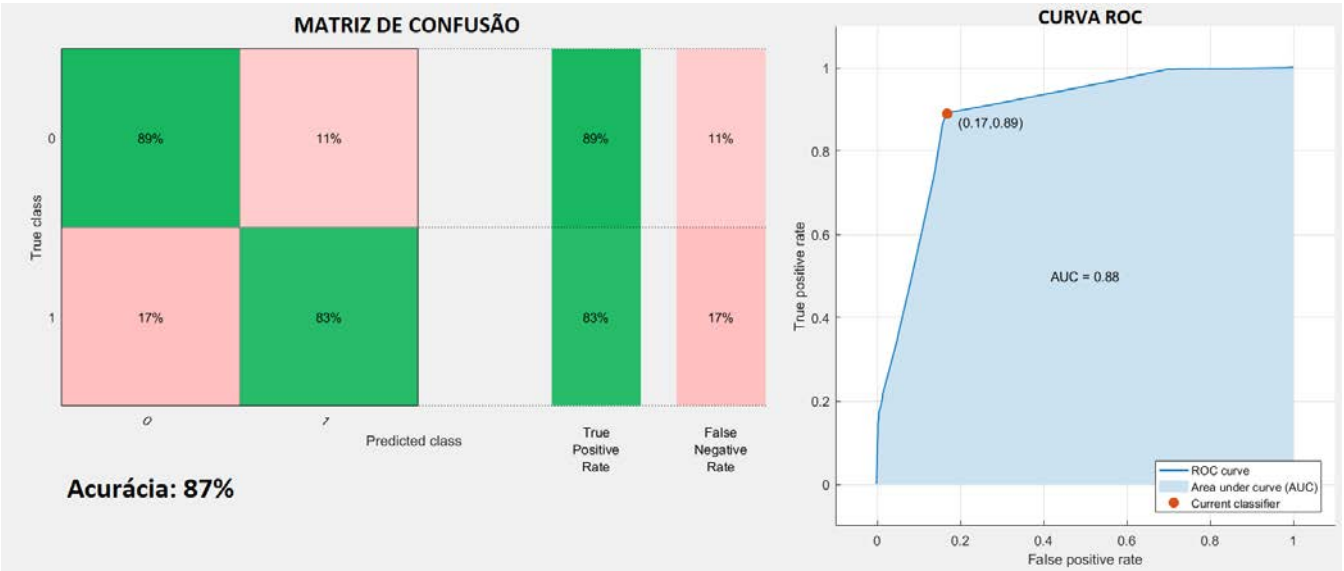
As áreas das curvas geradas pelas curvas ROC também mostram os ótimos resultados obtidos pelo *SVM* e pela Árvore de Decisão, além de reforçar a superioridade quando comparados ao *KNN*.

Figura 15 – Acurácia, Matriz de Confusão e Curva ROC - *K-nearest-neighbors*



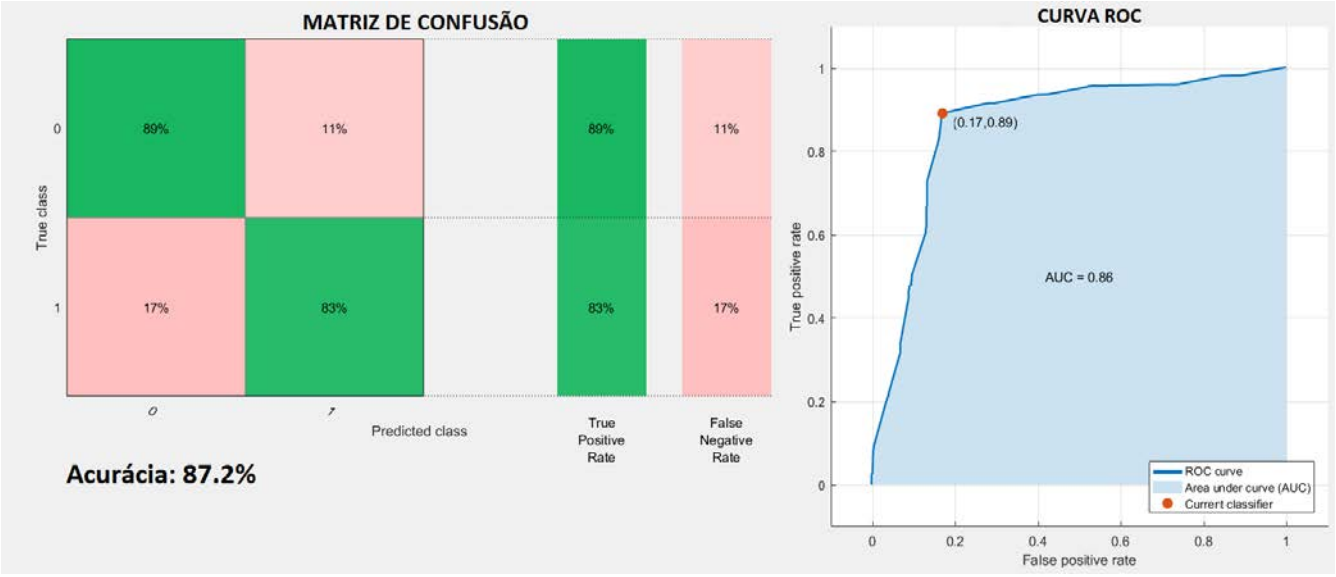
Fonte: Elaborada pelo autor

Figura 16 – Acurácia, Matriz de Confusão e Curva ROC - Árvore de Decisão



Fonte: Elaborada pelo autor

Figura 17 – Acurácia, Matriz de Confusão e Curva ROC - *Support Vector Machines*



Fonte: Elaborada pelo autor

Tabela 1 – Métricas de desempenho da classificação de padrões na base de dados *Malgenome*

Algoritmo de classificação	**Prediction Speed**	**Training Time**
KNN	~6200 obs/s	15.395 s
Árvore de decisão	~52000 obs/s	8.2044 s
SVM	~21000 obs/s	10.68 s

Fonte: Elaborada pelo autor

Devido à indisponibilidade de realizar testes em dispositivos certamente infectados por *malwares* e com o intuito demonstrar o funcionamento da aplicação com aplicativos mal-intencionados, foi feito uma função de teste dentro do aplicativo onde foram cadastrados informações de sessenta exemplos aleatórios de malwares da base *Malgenome* para serem analisados pelo classificador. O resultado obtido foi de que cinquenta e duas aplicações foram classificadas corretamente como um *malware* e apenas oito foram classificadas incorretamente como benignas, mostrando uma taxa de 86.66% de verdadeiros positivos.

Para realização de testes reais, a aplicação foi executada em quatro dispositivos diferentes pertencentes a usuários comuns. Os resultados fornecidos pelo aplicativo podem ser observados na tabela 2.

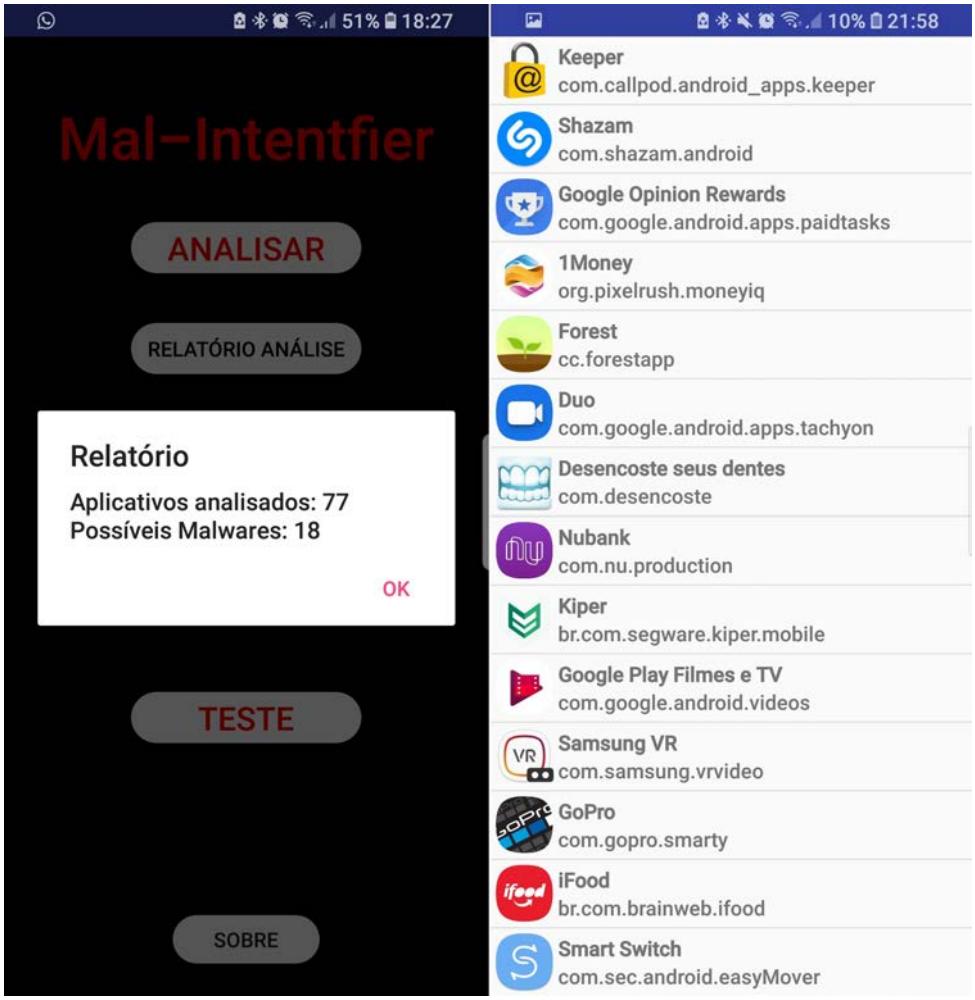
Tabela 2 – Resultados da execução da análise nos dispositivos

Dispositivos	**Quantidade de aplicativos analisados**	**Quantidade de possíveis *malwares***	**Tempo de execução (s)**
Samsung S8	77	18	70
Samsung S6	15	4	10
Samsung A5	22	2	20
Motorola G6	11	2	8

Fonte: Elaborada pelo autor

O tempo de execução da análise depende muito de cada dispositivo, devido a sua capacidade operacional e o número de aplicações instaladas. Como exemplo, na figura 18 estão os resultados mostrados pela aplicação referente ao primeiro dispositivo da tabela 2, onde ao lado esquerdo está o relatório mostrando a quantidade de aplicativos analisados e quantos foram considerados possíveis *malwares* e ao lado direito se encontram as informações destes aplicativos considerados possíveis *malwares*.

Figura 18 – Exemplo de execução da análise da aplicação em um dispositivo



Fonte: Elaborada pelo autor

4 CONCLUSÃO

Ao observar os resultados das análises do modelo de classificação como acurácia alta, nível alto na curva ROC, além de taxas de verdadeiros positivos altas e falsos negativos baixos, é possível notar um desempenho satisfatório na detecção de *malwares*.

Os resultados em dispositivos reais também são positivos, pois as taxas de falsos negativos também foram baixas. Já na simulação feita com informações da própria base, o aplicativo demonstrou-se muito eficiente na detecção.

É necessário ressaltar também a ótima velocidade na execução da análise, dado que no pior caso no qual o aparelho possuía uma alta quantidade de aplicações, o aplicativo *Mal-Intentfier* conseguiu classificar todos em aproximadamente um minuto.

Portanto, é possível concluir que a utilização de *intent-filters* aliado com o aprendizado de máquina é uma boa forma de detectar *malwares* e deve ser mais estudada, além de que talvez este projeto poderia ter exibido resultados ainda mais eficientes se o modelo pudesse ter sido treinado com uma base maior e com mais informações, afinal a base utilizada continha apenas 26 tipos de *intent-filters*.

5 TRABALHOS FUTUROS

Como sugestão de trabalhos futuros, pode-se considerar o aprimoramento dos resultados alcançados por este projeto. Para isto, é válido explorar uma base maior e contendo mais informações de *intent-filters*. Outra possibilidade seria utilizar os *intent-filters* para potencializar os resultados de um classificador baseado em outro tipo de abordagem como as permissões do *Android*, já que o classificador deste projeto mostrou resultados promissores.

Além disso, é necessário considerar a replicação dos experimentos realizados em dispositivos infectados em um ambiente controlado a fim de produzir resultados mais consistentes e realistas.

Referências

CORDEIRO, F. *Os Segredos do AndroidManifest*. 2016. Disponível em: <https://www-.androidpro.com.br/blog/desenvolvimento-android/androidmanifest>. Acesso em: 23 out. 2018.

COSTA, V. R. Androdialysis: Analysis of android intent effectiveness in malware detection. *Computers Security*, 2016. Disponível em: <https://www.sciencedirect.com-/science/article/pii/S0167404816301602>. Acesso em: 23 out. 2018.

DEVELOPERS, A. *Intents e filtro de intents*. 2018. Disponível em: <https:/-/developer.android.com/guide/components/intents-filters?hl=pt-br>. Acesso em: 23 out. 2018.

ERICSON, G.; ANTON, W.; JENKENS, A.; MARTENS, J.; ROHRER, B. *How to choose algorithms for Azure Machine Learning Studio*. Microsoft, 2018. Disponível em: <https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice>. Acesso em: 23 out. 2018.

GARG, R. *7 typer of Classification Algorithms*. Analytics India Magazine, 2018. Disponível em: <https://www.analyticsindiamag.com/7-types-classification-algorithms-/>. Acesso em: 23 out. 2018.

GUPTA, P. *Cross-Validation in Machine Learning*. Towards Data Science, 2017. Disponível em: <https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f>. Acesso em: 23 out. 2018.

IDREES, F.; RAJARAJAN, M. Investigating the android intents and permissions for malware detection. In: *2014 Seventh International Workshop on Selected Topics in Mobile and Wireless Computing*. [S.l.: s.n.], 2014. p. 354–358. Acesso em: 23 out. 2018.

KELLEY, R. *Making Predictive Models Robust: Holdout vs Cross-Validation*. KDnuggets, 2017. Disponível em: <https://www.kdnuggets.com/2017/08/dataiku-predictive-model-holdout-cross-validation.html>. Acesso em: 23 out. 2018.

KHAN, S.; NAUMAN, M.; OTHMAN, A. T.; MUSA, S. How secure is your smartphone: An analysis of smartphone security mechanisms. In: *2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic (CyberSec)*. [S.l.]: IEEE, 2012. p. 76–81. Acesso em: 23 out. 2018.

KORBUT, D. *Machine Learning Algorithms: Which One to Choose for Your Problem*. Stats and Bots, 2017. Disponível em: <https://blog.statsbot.co/machine-learning-algorithms-183cc73197c>. Acesso em: 23 out. 2018.

LORENA, A. C.; CARVALHO, A. C. P. L. F. Uma introdução às support vector machines. *RITA*, XIV, n. 2, p. 44–67, 2007. Acesso em: 23 out. 2018.

NAKATANI, M. H. *Avaliação de Classificadores para Detecção de Phising em E-mails*. 2017. Acesso em: 23 out. 2018.

NUNES, I.; HERNANE, D.; ANDRADE, R. *Redes Neurais Artificiais para engenharia e ciências aplicadas*. [S.l.]: Artliber, 2010. ISBN 978-85-88098-53-4.

PENG, S.; YU, S.; YANG, A. Smartphone malware and its propagation modeling: A survey. In: *IEEE Communications Surveys Tutorials, China*. [S.l.: s.n.], 2013. p. 925–941. Acesso em: 23 out. 2018.

POUILLOT, C. *Generating C Code from MATLAB for Use with Java and .NET Applications*. [S.l.], 2015. Disponível em: <https://www.mathworks.com/company-/newsletters/articles/generating-c-code-from-matlab-for-use-with-java-and-net-applications.html>. Acesso em: 23 out. 2018.

PRATI, R. C.; BATISTA, G. E. A. P. A.; MONARD, M. C. Curvas roc para avaliação de classificadores. *Revista IEEE America Latina*, IEEE, v. 6, n. 2, 2008. Acesso em: 23 out. 2018.

SALESFORCE. *Entenda os Principais Conceitos e o que e Inteligência Artificial*. 2018. Disponível em: <https://www.salesforce.com/br/products/einstein/ai-deep-dive>. Acesso em: 23 out. 2018.

SAS. *Machine Learning: O que e e qual sua importância?* 2018. Disponível em: <https://www.sas.com/pt_br/insights/analytics/machine-learning.html>. Acesso em: 23 out. 2018.

SAWLA, S. *K-Nearest Neighbors*. MEDIUM, 2018. Disponível em: <https://medium-.com/@srishtisawla/k-nearest-neighbors-f77f6ee6b7f5>. Acesso em: 23 out. 2018.

SCRIVANO, R. *Celulares brasileiros estão entre os mais atacados por virus no mundo*. 2017. Disponível em: <https://oglobo.globo.com/economia/celulares-brasileiros-estao-entre-os-mais-atacados-por-virus-no-mundo-21387847>. Acesso em: 23 out. 2018.

SHABTAI, A.; KANONOV, U.; ELOVICI, Y.; GLEZER, C.; WEISS, Y. Andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems, Alemanha*, Springer Science+Business Media, v. 38, p. 161–190, 2011. Acesso em: 23 out. 2018.

TIINSIDE. *Ataques a dispositivos moveis crescem 40%, diz Avast*. TIinside, 2017. Disponível em: <http://tiinside.com.br/tiinside/seguranca/mercado-seguranca/22-/09/2017/ataques-dispositivos-moveis-crescem-40-diz-avast>. Acesso em: 23 out. 2018.

WU, D.-J.; MAO, C.-H.; WEI, T.-E.; LEE, H.-M.; WU, K.-P. Droidmat: Android malware detection through manifest and api calls tracing. In: *2012 Seventh Asia Joint Conference on Information Security*. [S.l.: s.n.], 2012. p. 62–69. Acesso em: 23 out. 2018.

ZHOU, Y.; JIANG, X. Dissecting android malware: Characterization and evolution. In: *Proceedings of the 33rd IEEE Symposium on Security and Privacy*. [S.l.: s.n.], 2012. Acesso em: 23 out. 2018.